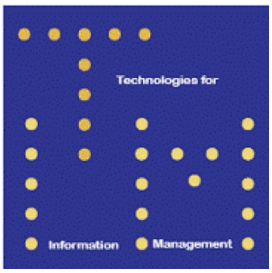

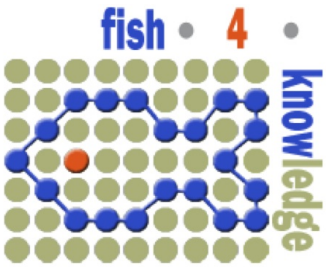

Fish4Knowledge Deliverable D2.5

UI components integrated into end-to-end system

Principal Author:	E. Beauxis-Aussalet, T. Perrucci, L. Hardman
Contributors:	CWI
Dissemination:	PU

Abstract: This document describes the integration of the Fish4Knowledge user interface.

Deliverable due: Month 36

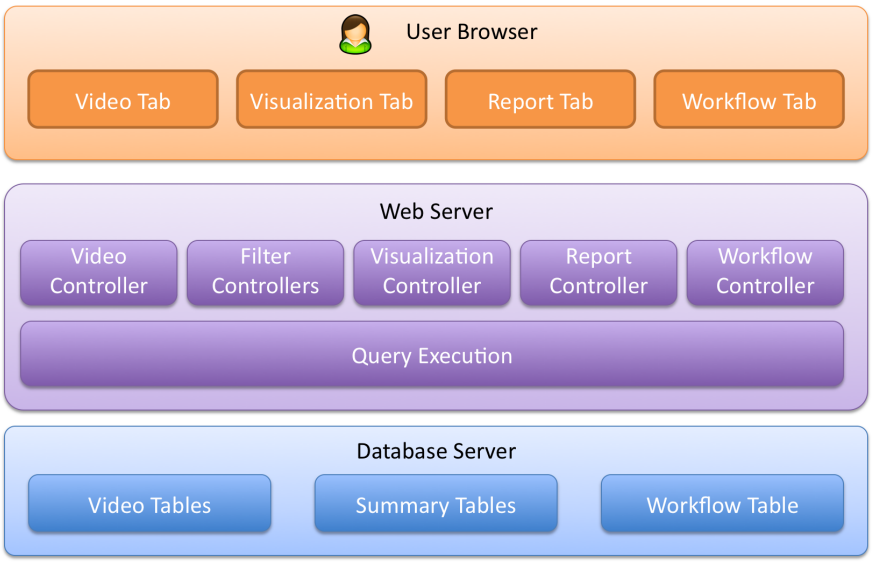

Figure 1: The *Model-View-Controller* architecture of the Public Query Interface.

1 Architecture of the Public Query Interface

The Fish4Knowledge User Interface (UI) deals with 4 main interactive processes: browsing videos (through the *Video* tab, Fig. 2), exploring the video analysis results (through the *Visualization* tab, Fig. 4), grouping visualizations of interest for a particular study (through the *Report* tab, Fig. 6) and requesting high priority video processing (through the *Workflow* tab, Fig. 8). The other functionalities of the UI are implemented in a static manner, e.g., html, css, and images, without interactive access to the F4K data. Their description is not of relevance to the public delivery of the UI, which open source code is available on sourceforge[1].

The architecture chosen for the Fish4Knowledge project uses the *Model-View-Controller* paradigm[2] (Fig. 1). Our architecture includes state-of-the art web-based visualization libraries (*D3*, d3js.org). The **User Browser** (e.g., Chrome or Firefox) displays the information to users. The **Web Server** receives user queries and updates the views (i.e., the displayed UI tabs). The web server interacts with the **Database Server** to retrieve the information needed to address user queries.

