Site Management System

This is a system designed to manage items and associated images for multiple websites. Each item can have various types of images: static images, preview images, and production images. The system provides functionality to delete multiple items and their associated images while considering certain constraints and operational considerations.

Next, find a brief description of some system design decisions. This does not pretend to be an exhaustive nor a complete document – these are notes of the high level design.

The data model

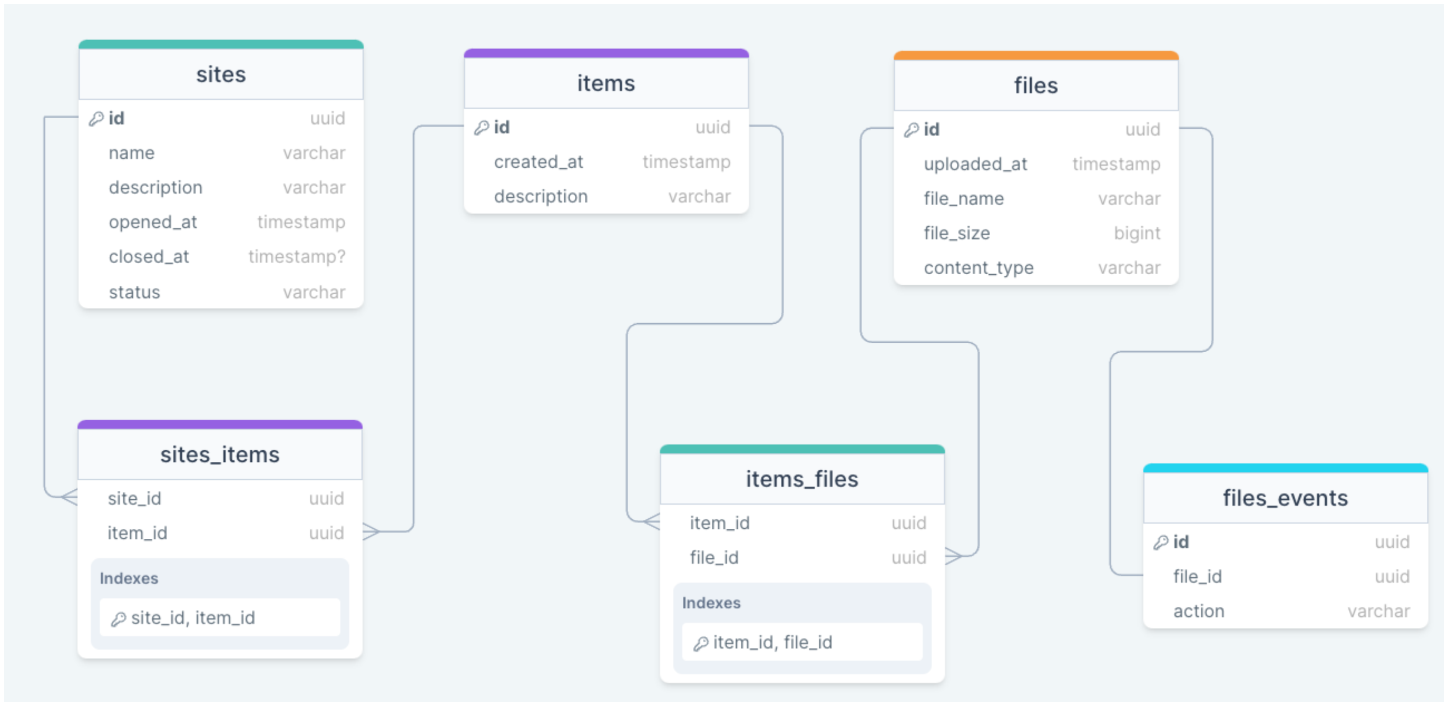

Figure 1 - ER Diagram

```
CREATE TABLE "sites"(
    "id" UUID PRIMARY KEY NOT NULL,
    "name" VARCHAR(255) NOT NULL,
    "description" VARCHAR(255) NOT NULL,
    "opened_at" TIMESTAMP NOT NULL,
    "closed_at" TIMESTAMP,
    "status" VARCHAR(255) NOT NULL
);

CREATE TABLE "items"(
    "id" UUID PRIMARY KEY NOT NULL,
    "created_at" TIMESTAMP NOT NULL,
    "description" VARCHAR(255) NOT NULL
);

CREATE TABLE "sites_items"(
    "site_id" UUID NOT NULL,
    "item_id" UUID NOT NULL,
    PRIMARY KEY("site_id", "item_id"),
    FOREIGN KEY("site_id") REFERENCES "sites"("id"),
    FOREIGN KEY("item_id") REFERENCES "items"("id")
);
```

```
CREATE TABLE "files"(
    "id" UUID PRIMARY KEY NOT NULL,
    "uploaded_at" TIMESTAMP NOT NULL,
    "file_name" VARCHAR(255) NOT NULL,
    "file_size" BIGINT NOT NULL,
    "content_type" VARCHAR(255) NOT NULL
);

CREATE TABLE "items_files"(
    "item_id" UUID NOT NULL,
    "file_id" UUID NOT NULL,
    PRIMARY KEY("item_id", "file_id"),
    FOREIGN KEY("item_id") REFERENCES "items"("id"),
    FOREIGN KEY("file_id") REFERENCES "files"("id")
);

CREATE TABLE "files_events"(
    "id" UUID PRIMARY KEY NOT NULL,
    "file_id" UUID NOT NULL,
    "action" VARCHAR(255) NOT NULL,
    FOREIGN KEY("file_id") REFERENCES "files"("id")
);
```

