# Session 4: SQL Queries in Logic and Reports

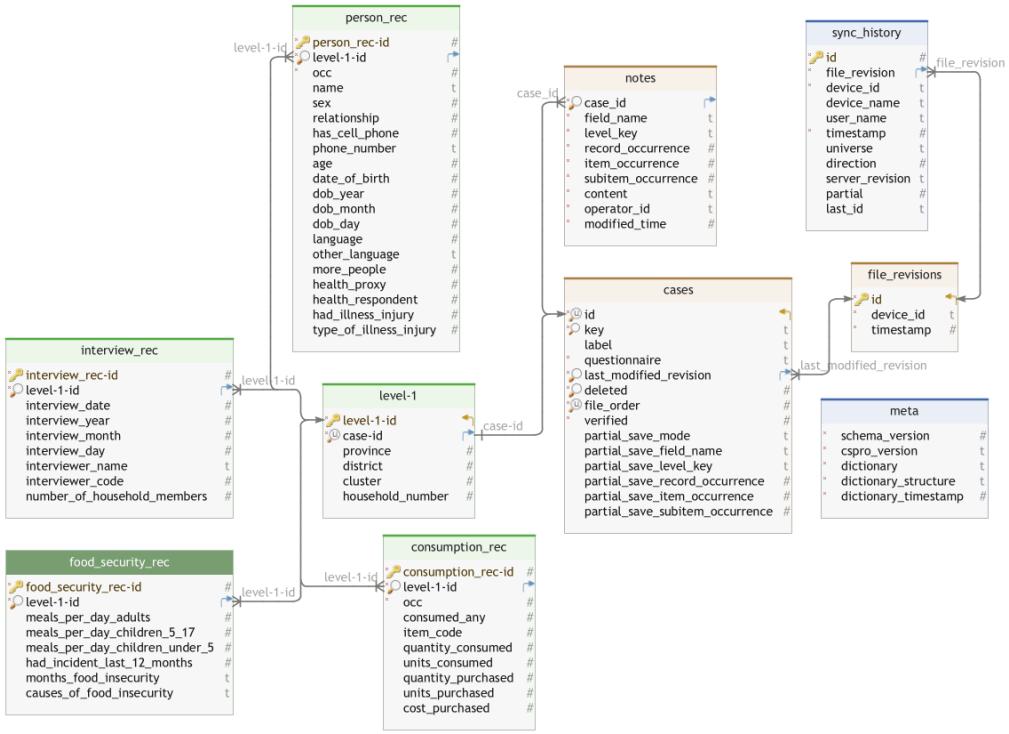
At the end of this session participants will be able to:

- Understand the tables and relationships in a CSPro SQL database
- Use SQL queries from CSPro logic
- Use a "working storage" dictionary to store the results of a query
- Write SQL queries that join tables from multiple data files
- Using HTML and SQL in CAPI question text
- HTML Reports in CSPro using SQL queries

## Structure of a csdb file

Behind the scenes, a csdb file is actually a SQL database. It uses the open-source SQLite database system. Each record in the CSPro dictionary has a corresponding table in the SQL database, and each variable in the dictionary has a corresponding column in the table. SQLite is a file-based database, so unlike SQL Server, MySQL, or Postgres SQL, it does not require a server to run.

There are many free tools you can download to work with SQLite files, including SQLite Studio (https://sqlitestudio.pl ) and DB Browser for SQLite (https://sqlitebrowser.org ). You can open csdb files in both products to see the table structure.



Above are the tables for the household data file. In addition to the tables corresponding to the dictionary records, there are the following tables in each csdb file that contain auxiliary information:

- Level-1: Contains the id-items for the dictionary. If the dictionary has multiple levels, there will also be a table called level-2 that contains the second-level id-items.
- Notes: Interviewer notes associated with the survey questions.
- Cases: Information related to each case in the data file, such as the unique id, case label, and partial save status.
- Sync_history: Each time the file is synchronized, a new row is inserted into this table with information about the synchronization, such as the server and timestamp.
- File_revisions: Each time the file is modified in CSPro (including synchronization), a new row is inserted into this table with the timestamp and unique id of the device the application is running on.
- Meta: General information about the file and the data dictionary used to create it.