2 Video Browser

The functionality for browsing videos uses the *Video Controller*, the *Filter Controllers*, the *Summary Tables*, and the *Video Tables*. The **Filter Controllers** are in charge of storing and

[1] http://sourceforge.net/projects/fish4knowledgesourcecode/
[2] en.wikipedia.org/wiki/Model-view-controller

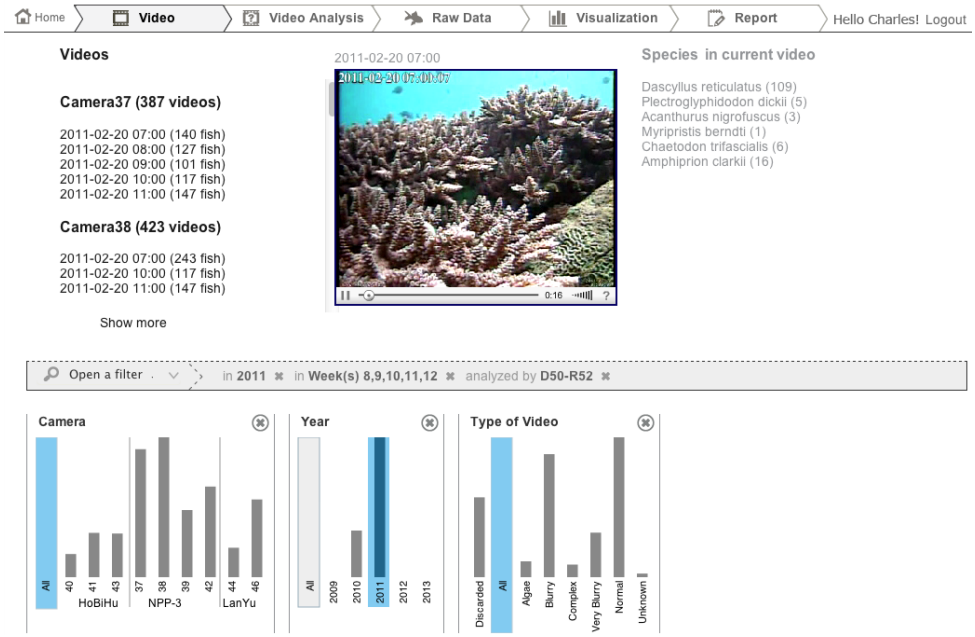

Figure 2: The Video tab.

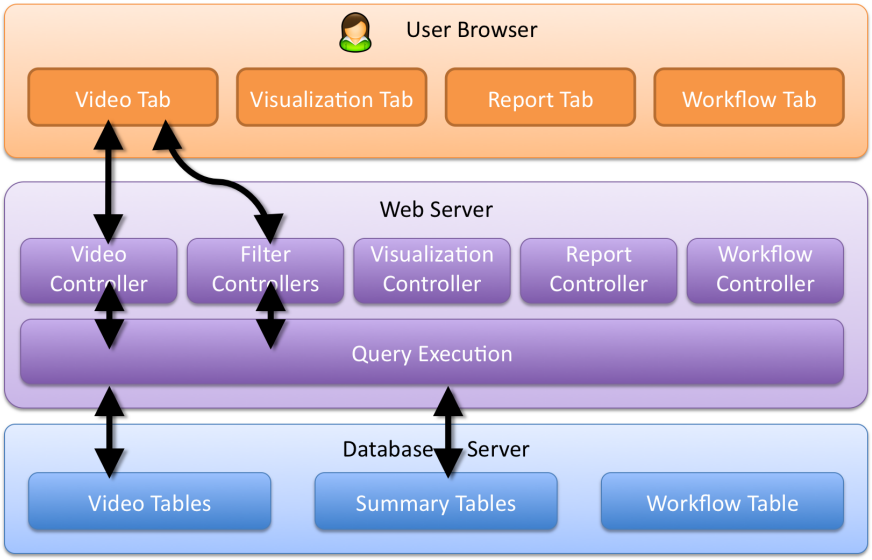

Figure 3: The components involved for browsing videos.

updating the parameters specifying the dataset of interest for the user, and the filter widgets[3] that must be displayed. The **Video Controller** retrieves the urls of the videos to display, depending on the filters set by users (e.g., videos from specific cameras and periods). These processes, illustrated by Fig. 3, are handled by the following pieces of software:

- The file */f4k_ui/service/video.py* contains code to queries to the database. It retrieves the urls of videos to display. It queries the database table named *video*.
- The file */f4k_ui/service/filter.py* contains code to query the database. It retrieves data for updating the filter widgets (e.g., the histograms of the widgets Camera, Year and Type of Video on the bottom part of Fig. 2). It queries the *Summary Tables*. There is one of them for each camera, and their name starts with *summary_table_[...]*. In some cases, the number of processed videos is queried from a *Video Table* named *processed_video*.
- In the repository *static/js/controller/*, the files *filter.js*, *dimension.js*, and *video.js* are executed on the web browser. They serve to send user requests to the web server, and to receive the response from the web server. The file *video.js* handles user inputs related to the specification of the videos to display, and *filter.js* handles user inputs related to the specification of filters. The file *dimension.js* handles user inputs specifying what is represented by the histograms of the filters (e.g., fish counts, species counts, video counts, as defined for the current visualization displayed in the *Visualization* tab).
- The functions *filter_data()* and *video_data()*, in the file *f4k_ui/views.py*, receive user requests and execute the corresponding mechanisms on the web server (i.e., the functions contained in *filter.py* and *video.py*.

3 Data Visualization

The functionality for visualizing the video analysis results uses the *Filter Controllers*, the *Visualization Controller*, the *Summary Tables*, and the *Video Tables*. The **Filter Controllers** is in charge of storing and updating the parameters specifying the dataset of interest for the user. The **Visualization Controller** is in charge of storing and updating the parameters specifying the main graph (Zone A in Fig. 4). These processes, illustrated by Fig. 5, are handled by the following pieces of software:

- The files */f4k_ui/service/viz.py* and */f4k_ui/service/filter.py* contain code to query the database. The file *viz.py* retrieves data for updating the main graph (Zone A of Fig. 5), and *filter.py* retrieves data for updating the filter widgets (Zone C of Fig. 5). Both query the *Summary Tables*. There is one of them for each camera, and their name starts with *summary_table_[...]*. In some cases, they query a *Video Table* named *processed_video*.
- In the repository *static/js/controller/*, the files *filter.js*, *dimension.js*, and *viz.js* are executed on the web browser. They serve to send user requests to the web server, and to receive the response from the web server. The file *filter.js* handles user inputs related to the specification of filters. The file *dimension.js* handles user inputs specifying what is

[3]The filter widgets are boxes displayed in the UI on user demand, and containing functionalities for filtering data. Fig. 2-4 show 3 filter widgets displayed in the bottom part of the UI part.

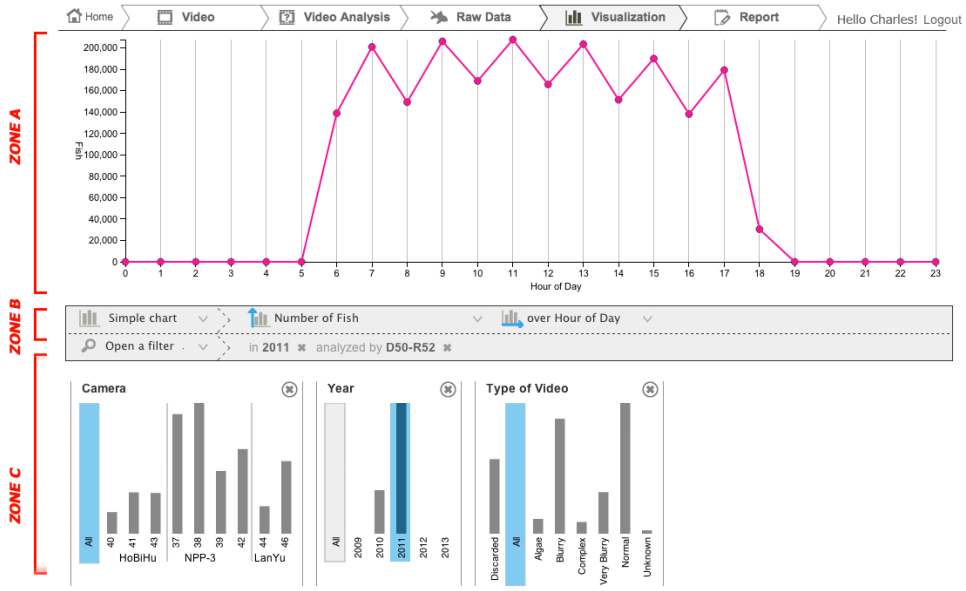

Figure 4: The Visualization tab.