System Design Notes

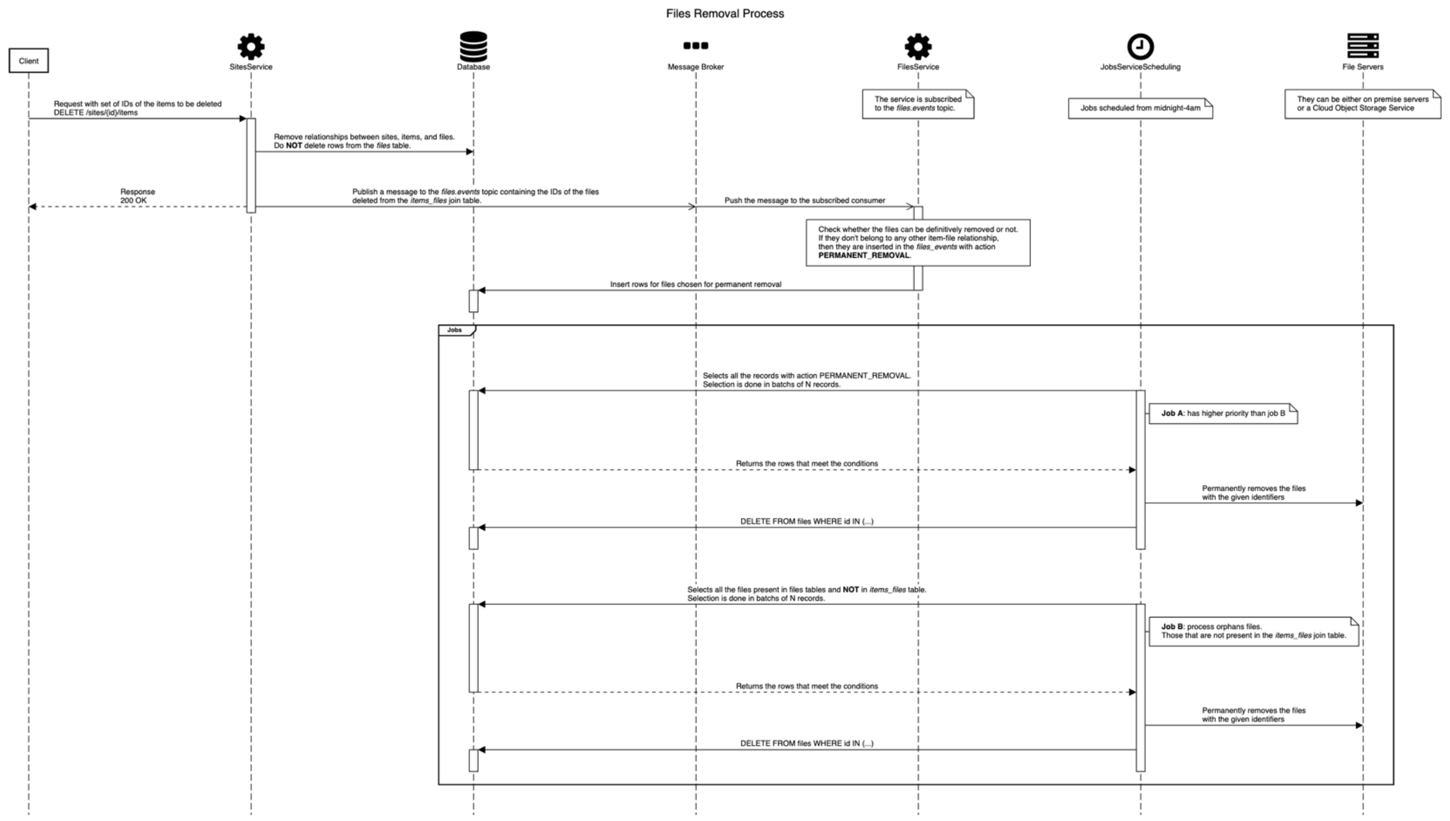
Files Removal Process
Client
SitesService
Database
Message Broker
FilesService
JobsServiceScheduling
File Servers
The service is subscribed to the files.events topic.
Jobs scheduled from midnight-4am
They can be either on premise servers or a Cloud Object Storage Service
Request with set of IDs of the items to be deleted DELETE /sites/{id}/items
Remove relationships between sites, items, and files. Do NOT delete rows from the files table.
Response 200 OK
Publish a message to the files.events topic containing the IDs of the files deleted from the items_files join table.
Push the message to the subscribed consumer
Check whether the files can be definitively removed or not. If they don't belong to any other item-file relationship, then they are inserted in the files_events with action PERMANENT_REMOVAL.
Insert rows for files chosen for permanent removal
Jobs
Selects all the records with action PERMANENT_REMOVAL. Selection is done in batchs of N records.
Job A: has higher priority than job B
Returns the rows that meet the conditions
Permanently removes the files with the given identifiers
DELETE FROM files WHERE id IN (...)
Selects all the files present in files tables and NOT in items_files table. Selection is done in batchs of N records.
Job B: process orphans files. Those that are not present in the items_files join table.
Returns the rows that meet the conditions
Permanently removes the files with the given identifiers
DELETE FROM files WHERE id IN (...)

Figure 2 - Sequence Diagram for the Sites Management System

You can zoom in to better see the diagram, in any case, you can also find it full size here: https://t.ly/site-management-system-sequence-diagram

If after reading the sequence diagram, you still have questions, find some additionals details here:

- The main service serving the requests is a RESTful API over HTTP, let's call it: `SitesService`. This service provides an endpoint (`DELETE /sites/{id}/items`) that receives a set of the IDs of the items to be deleted. The service:
  - Performs a DB deletion of all the items and the associations that they could have in the *items_files* table.
  - After that, it asynchronously publishes an event with a message that contains the IDs of all the deleted files, those that were removed from the join table. The message is published to the *files.events* topic.
  - Then, it returns with a 200 response if everything was OK.
- The `FilesService`, is subscribed to the *files.events* topic and each time a new message arrives, it process it taking into consideration:
  - Static images: as they are not shared, they are inserted in the *files_events* table with action **PERMANENT_REMOVAL**.
  - Preview images: if no occurrences in the *items_files* table, it means no item have them associated, so it's safe to delete them. They are inserted in the *files_events* table with action **PERMANENT_REMOVAL**.
  - Production images: these are skipped and are candidates for the *orphan_images_removal_job* (Job B, in the sequence diagram).