Database reverse engineering and assessment for Android applications

INFOM218 - Année académique 2024-2025

Groupe 1

Projet analysé : Vespucci

François Bechet - Thibaut Berg

Anaé De Baets - Carine Pochet

Professeur

Prof. Anthony Cleve

UNIVERSITÉ DE NAMUR

Université de Namur

8 janvier 2025

Table des matières

1 Introduction

Dans le cadre du cours INFOM218 à l'Université de Namur, ce rapport se concentre sur l'analyse et l'évolution des bases de données dans les applications Android, avec une étude de cas sur Vespucci, un éditeur OpenStreetMap open-source. Ce projet vise à effectuer une ingénierie inverse de la base de données, à élaborer des scénarios d'évolution, et à proposer des modifications pour améliorer la structure et la performance de l'application. En exploitant des outils tels que SQLInspect et des scripts personnalisés, nous avons étudié le schéma physique, logique et conceptuel de la base de données, tout en identifiant des opportunités d'optimisation. Ce rapport détaille nos démarches méthodologiques, nos découvertes et les solutions proposées pour adapter la base de données aux besoins évolutifs de l'application. Afin de garantir la transparence sur la répartition des tâches au sein du groupe et de permettre une consultation des travaux réalisés, le code produit durant ce projet a été publié sur un dépôt GitHub accessible via le lien suivant : INFOM128-vespucci.

2 Partie 1

2.1 Schéma physique

Pour extraire les données du schéma physique, nous avons utilisé le plugin SQLInspect sur Eclipse afin de récupérer toutes les requêtes SQL du programme. Un script python a ensuite été créé afin de pouvoir trier les différentes requêtes en fonction de mots-clefs. La combinaison des différentes requêtes de création des tables ainsi que des contraintes a mené à ce résultat. On peut y retrouver 6 clefs étrangères qui ne référencent pas toujours des champs uniques. La seule clef primaire composée est située sur la table `tiles`. On retrouve une multitude de champs qui sont nullable. Ces derniers peuvent être utilisés comme référence dans une clef étrangère. Toutes les tables présentent des noms qui sont plutôt explicites. On comprend son utilité et les données qui y sont stockées. Il en va de même pour la majorité des noms d'attributs.

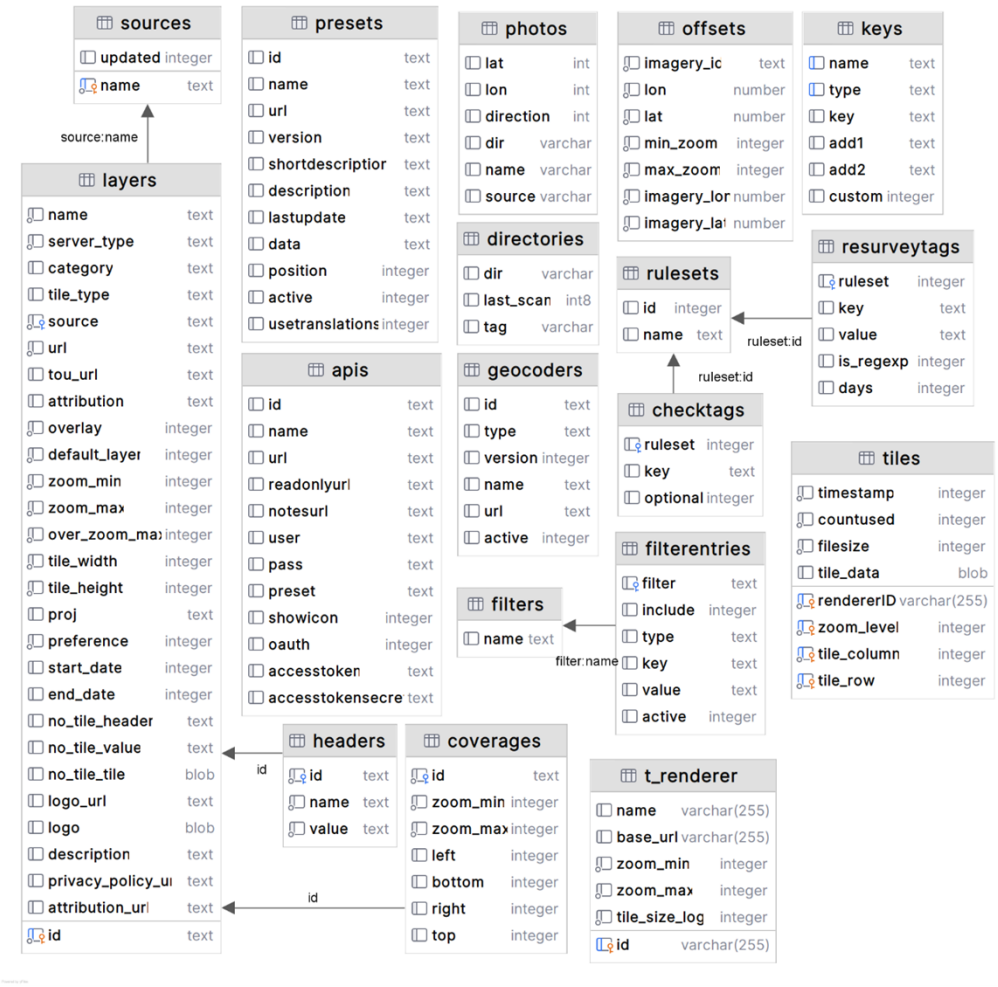


FIGURE 1 – Schéma physique

2.2 Analyse supplémentaire pour découvrir des Foreign Keys

2.2.1 Recherche dans le code source

Aucune clef étrangère implicite n'a été trouvée suite à l'analyse des requêtes `SELECT` avec la regex : `(FROM|from).*,.*(WHERE|where)`. Cette dernière permet de sélectionner les requêtes qui réalisent un `SELECT` sur plusieurs tables. Une seconde analyse des requêtes n'a donné aucun résultat pour le format de jointure SQL2. Une simple recherche du mot clef `JOIN` ne donne aucun résultat, autant dans les requêtes listées par SQLInspect ni dans le code source de l'application.

2.2.2 Recherche de patterns

Les tables `directories` et `photos` ont toutes les 2 un attribut « dir ». En analysant le code du fichier `PhotoIndex.java`, et plus précisément de la méthode

`private void indexDirectories()`, on remarque qu'une query sur la table `photos` utilise la valeur de l'attribut `dir` récupéré à partir d'un select sur la table `directories`. Il y a donc un lien implicite entre ces 2 tables. Toutefois la requête utilise le mot clef `LIKE` qui permet de récupérer les entrés où l'attribut « dir » de la table `photos` contient l'attribut `dir` de la table `directories` suivi ou non d'autres caractères (symbolisé par « % » dans la requête).

