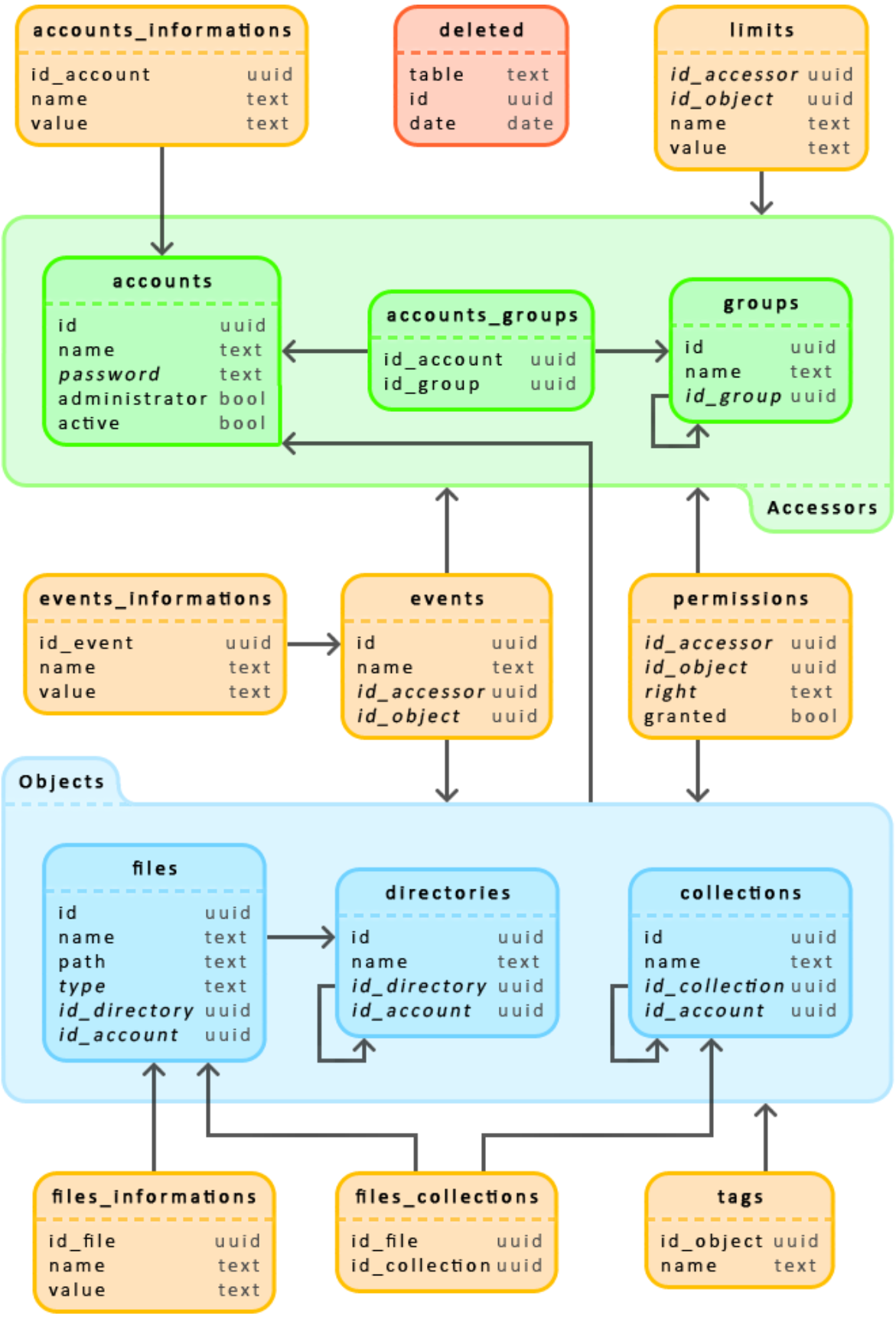
Base de données

Sommaire

I. Structure

Le schéma ci-dessous représente la structure base de données.



Afin de simplifier le schéma, les champs **created** et **modified** dont toutes les tables disposent à l'exception de **deleted**, ne sont pas représentés. De même, toutes les tables sauf deleted ont un champ **id**, mais seuls les id utilisés par d'autres tables, c'est-à-dire qui sont utilisés par des clés externes, sont affichés sur ce schéma.

Les noms des tables et des champs sont en minuscules. La valeur des champs en *italique* peut être vide, mais aucun champ ne peut être NULL.