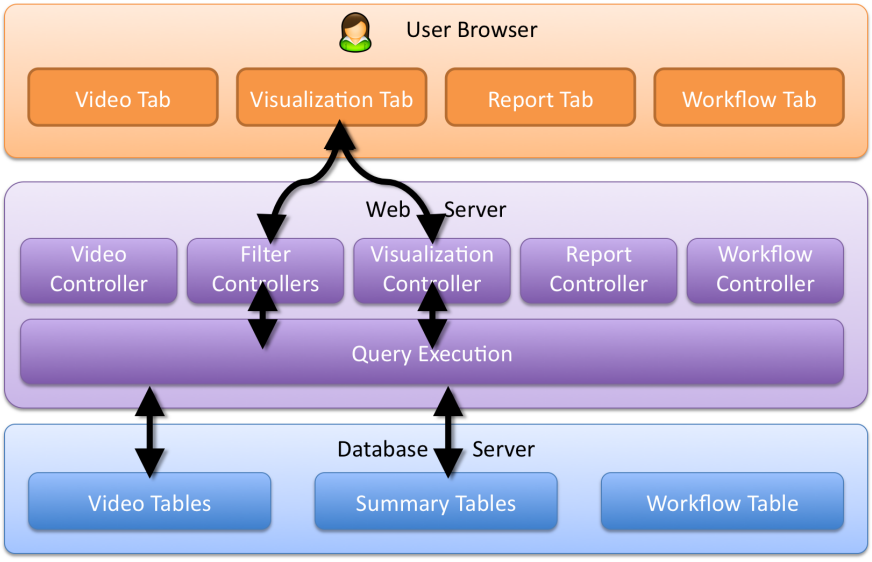

Figure 5: The components involved for visualizing the video analysis results.

represented by the histograms of the filters (e.g., fish counts, species counts, video counts, as defined for the current visualization displayed in the *Visualization* tab). It also handles the specification of what is represented by the Y axis of the main graph. The file *viz.js* handles user inputs related to the specification of the main graph.

- The functions *filter_data()* and *viz_data()*, in the file *f4k_ui/views.py*, receive user requests and execute the corresponding mechanisms on the web server (i.e., the functions contained in *filter.py* and *viz.py*.

4 Grouping visualizations in reports

The functionality for grouping visualizations of interest uses the *Report Controller*, the *Summary Tables*, and the *Video Tables*. The **Report Controller** is in charge of collecting and updating the parameters specifying the group of visualizations of interest for the user. The display of visualization thumbnails use the same mechanisms as for the *Visualization* tab. These processes, illustrated by Fig. 7, are handled by the following pieces of software:

- The file */f4k_ui/service/report.py* contains code to retrieve and store data in the user session, a memory space managed by the web server. It handles the data specifying the visualizations grouped in the *Report* tab. The web server memorizes one configuration of the *Report* tab for each user.

- The file */f4k_ui/service/viz.py* contains code to query the database. It retrieves data for displaying the grouped visualizations. It queries the *Summary Tables*. There is one of them for each camera, and their name starts with *summary_table_[...]*. In some cases, it queries a *Video Table* named *processed_video*.

- The file *static/js/controller/report.js*, executed on the web browser, serves to send user requests to the web server, and to receive the response from the web server. It handles user inputs related to the specification of the visualizations to insert or remove from the *Report* tab.

- The function *user_report()*, in the file *f4k_ui/views.py*, receive user requests and execute the corresponding mechanisms on the web server (i.e., the functions contained in *report.py* and *viz.py*).