## Querying Data Files with SQL

We can use standard SQL to query any of the tables. For example, we can list the name, age and sex of all individuals with the following query:

```
SELECT name, age, sex FROM person_rec
```

We can limit the list to heads of household using a *where* clause:

```
SELECT name, age, sex FROM person_rec WHERE relationship = 1
```

We can use *count* and *group by* to get the total number of males and females:

```
SELECT sex, count(*) FROM person_rec GROUP BY sex
```

What if we want to get the number of children under 5 in each *district*? To do that we need to group by the province and district but those columns are not in the person_rec table. We can join person_rec with level-1 to pull in the province and district:

```
SELECT province, district, count(*) as children_under_5
FROM person_rec p JOIN "level-1" l1 on p."level-1-id" = l1."level-1-id" WHERE age < 5
GROUP BY province, district;
```

Note that because the table name “level-1” contains a dash, it needs to be surrounded by quotes or backticks to distinguish it from subtraction.

Joining with the level-1 table is a very common pattern when working with CSPro data files in order to access the id-items. Using the level-1-id to join different dictionary records is also very common. For example, to find the number of female headed households that experienced food

insecurity in the last twelve months we can join the person_rec with the food_security_rec using the level-1-id:

```
SELECT count(*)
FROM person_rec p
JOIN food_security_rec f on p."level-1-id" = f."level-1-id"
WHERE p.relationship = 1 and p.sex = 2 and f.had_incident_last_12_months = 1
```

**Group Exercise**

Use a SQL query to determine to answer the following questions:

1. Number of individuals over aged 65 who have a cell phone
2. Number of children under 5 living in households with more than 8 members

## SQL Queries from CSPro Logic

In the menu program in the question text for the interviewer, let's show the number of households assigned to the interviewer. We can get this information from the household assignments file using SQL. For example, to find the number of households assigned to the interviewer with code 2:

```
SELECT count(*) FROM hh_assign_rec
WHERE hh_assign_interviewer_code = 2;
```

To show the value in the question text, we need to make this query from CSPro logic. We can use the function `sqlquery()`. It takes the name of the dictionary and the query as arguments and returns the query result.

```
numeric numAssigned = sqlquery(
    HH_ASSIGN_DICT,
    "SELECT count(*) FROM hh_assign_rec WHERE hh_assign_interviewer_code = 2");
```

Let's wrap this in a function so we can use it in the question text and also replace the hardcoded 2 with the user code of the interviewer using the menu application.

```
// Find the number of households assigned to the
// interviewer from the household assignments file
function getNumberOfAssignedHouseholds()
    string query = maketext("SELECT count(*) FROM hh_assign_rec "
                        "WHERE hh_assign_interviewer_code = %d", USER_CODE);
    getNumberOfAssignedHouseholds =
        sqlquery(HH_ASSIGN_DICT, query);
end;
```

Now we can use this function as a fill in the question text for the interviewer menu:

Interviewer: ~~USER_CODE~~

Assigned households: ~~getNumberOfAssignedHouseholds()~~

Note that we don't really need SQL in this case. We could have used the CSPro function `countcases` with a where clause:

```
// Find the number of households assigned to the
// interviewer from the household assignments file
function getNumberOfAssignedHouseholds()
    getNumberOfAssignedHouseholds =
        countcases(HH_ASSIGN_DICT where HH_ASSIGN_INTERVIEWER_CODE = USER_CODE);
end;
```

## Using Working Storage to Retrieve SQL Query Results

In the supervisor menu, let's show the number of households assigned to each interviewer. For example:

| Interviewer | Assigned Households |
|---|---|
| 002 | 2 |
| 003 | 2 |

We can adapt the previous query to group by the interviewer code:

```
SELECT hh_assign_interviewer_code, count(*)
FROM hh_assign_rec
GROUP BY hh_assign_interviewer_code;
```