II. Accessors et Objects

Le terme **« accessor »** regroupe les tables **accounts** et **directories**. De même, le terme **« object »** regroupe les tables **files**, **directories**, et **collections**. L'idée générale de cette représentation est qu'un accessor est une entité qui peut effectuer des opérations sur un object. Ceci est pratique par exemple pour la gestion des droits où un accessor est lié à un object via leurs id. Un compte peut ainsi avoir des droits sur un fichier, et un groupe peut en avoir sur une collection…

Ce système est rendu possible par l'utilisation systématique d'**universal unique identifiers** (ou **uuid**) pour identifier les entrées. Chaque accessor et object a ainsi un identifiant unique dans toute la base de données, et par extension dans le monde entier.

Tous les objects ont un champ **id_account** qui représente leur propriétaire s'il n'est pas vide.

III. Created et modified

Les champs **created** et **modified** présent dans chaque table permettent de savoir quand une entrée a été créée et modifiée. Leur date est au format yyyy-MM-dd hh:mm:ss. Ces champs sont initialisés automatiquement lors de la création d'une entrée, et le champ modified est mis à jour à chaque modification par un trigger.

IV. Unicité

Certaines tables sont soumises à des contraintes d'unicité plus ou moins restrictives, sur un ou plusieurs champs. Par exemple, le champ name de la table accounts est unique, ce qui signifie que tous les utilisateurs ont un nom différent.

Un autre exemple est la table files, dans laquelle deux entrées ne peuvent pas avoir les mêmes name et id_directory en même temps. Cela signifie que deux fichiers ne peuvent pas avoir le même nom s'ils sont dans le même dossier.

V. Abstraction

L'API du serveur propose une abstraction de la base de données, qui permet d'accéder facilement aux champs d'une entrée, et d'y effectuer des opérations plus ou moins complexes.

VI. Triggers

1. Modification

Lorsqu'une entrée est modifiée par la clause UPDATE, son champ **modified** est automatiquement mis à jour à la date courante.

De plus, un trigger interdit la modification de l'id d'une entrée. L'uuid est définitivement défini lors de la création de l'entrée, ce qui évite de se retrouver avec des clés étrangères qui ne pointent sur rien.

Enfin, un trigger vérifie que les champs qui ne sont pas en *italique* sur le schéma ne sont jamais vides.

2. Suppression

Lorsqu'une entrée est supprimée, son id, sa table, et la date de suppression sont sauvegardées dans la table **deleted**.

Ensuite, le trigger lance la suppression en cascade, c'est-à-dire que les entrées qui dépendent de l'entrée supprimée sont supprimées à leur tour. Par exemple, supprimer un fichier supprimera également les entrées files_informations, files_collections, tags, permissions, et events dont l'id_file ou l'id_object pointent sur ce fichier. Ceci permet de ne pas avoir des entrées qui pointent sur un élément qui n'existe plus, et donc de garder la base de données cohérente.

3. Clés étrangères

Les clés étrangères sont simulées à l'aide de triggers qui vérifient qu'elles pointent toujours vers un identifiant valide. Par exemple, lorsqu'une entrée est insérée ou modifiée dans la table tags, un trigger vérifie que le champ id_object pointe bien vers l'id d'un object qui existe, c'est-à-dire l'id d'un fichier, d'un dossier, ou d'une collection. Si ce n'est pas le cas, une erreur est générée, et la requête annulée par roll back.

Attention : Aucune vérification de hiérarchie cyclique sur les dossiers les collections et les groupes n'est effectués par la base de données ou le serveur. C'est aux utilisateurs de s'assurer qu'aucune hiérarchie cyclique n'est introduite. Par exemple si un dossier a comme parent un de ses fils, des boucles infinies risquent de se produire dans le serveur.

VII. Tables

1. Deleted

La table deleted permet de suivre les suppressions effectuées sur la base de données. Lorsqu'une entrée est supprimée, sont identifiant est inséré dans cette table. Cette opération est effectuée à l'aide de triggers spécifiques.

Champs

Table	Contient le nom de la table dont l'entrée qui a été supprimée.
Id	Identifiant de l'entrée supprimée.
Date	Date de suppression de l'entrée.

2. Accounts

Cette table stock les comptes enregistrés sur le serveur. La table **accounts_informations** contient quand à elle les informations d'un compte (adresse, téléphone, langue…).

Champs

Name	Nom du compte.
Password	Le mot de passe est stocké sous la forme d'un **SHA1**. S'il est vide, cela signifie que le compte est public, puisque n'importe qui peut l'utiliser pour se connecter.
Administrator	Indique si le compte est administrateur.
Active	Indique si le compte est activé ou non.