5 Requests to the Workflow

Users can request the workflow to schedule high-priority video processing. This functionality uses the *Workflow Controller* and the *Workflow Table*. The **Workflow Controller** is in charge of storing and updating the parameters specifying the high priority video processing to perform, and to list the high-priority video processing that were previously requested. These processes, illustrated by Fig. 9, are handled by the following pieces of software:

- The file */f4k_ui/service/workflow.py* contain code to query the database. It retrieves the list of previous request to the workflow, and adds new requests. It reads and writes the *Workflow Table* named *query_management* in the database.

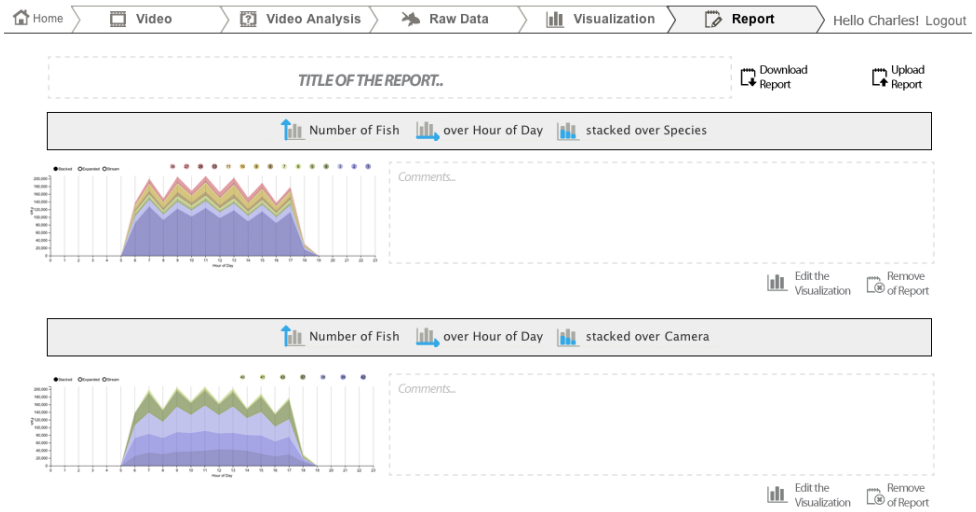

Figure 6: The Report tab.

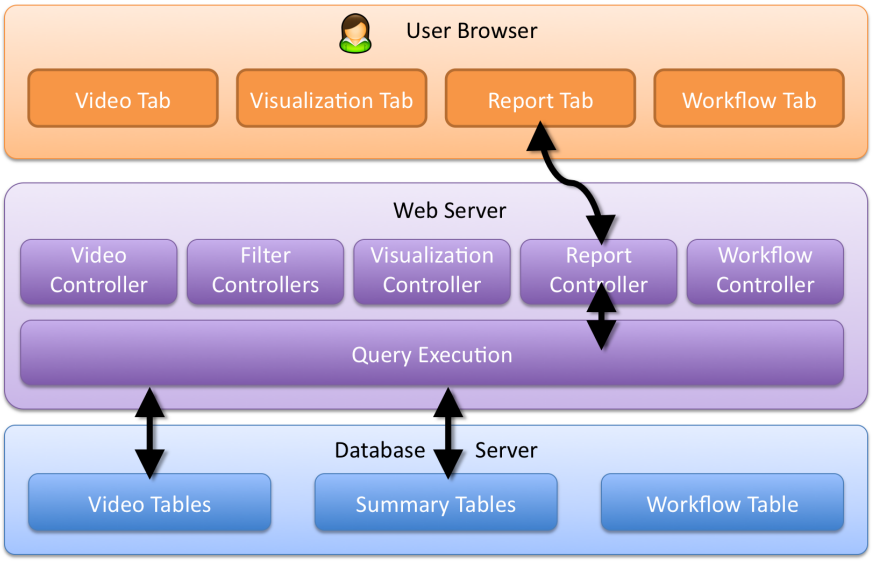

Figure 7: The components involved for grouping visualizations in the Report tab.