2.3 Schéma logique

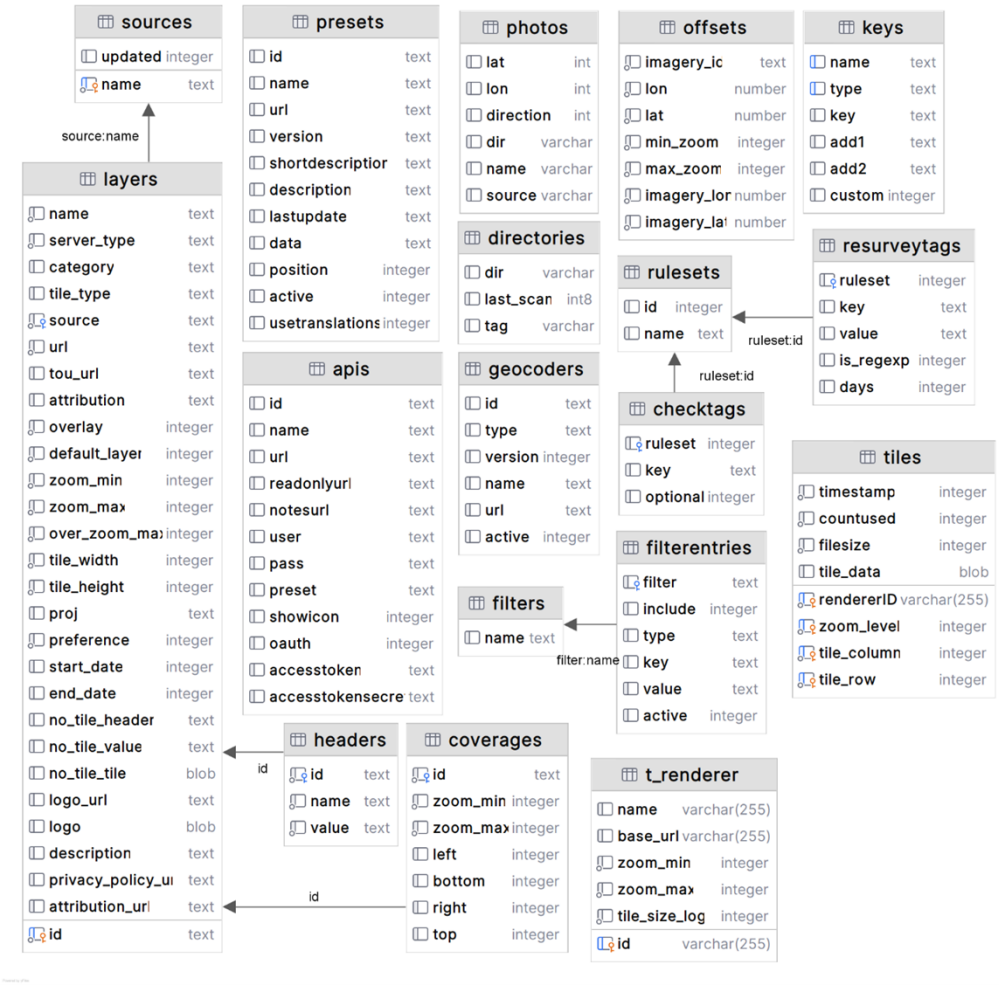


FIGURE 2 – Schéma logique

Note : ici le schéma logique est identique au schéma physique.