Unlike the previous query, this query returns more than a single value for the result. It returns two rows with two columns each. However, the `sqlquery` function can only return a single value. To retrieve all the rows and columns we can pass an optional *result set* argument to `sqlquery`. This result set can be either a list, an array or a dictionary record. The query results are copied into this result set. Using a list only supports queries that return a single column since lists are one dimensional. An array can be two dimensional but, in this case, let's use a dictionary record. Rather than create a new record in the main dictionary, we will use a working storage dictionary which is a special dictionary that is never saved to a data file. It is used for storing temporary data in memory while the program is running. First, we need to add a working storage dictionary to our menu application using *Add Files…* from the File menu. Then we add a new record to the working storage dictionary named WS_ASSIGNMENTS_PER_INTERVIEWER_REC with two items: one for each column in the result set. To avoid name conflicts we can prefix the names with WS_API_ so we will have the items WS_API_INTERVIEWER_CODE and WS_API_ASSIGNED_HOUSEHOLDS. We need to make sure we set the max on this new record to have enough occurrences to hold all rows in the query result. Since there are only three interviewers per supervisor, we can set the max to three.

Now we can pass this record as the result set in our call to `sqlquery`. When using a dictionary record as a result set, the names of the columns in the query result must exactly match the names of the corresponding items in the dictionary record. We use AS in the SQL query to name the columns appropriately.

```
    string query =
        "SELECT hh_assign_nterviewer_code as WS_API_INTERVIEWER_CODE,
                count(*) as WS_API_ASSIGNED_HOUSEHOLDS "
        "FROM hh_assign_rec "
        "GROUP BY hh_assign_interviewer_code";
    sqlquery(HH_ASSIGN_DICT, WS_ASSIGNMENTS_PER_INTERVIEWER_REC, query);
```

This will load the results of the query into the working storage record. We can use a for loop to iterate over the rows and build a string that puts the results into a table:

```
// Generate a table of number of interviewers
// with the number of assigned households for use in
// question text
function string getAssignedHouseholdsTable()
    string query =
        "SELECT hh_assign_interviewer_code as WS_API_INTERVIEWER_CODE,
                count(*) as WS_API_ASSIGNED_HOUSEHOLDS "
        "FROM hh_assign_rec "
        "GROUP BY hh_assign_interviewer_code";
    sqlquery(HH_ASSIGN_DICT, WS_ASSIGNMENTS_PER_INTERVIEWER_REC, query);
    string resultTable;
    for WS_ASSIGNMENTS_PER_INTERVIEWER_REC do
        resultTable = resultTable + maketext("%03d %3d\n",
                                WS_API_INTERVIEWER_CODE,
                                WS_API_ASSIGNED_HOUSEHOLDS);
    endfor;
    getAssignedHouseholdsTable = resultTable;
end;
```

We can use this function as a fill in the question text in the supervisor menu:

```
Supervisor: ~~USER_CODE~~

Assigned Households:

~~getAssignedHouseholdsTable()~~
```

This works, but it would look nicer if we used HTML to format the table like this:

```
// Generate a table of number of interviewers
// with the number of assigned households for use in
// question text
function string getAssignedHouseholdsTable()
    string query =
        "SELECT hh_assign_interviewer_code as WS_API_INTERVIEWER_CODE,
                count(*) as WS_API_ASSIGNED_HOUSEHOLDS "
        "FROM hh_assign_rec "
        "GROUP BY hh_assign_interviewer_code";
    sqlquery(HH_ASSIGN_DICT, WS_ASSIGNMENTS_PER_INTERVIEWER_REC, query);
    string resultTable = "<table><tbody>"
        "<tr>"
          "<td>Interviewer</td>"
          "<td>Assigned Households</td>"
        "</tr>";
    for WS_ASSIGNMENTS_PER_INTERVIEWER_REC do
        string tableRow =
            maketext("<tr>"
                       "<td>%03d</td>"
                       "<td>%d</td>"
                     "</tr>",
                    WS_API_INTERVIEWER_CODE,
                    WS_API_ASSIGNED_HOUSEHOLDS);
        resultTable = resultTable + tableRow;
    endfor;
    resultTable = resultTable + "</tbody></table>";

    getAssignedHouseholdsTable = resultTable;
end;
```