3. Groups

Les groupes permettent de rassembler plusieurs comptes afin de faciliter leur administration. Les comptes sont liés aux groupes grâce à la table accounts_groups. Un compte peut appartenir à plusieurs groupes, voir aucun, et un groupe peut être contenu dans un autre groupe.

Champs

Name	Le nom du groupe.
Id_group	Identifiant du groupe parent. Ceci permet d'imbriquer les groupes. Peut être vide si le groupe est à la racine, c'est-à-dire s'il n'a pas de parent.

4. Limits

Cette table permet de fixer des limites d'utilisation du serveur d'un accessor et/ou d'un object. Par exemple, cela peut être utilisé pour limiter le volume ou le nombre de fichier qu'un accessor peut télécharger en un jour / mois, ou limiter le nombre de fois qu'un fichier peut être lu.

5. Events

Stock les événements qui se produisent sur le serveur. Le champ **created** est utilisé pour connaitre la date à laquelle un événement c'est produit. La table **events_informations** peut contenir des informations supplémentaires sur l'événement.

Champs

Name	Nom de l'événement.
Id_accessor	Identifiant de l'accessor associé à l'événement. Peut être vide si l'événement n'a pas d'accessor.
Id_object	Identifiant de l'object associé à l'événement. Peut être vide si l'événement n'a pas d'object.

6. Permissions

Cette table gère le système de droits du serveur. Le concept général est de lier un accessor à un object, et de leur associer un droit qui peut être accordée ou refusée.

Champs

Id_accessor	Accessor sur lequel porte la permission. Si ce champ n'est pas rempli, la permission désignera tous le monde.
Id_object	Object visé par la permission. Si l'object n'est pas précisé, la permission portera sur la racine du serveur.
Right	Type de droit qui leur est associés. Si le droit est vide, tous les droits seront concernés.
Granted	Indique si la permission est accordée (***true***), ou refusée (***false***).

Le système de permissions est décrit plus précisément dans la documentation du serveur.

7. Files

Représente un fichier présent sur le serveur. La table **files_informations** contient des informations sur un fichier.

Champs

Name	Nom du fichier tel qu'il est affiché à l'utilisateur. Ce nom peut être différent de son nom réel.
Path	Chemin vers l'emplacement du fichier sur le disque dur. Peut être relatif ou absolu. Si le chemin est relatif, son point de départ est le dossier « **filesPath** » du serveur.
Type	Le type du fichier. Les valeurs possibles sont **image**, **video**, **audio**, **document**, ou **other**. Si ce champ vaut une autre valeur ou qu'il est vide, cela correspond à **other**.
Id_directory	Dossier dans lequel se trouve le fichier. S'il est vide, le fichier se trouve à la racine du serveur.
Id_account	Identifiant du propriétaire du fichier. Le propriétaire est le compte qui a ajouté le fichier.

Voici quelques exemples de données qui peuvent être contenues dans ta table **files_informations**. Leur présence et leur nombre dépend du type de fichier, et des plugins installés sur le serveur.

Size	La taille du fichier en octet.
Mime	Le type MIME du fichier.
Extension	Son extension.
Width	La largeur de l'image ou de la vidéo.
Height	La hauteur de l'image ou de la vidéo.
Format	Le format du fichier.
Duration	La durée de la vidéo ou de l'audio en seconde.
Title	Le titre de la musique ou du film.

8. Directories

Représente un dossier. Les dossiers ne correspondent pas à des dossiers réellement présents sur le disque dur du serveur. Ils peuvent stocker des fichiers, ou d'autres dossiers.

Champs

Name	Le nom du dossier.
Id_directory	Identifiant du dossier parent. Permet d'imbriquer les dossiers. Peut être vide s'il n'a pas de parent. Dans ce cas, le dossier est à la racine de l'arborescence du serveur.
Id_ account	Identifiant du propriétaire du dossier. Le propriétaire est le compte qui a créé le dossier.

9. Collections

Les collections sont des listes de fichiers. Elles peuvent être comparées à des listes de lecture. La table files_collections lie les fichiers aux collections. Un fichier peut être associé à plusieurs collections.

Champs

Name	Nom de la collection. Doit être unique dans les fils de son parent.
Id_collection	Identifiant de la collection parente. Permet d'imbriquer les collections. Peut être vide si elle n'a pas de parent.
Id_ account	Identifiant du propriétaire de la collection. Le propriétaire est le compte qui a créé la collection.

10. Tags

Un tag est une information qui peut être associée à un object, et qui permet de les retrouver facilement. Par exemple une photo de forêt pourra recevoir les tags "forêt", "nature", "bois", …