2.4 Schéma conceptuel

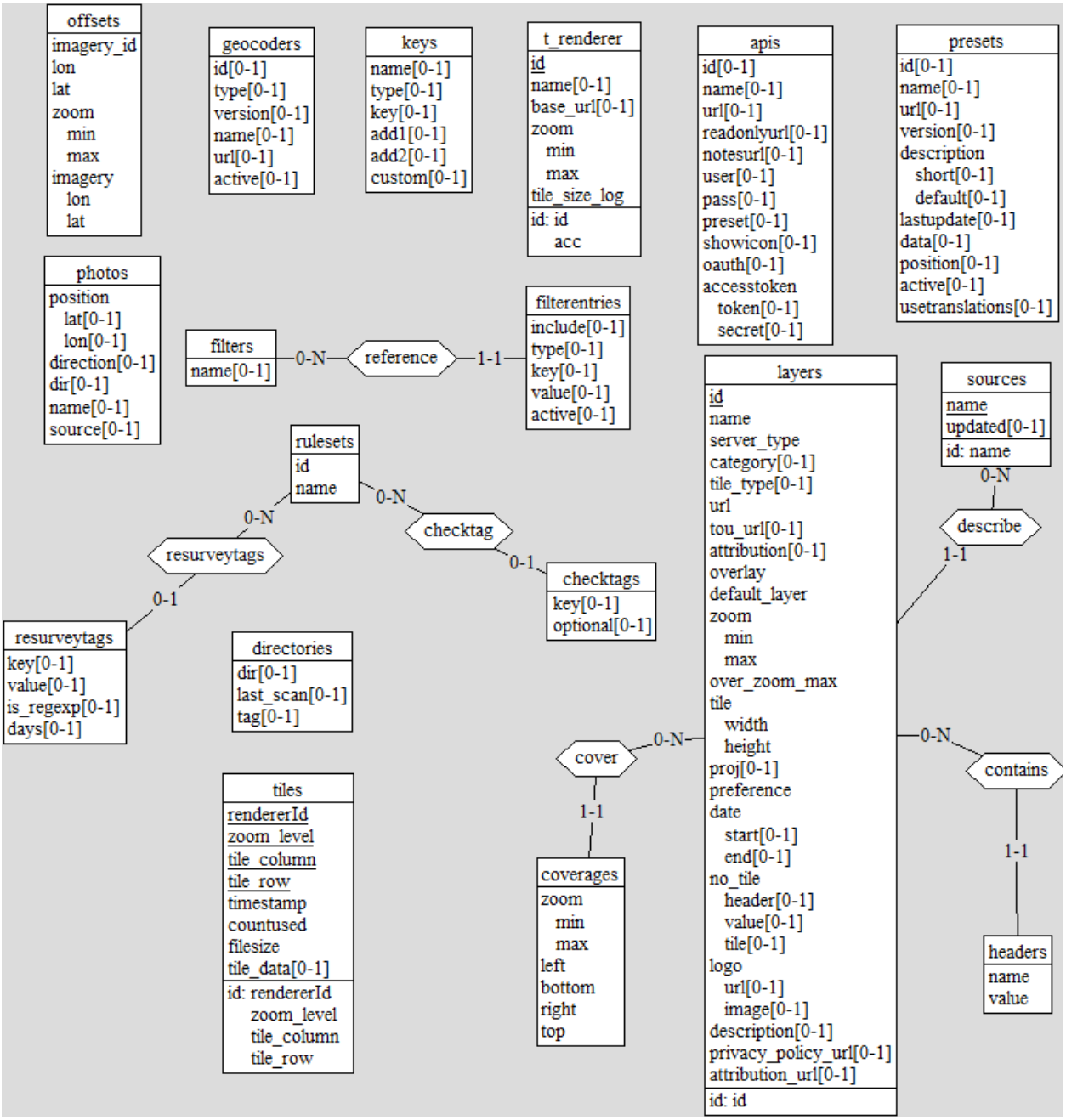


FIGURE 3 – Schéma conceptuel

3 Partie 2

3.1 Statistiques

Type	# trouvées par SOLInspect	# trouvées manuellement
SELECT	18	24
UPDATE	4	24
DELETE	2	27
INSERT	5	16
CREATE TABLE	18	18
CREATE INDEX	8	8
ALTER TABLE	24	24
Total	**79**	**141**

TABLE 1 – Nombre de requêtes trouvées par type.

Pour les requêtes de la colonne « # trouvées manuellement », un script `Python` a été utilisé pour effectuer une analyse statique des requêtes. Il permet de parcourir les différents fichiers Java et, grâce à une regex, les différents appels à SQLite tels que `db.update` ou encore `db.execSQL` sont filtrés et enregistrés dans des fichiers séparés.

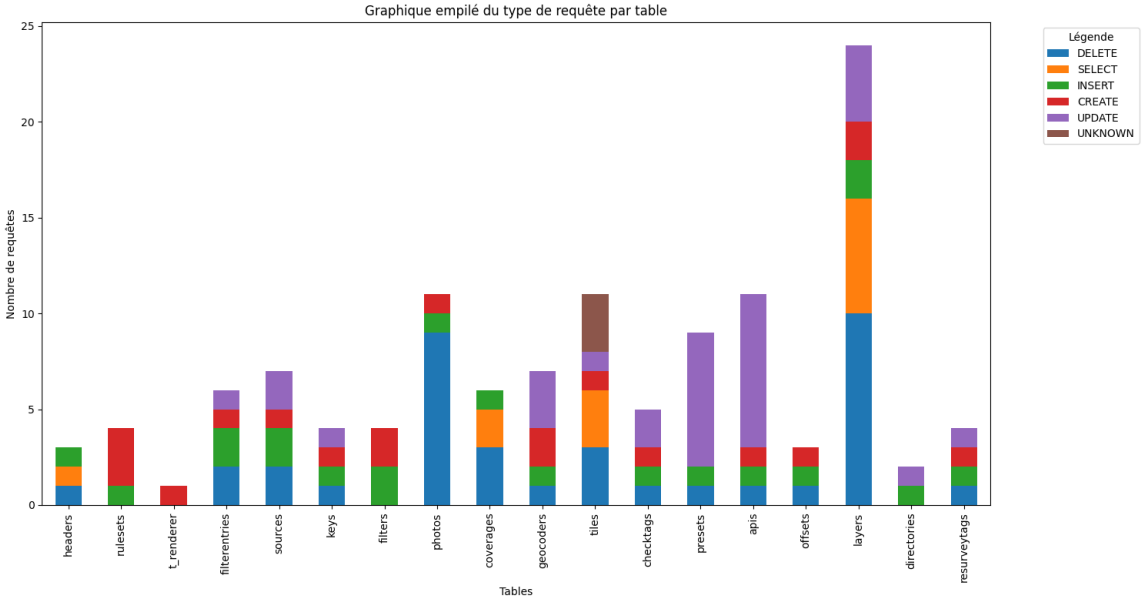
Pour la complexité de ces dernières, elles sont plutôt simples et courtes. On notera toutefois que les requêtes de création de table peuvent être plus longues, notamment pour la table `presets`, mais restent, malgré tout, simples à interpréter.

À propos de la répartition des requêtes dans le projet, elles sont regroupées dans 12 fichiers Java différents. Ces 12 fichiers sont répartis parmi 7 dossiers alors que le projet en compte 58.

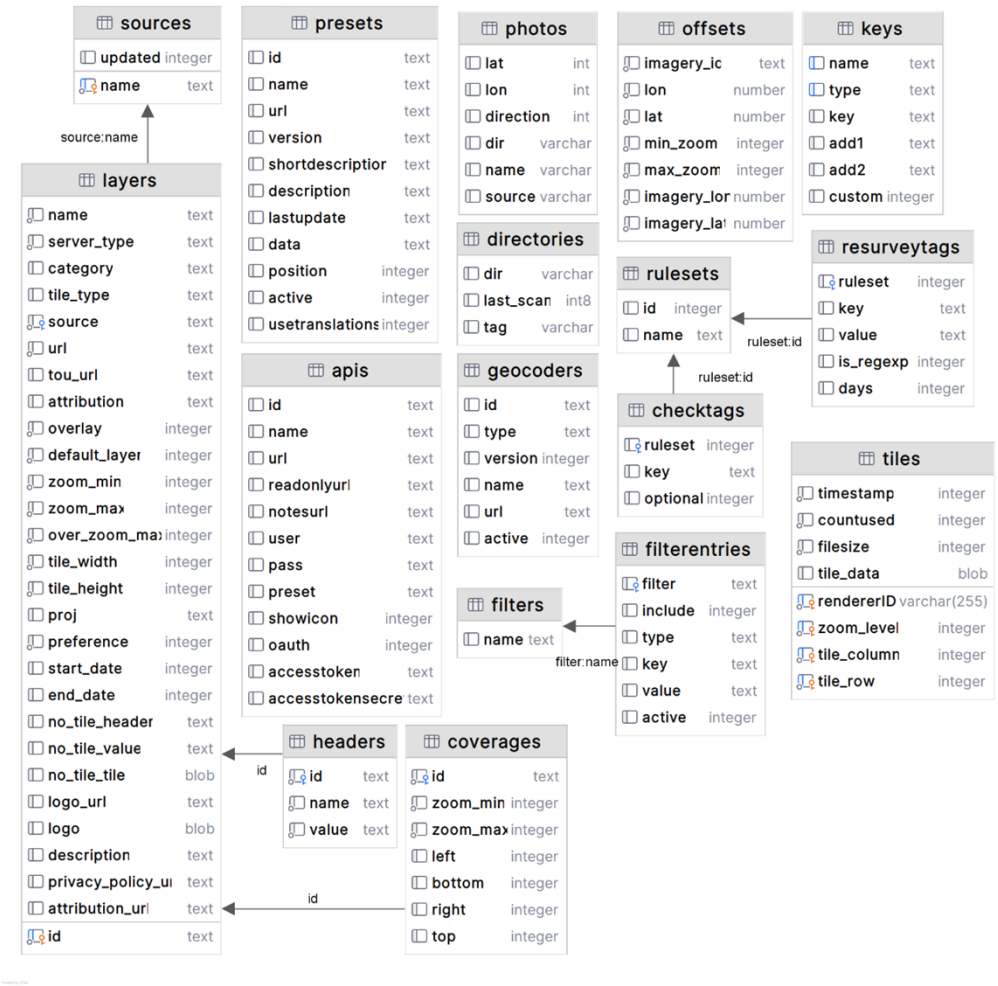
Dossier	Fichier
src/main/java/de/blau/android/filter	TagFilterActivity.java TagFilterDatabaseHelper.java
src/main/java/de/blau/android/imageryoffset	ImageryOffsetDatabase.java
src/main/java/de/blau/android/photos	PhotoIndex.java
src/main/java/de/blau/android/prefs	AdvancedPrefDatabase.java
src/main/java/de/blau/android/resources	KeyDatabaseHelper.java TileLayerDatabase.java
src/main/java/de/blau/android/services/util	MapTileProviderDataBase.java MBTileProviderDataBase.java
src/main/java/de/blau/android/validation	ValidatorRulesDatabase.java ValidatorRulesDatabaseHelper.java ValidatorRulesUI.java