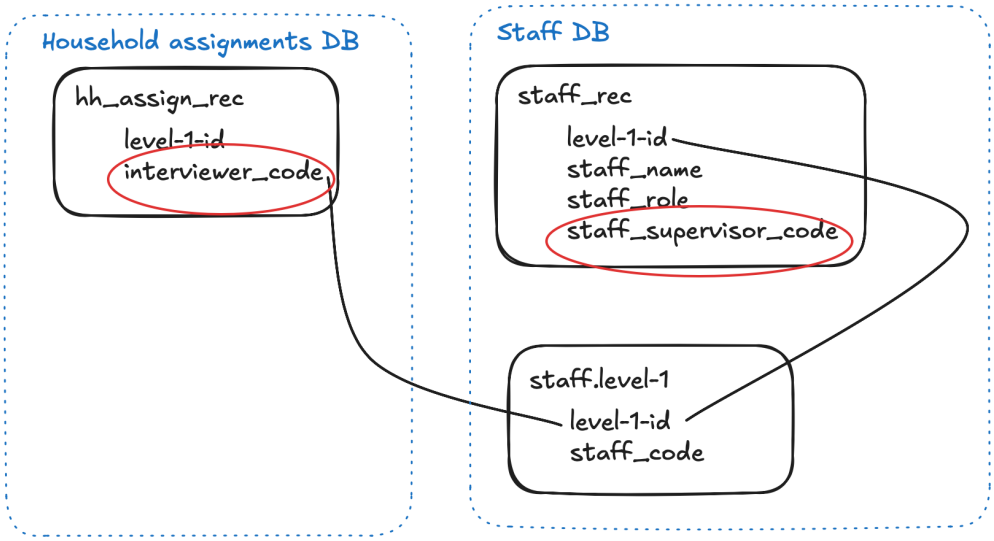
Note that to use an HTML fill in question text, we need to use three tildes instead of two, otherwise the HTML tags will be displayed.

```
Supervisor: ~~USER_CODE~~

~~~getAssignedHouseholdsTable()~~~
```

## Joining Tables in Multiple Data Files

Our query to find the interviewers is not actually correct. We are retrieving the assignments for ALL interviewers, not just the interviewers that report to the current supervisor. To limit the list, we need to filter the interviewers based on the supervisor code in the staff file. We can do this by joining the hh_assign_rec to the staff_rec in the staff data file on the interviewer code. Because the interviewer code is an id-item in the staff file, we need to use the level-1 table in the staff file to link them.




Note that we are using tables from two different CSPro data files. This is supported in SQLiteStudio if you open both databases at the same time. You have to prefix the table with the database name, e.g., staff."level-1". If you don't do this, you will get errors that the names are unknown.

```
SELECT a.interviewer_code, count(*)
FROM hh_assign_rec a
JOIN staff."level-1" sl1 ON sl1.staff_code = a.interviewer_code
JOIN staff.staff_rec s ON sl1."level-1-id" = s."level-1-id"
WHERE s.staff_supervisor_code = 1
GROUP BY a.interviewer_code;
```

When using multiple databases in CSPro, you need to first attach the additional database files by running a query with the ATTACH command. The ATTACH command has the syntax:

```
ATTACH <filepath> AS <database name>
```

For example, to attach the staff data file it would be:

```
ATTACH "../Data/staff.csdb" AS staff
```

In CSPro, we can get the path to the data file using the `filepath` command and then use `maketext` to construct the ATTACH command.

```
    sqlquery(HH_ASSIGN_DICT,
         maketext('ATTACH "%s" AS staff', filename(STAFF_DICT)));
```

Once the query is complete, we need to detach the database, otherwise we will get errors if we attach the same database again.

```
    sqlquery(HH_ASSIGN_DICT, 'DETACH staff');
```

Putting it all together, we end up with the following CSPro function to create the table:

```
function string getAssignedHouseholdsTable()
    sqlquery(HH_ASSIGN_DICT,
         maketext('ATTACH "%s" AS staff', filename(STAFF_DICT)));