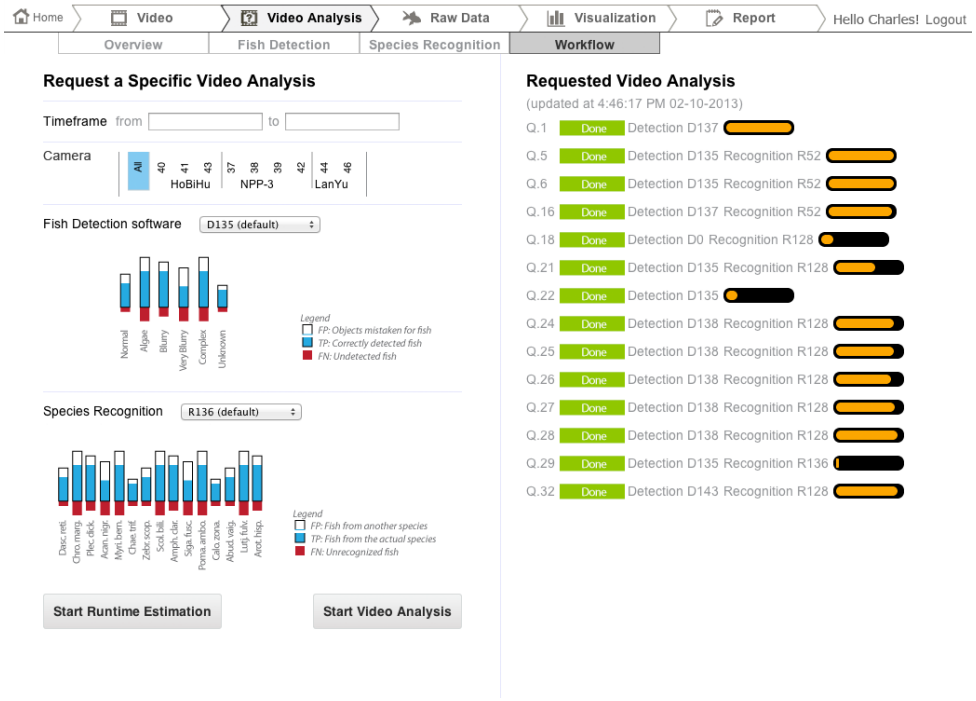

Figure 8: The Workflow sub-tab.

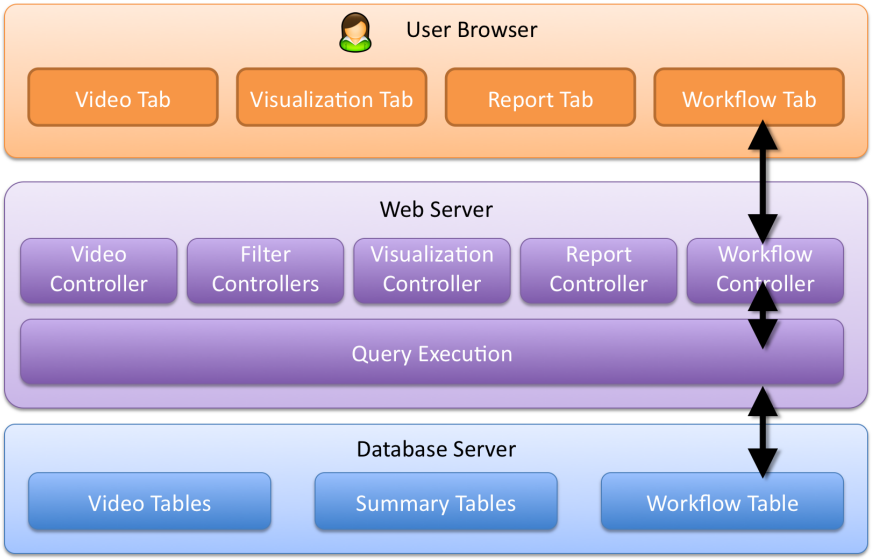

Figure 9: The components involved for requesting the workflow to execute high-priority video processing.