TABLE 2 – Répartition des fichiers contenant des requêtes SQL.

3.2 Répartition du type de requêtes par tables



3.3 Sous-schéma logique



4 Partie 3

4.1 Scénario d'évolution n°1

4.1.1 Scénario

Renommer la colonne `name` de la table `headers`.

4.1.2 Modifications

Pour renommer la colonne `name` de la table `headers`, les étapes suivantes sont nécessaires. Ces changements concernent le code source et le schéma SQL dans différents fichiers du projet.

1. **Changer la constante qui représente le nom de la colonne**
   - **Fichier concerné** : `TileLayerDatabase.java`
   - **Emplacement** : Ligne 82
   - **Ancien code** :

```
private static final String HEADER_NAME_FIELD = "name";
```

   - **Nouveau code** :

```
private static final String HEADER_NAME_FIELD = "newName";
```

2. **Modifier la définition de la table dans la méthode `createHeadersTable`**
   - **Fichier concerné** : `TileLayerDatabase.java`
   - **Emplacement** : Ligne 161
   - **Ancien code** :

```
private void createHeadersTable(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE headers (id TEXT NOT NULL, name TEXT NOT
    ↪  NULL, value TEXT NOT NULL,"
            + " FOREIGN KEY(id) REFERENCES layers(id) ON DELETE
            ↪  CASCADE)");
    db.execSQL("CREATE INDEX headers_idx ON headers(id)");
}
```

   - **Nouveau code** :

```
private void createHeadersTable(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE headers (id TEXT NOT NULL, newName TEXT NOT
    ↪  NULL, value TEXT NOT NULL,"
            + " FOREIGN KEY(id) REFERENCES layers(id) ON DELETE
            ↪  CASCADE)");
    db.execSQL("CREATE INDEX headers_idx ON headers(id)");
}
```

3. **Mettre à jour la méthode `getHeadersById` pour refléter le changement**
   — **Fichier concerné** : `TileLayerDatabase.java`
   — **Emplacement** : Ligne 716
   — **Ancien code** :

```
@NonNull
private static Map<String, List<Header>> getHeadersById(SQLiteDatabase
↪  db, boolean overlay) {
    Map<String, List<Header>> headersById = new HashMap<>();
    try (Cursor headerCursor = db.rawQuery(
            "SELECT headers.id as id,headers.name as name,value FROM
            ↪  layers,headers WHERE headers.id=layers.id AND overlay=?",
            new String[] { boolean2intString(overlay) })) {
        if (headerCursor.getCount() ⩾ 1) {
            initHeaderFieldIndices(headerCursor);
            boolean haveEntry = headerCursor.moveToFirst();
            while (haveEntry) {
                String id = headerCursor.getString(headerIdFieldIndex);
                List<Header> headers = headersById.get(id);
                if (headers == null) {
                    headers = new ArrayList<>();
                    headersById.put(id, headers);
                }
                headers.add(getHeaderFromCursor(headerCursor));
                haveEntry = headerCursor.moveToNext();
            }
        }
    }
    return headersById;
}
```

   — **Nouveau code** :

```
@NonNull
private static Map<String, List<Header>> getHeadersById(SQLiteDatabase
↪  db, boolean overlay) {
    Map<String, List<Header>> headersById = new HashMap<>();
    try (Cursor headerCursor = db.rawQuery(
            "SELECT headers.id as id, headers.newName as newName, value
            ↪  FROM layers, headers WHERE headers.id=layers.id AND
            ↪  overlay=?",
            new String[] { boolean2intString(overlay) })) {
        if (headerCursor.getCount() ⩾ 1) {
            initHeaderFieldIndices(headerCursor);
            boolean haveEntry = headerCursor.moveToFirst();
            while (haveEntry) {
                String id = headerCursor.getString(headerIdFieldIndex);
                List<Header> headers = headersById.get(id);
                if (headers == null) {
                    headers = new ArrayList<>();
                    headersById.put(id, headers);
                }
                headers.add(getHeaderFromCursor(headerCursor));
                haveEntry = headerCursor.moveToNext();
            }
        }
    }
    return headersById;
}
```

4.2 Scénario d'évolution n°2

4.2.1 Scénario

Supprimer la table `filters`.

4.2.2 Modifications

Tous les schémas sont concernés par cette modification, il faut donc y supprimer la table `filters`.

1. **Modifier la méthode `onCreate`**
   - **Fichier concerné** : `TagFilterDatabaseHelper.java`
   - **Emplacement** : Ligne 30
   - **Ancien code** :

```
@Override
public void onCreate(SQLiteDatabase db) {
    try {
        db.execSQL("CREATE TABLE filters (name TEXT)");
        db.execSQL("INSERT INTO filters VALUES ('Default')");
        db.execSQL(
                "CREATE TABLE filterentries (filter TEXT, include INTEGER
                ↪  DEFAULT 0, type TEXT DEFAULT '*', key TEXT DEFAULT
                ↪  '*', value TEXT DEFAULT '*', active INTEGER DEFAULT
                ↪  0, FOREIGN KEY(filter) REFERENCES filters(name))");
    } catch (SQLException e) {
        Log.w(DEBUG_TAG, "Problem creating database", e);
    }
}
```

   - **Nouveau code** :

```
@Override
public void onCreate(SQLiteDatabase db) {
    try {
        db.execSQL("CREATE TABLE filters (name TEXT)");
        db.execSQL("INSERT INTO filters VALUES ('Default')");
        db.execSQL(
                "CREATE TABLE filterentries (filter TEXT, include INTEGER
                ↪  DEFAULT 0, type TEXT DEFAULT '*', key TEXT DEFAULT
                ↪  '*', value TEXT DEFAULT '*', active INTEGER DEFAULT
                ↪  0, FOREIGN KEY(filter) REFERENCES filters(name))");
```

```
    } catch (SQLException e) {
        Log.w(DEBUG_TAG, "Problem creating database", e);
    }
}
```

4.2.3 Scénario

Renommer la colonne `name` de la table `headers`.