    string query =
    "SELECT a.interviewer_code as WS_API_INTERVIEWER_CODE, "
           "count(*) as WS_API_ASSIGNED_HOUSEHOLDS "
    "FROM hh_assign_rec a "
    "JOIN staff.`level-1` sl1 ON sl1.staff_code = a.interviewer_code "
    "JOIN staff.staff_rec s ON sl1.`level-1-id` = s.`level-1-id` "
    "WHERE s.staff_supervisor_code = 1 "
    "GROUP BY a.interviewer_code";
    sqlquery(HH_ASSIGN_DICT, WS_ASSIGNMENTS_PER_INTERVIEWER_REC, query);
    string resultTable = "<table><tbody>"
        "<tr>"
          "<td>Interviewer</td>"
          "<td>Assigned Households</td>"
        "</tr>";
    for WS_ASSIGNMENTS_PER_INTERVIEWER_REC do
        string tableRow =
            maketext("<tr>"
                       "<td>%03d</td>"
                       "<td>%d</td>"
                     "</tr>",
                     WS_API_INTERVIEWER_CODE,
                     WS_API_ASSIGNED_HOUSEHOLDS);
        resultTable = resultTable + tableRow;
    endfor;
    resultTable = resultTable + "</tbody></table>";
    sqlquery(HH_ASSIGN_DICT, 'DETACH staff');

    getAssignedHouseholdsTable = resultTable;
end;
```

**Group Exercise**

In addition to showing the interviewer code, we should also show the name of the interviewer. Modify the table in the supervisor menu to add a column for the interviewer's name:

| Interviewer | Code | Assigned Households |
|---|---|---|
| Anura Wijesinghe | 002 | 2 |
| Anushka Peris | 003 | 2 |

To do this, in addition to updating the SQL query for the interviewer's name, you will also need to add a new item to the record in the working storage dictionary to capture the name and use it in the logic that generates the table.

## Reports

Let's implement the household status report. It lists each household in the cluster that has been interviewed along with the number of household members broken down by sex.

**Household Status Report**

Province: 1
District: 1
Cluster: 1

| Household Number | Total Members | Male | Female |
|---|---|---|---|
| 1 | 7 | 5 | 2 |
| 2 | 8 | 3 | 5 |
| 3 | 6 | 5 | 1 |

To add a new report to the menu application, go to "Add files"... and choose "Report". Click the "..." button to choose the folder in which to place the new report. We will create a new folder "Reports" to store all of our reports. Name the file "Household Status Report". CSPro will ask if you want a "Basic report" or a "Report with sections and a header/footer". Choose "Basic report". You will now see a new section in the form tree on the left called "Reports" and your new report will be in that section. Go to the logic view and click on the new report. This displays the HTML code for the report. CSPro generates a simple HTML page that displays the current date and time. We will customize the HTML to display the list of households.

To recreate the report with fixed data we could write HTML like this:

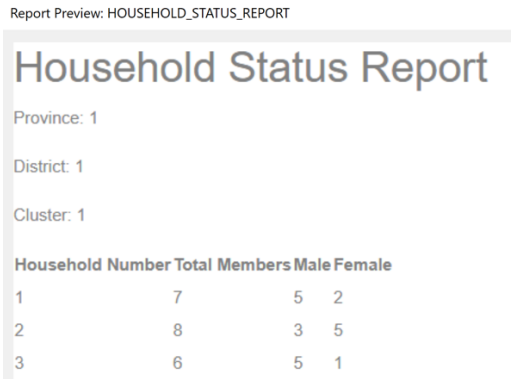
```
<h1>Household Status Report</h1>

<p>Province: 1</p>
<p>District: 1</p>
<p>Cluster: 1</p>

<table>
  <tr>
    <th>Household Number</th>
    <th>Total Members</th>
    <th>Male</th>
    <th>Female</th>
  </tr>
  <tr>
    <td>1</td>
    <td>7</td>
    <td>5</td>
    <td>2</td>
  </tr>
```

```
  <tr>
    <td>2</td>
    <td>8</td>
    <td>3</td>
    <td>5</td>
  </tr>
  <tr>
    <td>3</td>
    <td>6</td>
    <td>5</td>
    <td>1</td>
  </tr>
</table>
```

Pasting this into the report and previewing it by hitting ctrl+F5 we get:




## Using logic in a report template

Instead of hardcoding the data in the table, we want to compute it when we run the report. TO do this we need to use CSPro logic to loop through the household data file. We can add a loop to our report by embedding the CSPro logic in the HTML. The CSPro logic needs to be wrapped in <? ?> similar to embedding PHP in a webpage. We can add a `forcase` to loop through the households with:

```
<? forcase HOUSEHOLD_DICT do ?>

<? endfor; ?>
```

Inside the loop we can write HTML that will be output in the report each time the loop is executed. We can use ~~ ~~ as in CAPI text to write the values of dictionary variables. using this, the code for the table output would be:

```
<h1>Household Status Report</h1>

<p>Province: 1</p>
<p>District: 1</p>
<p>Cluster: 1</p>

<table>
  <tr>
    <th>Household Number</th>
    <th>Total Members</th>
    <th>Male</th>
    <th>Female</th>
  </tr>
  <? forcase HOUSEHOLD_DICT do ?>
  <tr>
    <td>~~HOUSEHOLD_NUMBER~~</td>
    <td>~~COUNT(PERSON_REC)~~</td>
    <td>~~COUNT(PERSON_REC where SEX=1)~~</td>
    <td>~~COUNT(PERSON_REC where SEX=2)~~</td>
  </tr>
  <? endfor; ?>
</table>
```

We can view the report using ctrl+F5, however, this does not execute the logic. It is useful to check the table structure, but to properly test the report, we need to launch it from the menu program so that we can view it with data.

## Displaying a CSPro Report

To view a CSPro report, use the report name followed by ".view()".

```
HOUSEHOLD_STATUS_REPORT.view();
```

You can copy the report name by right clicking the report in the form tree and choosing "Copy name".

We can call this in the reports menu to show the report, however, this will not limit the households to just a single cluster. We will need an additional menu to choose the cluster. Add a new dictionary variable HOUSEHOLD_STATUS_REPORT_CHOOSE_CLUSTER and add it to the form. Modify the postproc of REPORTS to skip to this new field.

```
if $ = 1 then
    // Household status
    skip to HOUSEHOLD_STATUS_REPORT_CHOOSE_CLUSTER;
```

In the onfocus of the new field create a dynamic value set from all the clusters assigned to the supervisor.

```
PROC HOUSEHOLD_STATUS_REPORT_CHOOSE_CLUSTER
onfocus
// Create dynamic value set of clusters assigned to supervisor
valueset string clusterVSet;
forcase CLUSTER_ASSIGN_DICT where CLUSTER_ASSIGN_SUPERVISOR_CODE = USER_CODE do
    string label = maketext("%v-%v-%v",
        CLUSTER_ASSIGN_PROVINCE,
        CLUSTER_ASSIGN_DISTRICT,
        CLUSTER_ASSIGN_CLUSTER);
    string code = key(CLUSTER_ASSIGN_DICT);
    clusterVSet.add(label, code);
endfor;

setvalueset($, clusterVSet);
```

Then in the postproc, load the selected case so that we can access the province, district and cluster number variables in the lookup file from inside the report. Finally show the report using the `view()` method.

```
postproc
// Show the report
loadcase(CLUSTER_ASSIGN_DICT, $);
HOUSEHOLD_STATUS_REPORT.view();
```

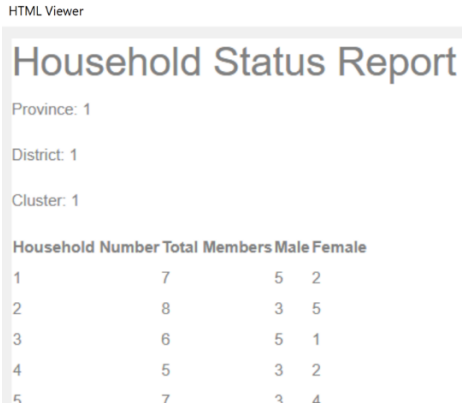
We can then use the variable HOUSEHOLD_STATUS_REPORT_CHOOSE_CLUSTER inside the report logic to limit the households to those in the chosen cluster.

```
  <? forcase HOUSEHOLD_DICT(startswith, HOUSEHOLD_STATUS_REPORT_CHOOSE_CLUSTER) do ?>
```

We can also fill in the values for the province, district and cluster using the variables in the lookup file.