- The file *static/js/controller/workflow.js*, executed on the web browser, serves to send user requests to the web server, and to receive the response from the web server (e.g., the updated list of requests to the workflow).
- The function *user_query_data()*, in the file *f4k_ui/views.py*, receives user requests and execute the corresponding mechanisms on the web server (i.e., the functions contained in *workflow.py*.

6 The Summary Tables

The F4K database contains an extensive amount of data, spread over several tables. To address user information needs, we need data from several of these large tables. This triggers issues regarding the latency of database queries. We implemented views that gather information into the same table, thus suppressing relations between tables, and time-consuming joints in SQL queries. To reduce further the latency, and we reduced the size of the tables that are queried, and implemented one view per camera. These views are the *Summary Tables*, which names start with *summary_table_[...]* followed by the camera id.

Each row of the *Summary Tables* represents a single fish occurrence and its characteristics, as described by both Fish Detection and Species Recognition components. Fig. 10 gives examples of fish description in the *Summary Tables*. The fields provides the following information:

- *fish_id*: unique identifier of the fish occurrence (i.e., after tracking unique fish), originating from the table named *fish*.
- *species_id*: identifier of the fish species, originating from the table named *fish_species_cert*.
- *video_class*: identifier of the type of video quality (e.g., Normal, Algae, Blurry, ...), originating from the table named *video_class*.
- *best_video_id*: identifier of the video in which the fish occurs, originating from the table named *fish*.
- *det_certainty*: average certainty score of all fish images along the fish trajectory, originating from the table named *fish*.
- *tracking_certainty*: certainty score of the fish trajectory, originating from the table named *fish*.
- *rec_certainty*: certainty score of the species recognition, originating from the table named *fish_species_cert*.
- *det_component_id*: identifier of the version of the Fish Detection component that detected the fish, originating from the table named *fish*.
- *rec_component_id*: identifier of the version of the Species Recognition component that recognized the fish species, originating from the table named *fish_species_cert*.
- *date*: identifier of the 10-minute time period when the fish occurred, originating from the table named *video*.

fish_id	species_id	video_class	best_video_id	det_certainty	tracking_certainty	rec_certainty	det_component_id	rec_component_id	date
28	1	6	962521d9336ea03ed1666807155c...	0.031107	0.999998	0.91667	19	52	2012-02-05 06:10:00
29	1	6	962521d9336ea03ed1666807155c...	0.155214	0.997911	0.958183	19	17	2012-02-05 06:10:00
30	1	6	962521d9336ea03ed1666807155c...	0.932351	0.972163	0.96645	19	52	2012-02-05 06:10:00
31	1	6	962521d9336ea03ed1666807155c...	0.70774	0.888746	0.945946	19	17	2012-02-05 06:10:00

Figure 10: Examples of the information stored in the *Summary Tables*. Each row represents a single fish occurrence.

The *Summary Tables* need to be updated as long as the videos are being processed by the workflow (both for automatic processing or on-demand user requests). The file */database/summary_camera_proc.sql* is a script able to update the *Summary Tables*. It can be executed automatically at regular interval, or manually, if needed. The update of the *Summary Tables* is important for the UI to display accurate data. In particular, the visualization of average fish counts per video is biases if the tables are not up-to-date. The numbers of videos are extracted from the table *processed_videos*, which is constantly updated after each video is processed. But the numbers of fish are extracted from the *Summary Tables*, which are not guaranteed to be up-to-date anytime. Thus the average number of fish per video can be artificially under-estimated.

The first users of our system must be aware of this inconvenience. They have to make sure the *Summary Tables* are up-to-date before performing data analyses. They must be able to trigger the update of the tables if needed (e.g., launch the .sql script). For a more intensive use of the system, it is recommended to implement a consistent mechanism for updating the *Summary Tables*. Updates would take place either after automatic batch video processing, or after user requests to the workflow are completed. The *processed_videos* and the *Summary Tables* should be updated synchronously.