4.2.4 Modifications

Pour renommer la colonne `name` de la table `headers`, les étapes suivantes sont nécessaires. Ces changements concernent le code source et le schéma SQL dans différents fichiers du projet.

1. **Changer la constante qui représente le nom de la colonne**
   - **Fichier concerné** : `TileLayerDatabase.java`
   - **Emplacement** : Ligne 82
   - **Ancien code** :

```
private static final String HEADER_NAME_FIELD = "name";
```

   - **Nouveau code** :

```
private static final String HEADER_NAME_FIELD = "newName";
```

2. **Modifier la définition de la table dans la méthode `createHeadersTable`**
   - **Fichier concerné** : `TileLayerDatabase.java`
   - **Emplacement** : Ligne 161
   - **Ancien code** :

```
private void createHeadersTable(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE headers (id TEXT NOT NULL, name TEXT NOT
    ↪  NULL, value TEXT NOT NULL,"
            + " FOREIGN KEY(id) REFERENCES layers(id) ON DELETE
            ↪  CASCADE)");
    db.execSQL("CREATE INDEX headers_idx ON headers(id)");
}
```

— **Nouveau code** :

```
private void createHeadersTable(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE headers (id TEXT NOT NULL, newName TEXT NOT
    ↪  NULL, value TEXT NOT NULL,"
            + " FOREIGN KEY(id) REFERENCES layers(id) ON DELETE
            ↪  CASCADE)");
    db.execSQL("CREATE INDEX headers_idx ON headers(id)");
}
```

4.3 Scénario d'évolution n°3

4.3.1 Scénario

Modifier le type de la colonne `id` de la table `coverages` de `TEXT` à `INTEGER`.

4.3.2 Modifications

Les schémas physiques et logiques sont concernés par cette modification, il faut donc les éditer pour faire propager les modifications faites dans le code.

1. **Création de la table dans la méthode `onCreate`**
   - **Fichier concerné** : `TileLayerDatabase.java`
   - **Emplacement** : Ligne 114
   - **Ancien code** :

```
@Override
public void onCreate(SQLiteDatabase db) {
    try {
        db.execSQL("CREATE TABLE sources (name TEXT NOT NULL PRIMARY KEY,
        ↪  updated INTEGER)");
        addSource(db, SOURCE_JOSM_IMAGERY);
        addSource(db, SOURCE_CUSTOM);
        addSource(db, SOURCE_MANUAL);

        db.execSQL(
            "CREATE TABLE layers (id TEXT NOT NULL PRIMARY KEY, name TEXT
            ↪  NOT NULL, server_type TEXT NOT NULL, category TEXT
            ↪  DEFAULT NULL, tile_type TEXT DEFAULT NULL,"
            + " source TEXT NOT NULL, url TEXT NOT NULL," + " tou_url
            ↪  TEXT, attribution TEXT, overlay INTEGER NOT NULL DEFAULT
            ↪  0,"
            + " default_layer INTEGER NOT NULL DEFAULT 0, zoom_min
            ↪  INTEGER NOT NULL DEFAULT 0, zoom_max INTEGER NOT NULL
            ↪  DEFAULT 18,"
            + " over_zoom_max INTEGER NOT NULL DEFAULT 4, tile_width
            ↪  INTEGER NOT NULL DEFAULT 256, tile_height INTEGER NOT
            ↪  NULL DEFAULT 256,"
            + " proj TEXT DEFAULT NULL, preference INTEGER NOT NULL
            ↪  DEFAULT 0, start_date INTEGER DEFAULT NULL, end_date
            ↪  INTEGER DEFAULT NULL,"
            + " no_tile_header TEXT DEFAULT NULL, no_tile_value TEXT
            ↪  DEFAULT NULL, no_tile_tile BLOB DEFAULT NULL, logo_url
            ↪  TEXT DEFAULT NULL, logo BLOB DEFAULT NULL,"
```

```
            + " description TEXT DEFAULT NULL, privacy_policy_url TEXT
            ↪  DEFAULT NULL, attribution_url TEXT DEFAULT NULL, FOREIGN
            ↪  KEY(source) REFERENCES sources(name) ON DELETE
            ↪  CASCADE)");
        db.execSQL("CREATE INDEX layers_overlay_idx ON layers(overlay)");
        db.execSQL("CREATE INDEX layers_source_idx ON layers(source)");
        db.execSQL("CREATE TABLE coverages (id TEXT NOT NULL, zoom_min
        ↪  INTEGER NOT NULL DEFAULT 0, zoom_max INTEGER NOT NULL DEFAULT
        ↪  18,"
                + " left INTEGER DEFAULT NULL, bottom INTEGER DEFAULT
                ↪  NULL, right INTEGER DEFAULT NULL, top INTEGER DEFAULT
                ↪  NULL,"
                + " FOREIGN KEY(id) REFERENCES layers(id) ON DELETE
                ↪  CASCADE)");
        db.execSQL("CREATE INDEX coverages_idx ON coverages(id)");
        createHeadersTable(db);
    } catch (SQLException e) {
        Log.w(DEBUG_TAG, "Problem creating database", e);
    }
}
```