```
<p>Province: ~~CLUSTER_ASSIGN_PROVINCE~~</p>
<p>District: ~~CLUSTER_ASSIGN_DISTRICT~~</p>
<p>Cluster: ~~CLUSTER_ASSIGN_CLUSTER~~</p>
```

Now we see the full report with the data.



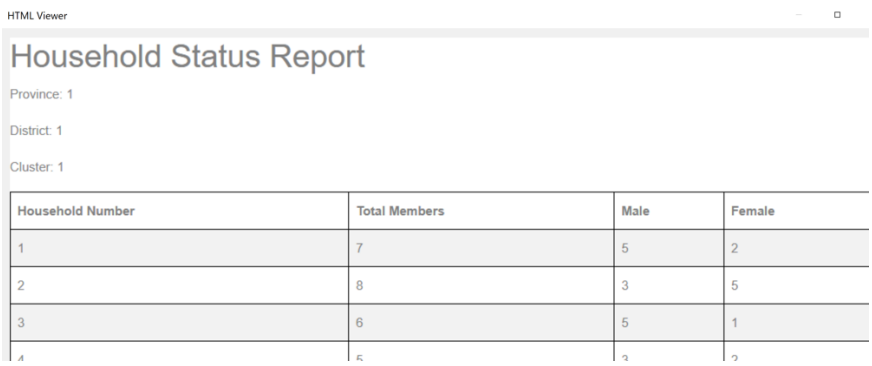
We need to add some styling to make it look nicer. Let's add table borders and highlight every other row. Add the following styles to the <head> section of the report:

```
<style>
table {
  border-collapse: collapse;
  width: 100%;
}

th, td {
  text-align: left;
  padding: 8px;
  border: 1px solid black;
}

tr:nth-child(even) {background-color: #f2f2f2;}
</style>
```

This improves the table output.




## Using SQL in a Report

For more complicated reports we can use `sqlquery` and a working storage record like we did for the assigned households table in the question text of the supervisor menu. Let's use that approach for the second report: cluster progress. The report should show each cluster assigned to the supervisor along with the total households and the number of households completed.

**Cluster Progress Report**

| Province | District | Cluster | Total Households | Completed Households |
|---|---|---|---|---|
| 1 | 1 | 1 | 25 | 17 |
| 1 | 1 | 2 | 30 | 7 |
| 1 | 1 | 3 | 27 | 0 |

For this report we need to join the following:

- Sample file to get the total households in each cluster
- Household file to get the completed households
- Cluster assignments to limit to clusters assigned to the supervisor

To get the completed households we use a left join on the household file and count the rows where there is no match.

```
SELECT s.sample_province,
       s.sample_district,
       s.sample_cluster,
       count( * ) AS total,
       SUM(CASE WHEN hh.province IS NOT NULL THEN 1 ELSE 0 END) AS completed
  FROM Sample.`level-1` s
       LEFT JOIN
       `level-1` hh ON hh.province = s.sample_province AND
                       hh.district = s.sample_district AND
                       hh.cluster = s.sample_district AND
                       hh.household_number = s.sample_household_number
       JOIN
       `cluster assignment`.`level1` cl1 ON
                   cl1.cluster_assign_province = s.sample_province AND
                   cl1.cluster_assign_district = s.sample_district AND
                   cl1.cluster_assign_cluster = s.sample_cluster
       JOIN
       `cluster assignment`.cluster_assign_rec ca ON
                   ca.`level-1-id` = cl1.`level-1-id`
 WHERE ca.cluster_assign_supervisor_code = 1
 GROUP BY s.sample_province,
          s.sample_district,
          s.sample_cluster;
```

Create a new record in the working storage dictionary to store the results of the query. Call it WS_CLUSTER_PROGRESS_REC. Add items for each of the columns in the report.

| | | | | |
|---|---|---|---|---|
| Province | WS_CLUSTER_PROGRESS_PROVINCE | Numeric | 3 | 1 |
| District | WS_CLUSTER_PROGRESS_DISTRICT | Numeric | 4 | 2 |
| Cluster | WS_CLUSTER_PROGRESS_CLUSTER | Numeric | 6 | 3 |
| Total households | WS_CLUSTER_PROGRESS_TOTAL_HOUSEHOI | Numeric | 9 | 3 |
| Completed households | WS_CLUSTER_PROGRESS_COMPLETED_HOU | Numeric | 12 | 3 |

Now we can create the report template. Add a new report named CLUSTER_PROGRESS_REPORT and in the reports menu show the new report when the user picks option 2.