— **Nouveau code** :

```
@Override
public void onCreate(SQLiteDatabase db) {
    try {
        db.execSQL("CREATE TABLE sources (name TEXT NOT NULL PRIMARY KEY,
        ↪  updated INTEGER)");
        addSource(db, SOURCE_JOSM_IMAGERY);
        addSource(db, SOURCE_CUSTOM);
        addSource(db, SOURCE_MANUAL);

        db.execSQL(
                "CREATE TABLE layers (id TEXT NOT NULL PRIMARY KEY, name
                ↪  TEXT NOT NULL, server_type TEXT NOT NULL, category
                ↪  TEXT DEFAULT NULL, tile_type TEXT DEFAULT NULL,"
                + " source TEXT NOT NULL, url TEXT NOT NULL," + " tou_url
                ↪  TEXT, attribution TEXT, overlay INTEGER NOT NULL
                ↪  DEFAULT 0,"
                + " default_layer INTEGER NOT NULL DEFAULT 0, zoom_min
                ↪  INTEGER NOT NULL DEFAULT 0, zoom_max INTEGER NOT NULL
                ↪  DEFAULT 18,"
```

```
                + " over_zoom_max INTEGER NOT NULL DEFAULT 4, tile_width
                ↪  INTEGER NOT NULL DEFAULT 256, tile_height INTEGER NOT
                ↪  NULL DEFAULT 256,"
                + " proj TEXT DEFAULT NULL, preference INTEGER NOT NULL
                ↪  DEFAULT 0, start_date INTEGER DEFAULT NULL, end_date
                ↪  INTEGER DEFAULT NULL,"
                + " no_tile_header TEXT DEFAULT NULL, no_tile_value TEXT
                ↪  DEFAULT NULL, no_tile_tile BLOB DEFAULT NULL,
                ↪  logo_url TEXT DEFAULT NULL, logo BLOB DEFAULT NULL,"
                + " description TEXT DEFAULT NULL, privacy_policy_url
                ↪  TEXT DEFAULT NULL, attribution_url TEXT DEFAULT NULL,
                ↪  FOREIGN KEY(source) REFERENCES sources(name) ON
                ↪  DELETE CASCADE)");
        db.execSQL("CREATE INDEX layers_overlay_idx ON layers(overlay)");
        db.execSQL("CREATE INDEX layers_source_idx ON layers(source)");
        db.execSQL("CREATE TABLE coverages (id TEXT NOT NULL, zoom_min
        ↪  INTEGER NOT NULL DEFAULT 0, zoom_max INTEGER NOT NULL DEFAULT
        ↪  18,"
                + " left INTEGER DEFAULT NULL, bottom INTEGER DEFAULT
                ↪  NULL, right INTEGER DEFAULT NULL, top INTEGER DEFAULT
                ↪  NULL,"
                + " FOREIGN KEY(id) REFERENCES layers(id) ON DELETE
                ↪  CASCADE)");
        db.execSQL("CREATE INDEX coverages_idx ON coverages(id)");
        createHeadersTable(db);
    } catch (SQLException e) {
        Log.w(DEBUG_TAG, "Problem creating database", e);
    }
}
```

2. **Modifier les queries de la méthode TileLayerSource**
   - **Fichier concerné** : `TileLayerSource.java`
   - **Emplacement** : Ligne 370
   - **Ancien code** :

```
@Nullable
public static TileLayerSource getLayer(@NonNull Context context,
↪  @NonNull SQLiteDatabase db, @NonNull String id) {
    TileLayerSource layer = null;
    try (Cursor providerCursor = db.query(COVERAGES_TABLE, null,
    ↪  ID_FIELD + "='" + id + "'", null, null, null, null)) {
```

```
        Provider provider = getProviderFromCursor(providerCursor);
        try (Cursor layerCursor = db.query(LAYERS_TABLE, null, ID_FIELD
        ↪  + "='" + id + "'", null, null, null, null)) {
            if (layerCursor.getCount() ⩾ 1) {
                boolean haveEntry = layerCursor.moveToFirst();
                if (haveEntry) {
                    initLayerFieldIndices(layerCursor);
                    layer = getLayerFromCursor(context, provider,
                    ↪  layerCursor);
                    setHeadersForLayer(db, layer);
                }
            }
        }
    }
    return layer;
}
```

— **Nouveau code** :

```
@Nullable
public static TileLayerSource getLayer(@NonNull Context context,
↪  @NonNull SQLiteDatabase db, @NonNull String id) {
    TileLayerSource layer = null;
    try (Cursor providerCursor = db.query(COVERAGES_TABLE, null,
    ↪  ID_FIELD + "=" + id, null, null, null, null)) {
        Provider provider = getProviderFromCursor(providerCursor);
        try (Cursor layerCursor = db.query(LAYERS_TABLE, null, ID_FIELD
        ↪  + "=" + id, null, null, null, null)) {
            if (layerCursor.getCount() ⩾ 1) {
                boolean haveEntry = layerCursor.moveToFirst();
                if (haveEntry) {
                    initLayerFieldIndices(layerCursor);
                    layer = getLayerFromCursor(context, provider,
                    ↪  layerCursor);
                    setHeadersForLayer(db, layer);
                }
            }
        }
    }
    return layer;
}
```

3. **Modifier la méthode `getLayerWithUrl`**
   — **Fichier concerné** : `TileLayerDatabase.java`
   — **Emplacement** : Ligne 419
   — **Ancien code** :

```
@Nullable
public static TileLayerSource getLayerWithUrl(@NonNull Context context,
↪  @NonNull SQLiteDatabase db, @NonNull String url) {
    TileLayerSource layer = null;
    try (Cursor layerCursor = db.query(LAYERS_TABLE, null,
    ↪  TILE_URL_FIELD + "=?", new String[] { url }, null, null, null)) {
        if (layerCursor.getCount() ⩾ 1) {
            boolean haveEntry = layerCursor.moveToFirst();
            if (haveEntry) {
                initLayerFieldIndices(layerCursor);
                String id = layerCursor.getString(idLayerFieldIndex);
                try (Cursor providerCursor = db.query(COVERAGES_TABLE,
                ↪  null, ID_FIELD + "='" + id + "'", null, null, null,
                ↪  null)) {
                    Provider provider =
                    ↪  getProviderFromCursor(providerCursor);
                    layer = getLayerFromCursor(context, provider,
                    ↪  layerCursor);
                    setHeadersForLayer(db, layer);
                    }
                }
            }
        }
    return layer;
}
```

   — **Nouveau code** :

```
@Nullable
public static TileLayerSource getLayerWithUrl(@NonNull Context context,
↪  @NonNull SQLiteDatabase db, @NonNull String url) {
    TileLayerSource layer = null;
    try (Cursor layerCursor = db.query(LAYERS_TABLE, null,
    ↪  TILE_URL_FIELD + "=?", new String[] { url }, null, null, null)) {
        if (layerCursor.getCount() ⩾ 1) {
            boolean haveEntry = layerCursor.moveToFirst();
```

```
            if (haveEntry) {
                initLayerFieldIndices(layerCursor);
                String id = layerCursor.getString(idLayerFieldIndex);
                try (Cursor providerCursor = db.query(COVERAGES_TABLE,
                ↪  null, ID_FIELD + "=" + id, null, null, null, null)) {
                    Provider provider =
                    ↪  getProviderFromCursor(providerCursor);
                    layer = getLayerFromCursor(context, provider,
                    ↪  layerCursor);
                    setHeadersForLayer(db, layer);
                }
            }
        }
    }
    return layer;
}
```

4. **Mettre à jour la méthode `getCoveragesById` pour refléter le changement**
   La colonne id de layers étant de type text, on doit convertir le type dans la requête en utilisant CAST
   — **Fichier concerné** : `TileLayerDatabase.java`
   — **Emplacement** : Ligne 746
   — **Ancien code** :

```
private static MultiHashMap<String, CoverageArea>
↪  getCoveragesById(SQLiteDatabase db, boolean overlay) {
    MultiHashMap<String, CoverageArea> coveragesById = new
    ↪  MultiHashMap<>();
    try (Cursor coverageCursor = db.rawQuery(
            "SELECT coverages.id as
            ↪  id,left,bottom,right,top,coverages.zoom_min as
            ↪  zoom_min,coverages.zoom_max as zoom_max FROM
            ↪  layers,coverages WHERE coverages.id=layers.id AND
            ↪  overlay=?",
            new String[] { boolean2intString(overlay) })) {
        if (coverageCursor.getCount() ⩾ 1) {
            initCoverageFieldIndices(coverageCursor);
            boolean haveEntry = coverageCursor.moveToFirst();
            while (haveEntry) {
                String id =
                ↪  coverageCursor.getString(coverageIdFieldIndex);
```

```
                CoverageArea ca = getCoverageFromCursor(coverageCursor);
                coveragesById.add(id, ca);
                haveEntry = coverageCursor.moveToNext();
            }
        }
    }
    return coveragesById;
}
```

— **Nouveau code** :

```
private static MultiHashMap<String, CoverageArea>
↪  getCoveragesById(SQLiteDatabase db, boolean overlay) {
    MultiHashMap<String, CoverageArea> coveragesById = new
    ↪  MultiHashMap<>();
    try(Cursor coverageCursor = db.rawQuery(

    "SELECT coverages.id as id, left, bottom, right, top,
    ↪  coverages.zoom_min as zoom_min, coverages.zoom_max as zoom_max "
    ↪  +

    "FROM layers, coverages " +

    "WHERE CAST(coverages.id AS TEXT) = layers.id AND overlay=?",

    new String[] { boolean2intString(overlay) })) {
        if (coverageCursor.getCount() ⩾ 1) {
            initCoverageFieldIndices(coverageCursor);
            boolean haveEntry = coverageCursor.moveToFirst();
            while (haveEntry) {
                String id =
                ↪  coverageCursor.getString(coverageIdFieldIndex);
                CoverageArea ca = getCoverageFromCursor(coverageCursor);
                coveragesById.add(id, ca);
                haveEntry = coverageCursor.moveToNext();
            }
        }
    }
    return coveragesById;
}
```

5. **Mettre à jour la méthode `deleteCoverage` pour refléter le changement**
   — **Fichier concerné** : `TileLayerDatabase.java`
   — **Emplacement** : Ligne 935
   — **Ancien code** :

```
public static void deleteCoverage(@NonNull SQLiteDatabase db, @NonNull
↪  String id) {
    db.delete(COVERAGES_TABLE, "id=?", new String[] { id });
}
```

   — **Nouveau code** :

```
public static void deleteCoverage(@NonNull SQLiteDatabase db, @NonNull
↪  String id) {
    db.delete(COVERAGES_TABLE, "id=?", new String[] { id });
}
```

6. **Mettre à jour la methode `deleteHeader` pour refléter le changement**
   — **Fichier concerné** : `TileLayerDatabase.java`
   — **Emplacement** : Ligne 960
   — **Ancien code** :

```
public static void deleteHeader(@NonNull SQLiteDatabase db, @NonNull
↪  String id) {
    db.delete(COVERAGES_TABLE, "id=?", new String[] { id });
}
```

   — **Nouveau code** :

```
public static void deleteHeader(@NonNull SQLiteDatabase db, @NonNull
↪  String id) {
    db.delete(COVERAGES_TABLE, "id=?", new String[] { id });
}
```

4.4 Scénario d'évolution n°4

4.4.1 Scénario

Fusionner deux tables en une seule : fusionner les tables `filterentries` et `filters`.

Raison éventuelle : Ces tables sont liées par une clé étrangère (`filter`) et décrivent toutes les deux les filtres et leurs entrées.

Les tables seront fusionnées en une table commune : filters.

4.4.2 Modifications

Tous les schémas sont concernés par cette modification, il faut donc les éditer pour faire propager les modifications faites dans le code.

1. **Modifier la méthode `onCreate` qui crée les deux tables**
   - **Fichier concerné** : `TagFilterDatabaseHelper.java`
   - **Emplacement** : Ligne 28
   - **Ancien code** :

```
@Override
public void onCreate(SQLiteDatabase db) {
    try {
        db.execSQL("CREATE TABLE filters (name TEXT)");
        db.execSQL("INSERT INTO filters VALUES ('Default')");
        db.execSQL(
                "CREATE TABLE filterentries (filter TEXT, include INTEGER
                ↪  DEFAULT 0, type TEXT DEFAULT '*', key TEXT DEFAULT
                ↪  '*', value TEXT DEFAULT '*', active INTEGER DEFAULT
                ↪  0, FOREIGN KEY(filter) REFERENCES filters(name))");
    } catch (SQLException e) {
        Log.w(DEBUG_TAG, "Problem creating database", e);
    }
}
```

   - **Nouveau code** :

```
@Override
public void onCreate(SQLiteDatabase db) {
    try {
        db.execSQL(
                "CREATE TABLE filters (" +
```

```
                "id INTEGER PRIMARY KEY AUTOINCREMENT, " + // Identifiant
                ↪ unique pour chaque entrée
                "name TEXT NOT NULL, " +                  // Nom du
                ↪ filtre
                "include INTEGER DEFAULT 0, " +           // Inclusion
                "type TEXT DEFAULT '*', " +               // Type
                "key TEXT DEFAULT '*', " +                // Clé
                "value TEXT DEFAULT '*', " +              // Valeur
                "active INTEGER DEFAULT 0" +              // Indicateur
                ↪ actif
                ")");
        // Insérer le filtre par défaut
        db.execSQL("INSERT INTO filters (name, include, type, key, value,
        ↪ active) VALUES ('Default', 0, '*', '*', '*', 0)");
    } catch (SQLException e) {
        Log.w(DEBUG_TAG, "Problem creating database", e);
    }
}
```