```
elseif $ = 2 then
    // Cluster progress
    CLUSTER_PROGRESS_REPORT.view();
```

In the HTML for the report template, we start with the report title and the table header:

```
<body>
<h1>Cluster Progress Report</h1>
<table>
<tbody>
<tr>
    <th>Province</th>
    <th>District</th>
    <th>Cluster</th>
    <th>Total Households</th>
    <th>Completed Households</th>
</tr>
```

For the table body, we need to first attach the sample and cluster assignments files to the HOUSEHOLD_DICT so we can join tables from all three databases:

```
<?

sqlquery(HOUSEHOLD_DICT, maketext('ATTACH "%s" AS sample;', filename(SAMPLE_DICT)));
sqlquery(HOUSEHOLD_DICT,
    maketext('ATTACH "%s" AS cluster_assign;', filename(CLUSTER_ASSIGN_DICT)));
```

Next, we create the query string and substitute in the supervisor code:

```
string query = maketext(
"SELECT s.sample_province AS WS_CLUSTER_PROGRESS_PROVINCE, "
"       s.sample_district AS WS_CLUSTER_PROGRESS_DISTRICT, "
"       s.sample_cluster AS WS_CLUSTER_PROGRESS_CLUSTER, "
"       count(*) AS WS_CLUSTER_PROGRESS_TOTAL_HOUSEHOLDS, "
"       SUM(CASE WHEN hh.province IS NOT NULL THEN 1 ELSE 0 END) AS "
"                     WS_CLUSTER_PROGRESS_COMPLETED_HOUSEHOLDS "
"  FROM Sample.`level-1` s "
"       LEFT JOIN "
"       `level-1` hh ON hh.province = s.sample_province AND "
"                       hh.district = s.sample_district AND "
"                       hh.cluster = s.sample_district AND "
"                       hh.household_number = s.sample_household_number "
"       JOIN "
"       `cluster_assign`.`level-1` cl1 ON "
"                   cl1.cluster_assign_province = s.sample_province AND "
"                   cl1.cluster_assign_district = s.sample_district AND "
"                   cl1.cluster_assign_cluster = s.sample_cluster "
"       JOIN "
"       `cluster_assign`.cluster_assign_rec ca ON "
"                   ca.`level-1-id` = cl1.`level-1-id` "
" WHERE ca.cluster_assign_supervisor_code = %d "
" GROUP BY s.sample_province, "
"          s.sample_district, "
"          s.sample_cluster;"
, USER_CODE);

sqlquery(HOUSEHOLD_DICT, WS_CLUSTER_PROGRESS_REC, query);
```

Then we iterate through the query results and add a row in the table for each occurrence.

```
for WS_CLUSTER_PROGRESS_REC do ?>
  <tr>
    <td>~~WS_CLUSTER_PROGRESS_PROVINCE~~</td>
    <td>~~WS_CLUSTER_PROGRESS_DISTRICT~~</td>
    <td>~~WS_CLUSTER_PROGRESS_CLUSTER~~</td>
    <td>~~WS_CLUSTER_PROGRESS_TOTAL_HOUSEHOLDS~~</td>
    <td>~~WS_CLUSTER_PROGRESS_COMPLETED_HOUSEHOLDS~~</td>
  </tr>
<? endfor;
```

Finally, detach the dictionaries and close the table HTML.

```
sqlquery(HOUSEHOLD_DICT, 'DETACH sample;');
sqlquery(HOUSEHOLD_DICT, 'DETACH cluster_assign;');
?>
</tbody>
</table>
```

## Exercises

Implement the Interviewer Progress Report. It should list each of the interviewers assigned to the current supervisor along with the number of households assigned to them and the number of households completed.

**Interviewer Progress Report**

| Interviewer | Households Assigned | Households Completed |
|---|---|---|
| Anura Wijesinghe | | |
| Anushka Peris | | |
| Kanchana Jayawardena | | |

You will need to join multiple data files:

- Staff to get the interviewer's name
- Household assignments to get the number of households assigned
- The household file to get the number of households completed

Don't forget to only show the interviewers that report to the current supervisor.