2. **Adapter la query suivante**
   - **Fichier concerné** : `TagFilterDatabaseHelper.java`
   - **Emplacement** : Ligne 145
   - **Ancien code** :

```
@Override
public void init(Context context) {
    try (TagFilterDatabaseHelper tfDb = new
    ↪ TagFilterDatabaseHelper(context); SQLiteDatabase mDatabase =
    ↪ tfDb.getReadableDatabase()) {
        //
        filter.clear();
        Cursor dbresult = mDatabase.query("filterentries", new String[]
        ↪ { "include", "type", "key", "value", "active" }, "filter =
        ↪ ?",
                new String[] { DEFAULT_FILTER }, null, null, null);
        dbresult.moveToFirst();
        for (int i = 0; i < dbresult.getCount(); i++) {
            try {
                filter.add(new FilterEntry(dbresult.getInt(0) == 1,
                ↪ dbresult.getString(1), dbresult.getString(2),
                ↪ dbresult.getString(3),
```

```
                        dbresult.getInt(4) == 1));
            } catch (PatternSyntaxException psex) {
                Log.e(DEBUG_TAG, "exception getting FilterEntry " +
                ↪  psex.getMessage());
                if (context instanceof Activity) {
                    ScreenMessage.barError((Activity) context,
                            context.getString(R.string.toast_invalid_fil
                            ↪  ter_regexp, dbresult.getString(2),
                            ↪  dbresult.getString(3)));
                }
            }
            dbresult.moveToNext();
        }
        dbresult.close();
    }
}
```

— **Nouveau code** :

```
Cursor dbresult = mDatabase.query("filters", new String[] { "include",
↪  "type", "key", "value", "active" }, "name = ?", new String[] {
↪  DEFAULT_FILTER }, null, null, null);
```

4.5 Scénario d'évolution n°5

4.5.1 Scénario

Diviser la table `photos` en `photo_metadata` et en `photo_location`.

Raison éventuelle : La table photos contient à la fois des métadonnées (`name`, `dir` et `source`) et des informations géospatiales (`lat`, `lon`, `direction`, `orientation`).

4.5.2 Modifications

Tous les schémas sont concernés par cette modification, il faut donc les éditer pour faire propager les modifications faites dans le code.

Modifier la méthode onCreate du fichier `PhotoIndex.java`.

1. **Modifier la méthode** `onCreate`
   - **Fichier concerné** : `PhotoIndex.java`
   - **Emplacement** : Ligne 102
   - **Ancien code** :

```
@Override
public synchronized void onCreate(SQLiteDatabase db) {
    Log.d(DEBUG_TAG, "Creating photo index DB");
    db.execSQL("CREATE TABLE IF NOT EXISTS " + PHOTOS_TABLE
            + " (lat int, lon int, direction int DEFAULT NULL, dir
            ↪  VARCHAR, name VARCHAR, source VARCHAR DEFAULT NULL,
            ↪  orientation int DEFAULT 0);");
    db.execSQL("CREATE INDEX latidx ON " + PHOTOS_TABLE + " (lat)");
    db.execSQL("CREATE INDEX lonidx ON " + PHOTOS_TABLE + " (lon)");
    db.execSQL("CREATE TABLE IF NOT EXISTS " + SOURCES_TABLE + " (dir
    ↪  VARCHAR, last_scan int8, tag VARCHAR DEFAULT NULL);");
    initSource(db, DCIM, null);
    initSource(db, Paths.DIRECTORY_PATH_VESPUCCI, null);
    initSource(db, OSMTRACKER, null);
    initSource(db, MEDIA_STORE, "");
}
```

   - **Nouveau code** :

```
@Override
public synchronized void onCreate(SQLiteDatabase db) {
    Log.d(DEBUG_TAG, "Creating photo index DB");
```

```
    // Créer la table pour les métadonnées des photos
    db.execSQL("CREATE TABLE IF NOT EXISTS photo_metadata ("
            + "id INTEGER PRIMARY KEY AUTOINCREMENT, "
            + "name VARCHAR, "
            + "dir VARCHAR, "
            + "source VARCHAR DEFAULT NULL"
            + ");");

    // Créer la table pour les données géospatiales des photos
    db.execSQL("CREATE TABLE IF NOT EXISTS photo_location ("
            + "id INTEGER PRIMARY KEY AUTOINCREMENT, "
            + "photo_id INTEGER NOT NULL, "
            + "lat INTEGER, "
            + "lon INTEGER, "
            + "direction INTEGER DEFAULT NULL, "
            + "orientation INTEGER DEFAULT 0, "
            + "FOREIGN KEY(photo_id) REFERENCES photo_metadata(id) ON
            ↪  DELETE CASCADE"
            + ");");
    db.execSQL("CREATE INDEX latidx ON photo_location (lat);");
    db.execSQL("CREATE INDEX lonidx ON photo_location (lon);");
    db.execSQL("CREATE TABLE IF NOT EXISTS " + SOURCES_TABLE
            + " (dir VARCHAR, last_scan int8, tag VARCHAR DEFAULT
            ↪  NULL);");
        initSource(db, DCIM, null);
    initSource(db, Paths.DIRECTORY_PATH_VESPUCCI, null);
    initSource(db, OSMTRACKER, null);
    initSource(db, MEDIA_STORE, "");
}
```

4.6 Scénario d'évolution n°6

4.6.1 Scénario

Suppression de la table `t_renderer`.

4.6.2 Modifications

Tous les schémas sont concernés par cette modification, il faut donc les éditer pour faire propager les modifications faites dans le code.

1. **supprimer l'exécution de la requête de création de la table `t_renderer`**
   — **Fichier concerné** : `MapTileProviderDataBase.java`
   — **Emplacement** : Ligne 428
   — **Ancien code** :

```
@Override
public void onCreate(SQLiteDatabase db) {
    try {
        db.execSQL(T_RENDERER_CREATE_COMMAND);
        db.execSQL(T_FSCACHE_CREATE_COMMAND);
    } catch (SQLException e) {
        Log.w(MapTileFilesystemProvider.DEBUG_TAG, "Problem creating
        ↪  database", e);
    }
}
```

   — **Nouveau code** :

```
@Override
public void onCreate(SQLiteDatabase db) {
    try {
        db.execSQL(T_FSCACHE_CREATE_COMMAND);
    } catch (SQLException e) {
        Log.w(MapTileFilesystemProvider.DEBUG_TAG, "Problem creating
        ↪  database", e);
    }
}
```

2. **Étant donné que la table n'est pas utilisée par le programme, il est donc intéressant de supprimer ses dernières références présentes dans le même**

fichier. On peut donc également supprimer les lignes de 58 à 64 ainsi que la déclaration de la variable `T_RENDERER_CREATE_COMMAND` (lignes 72 à 74).

— **Fichier concerné** : `MapTileProviderDataBase.java`

— **Emplacements** : Lignes 58-64 et Lignes 72-74

— **Ancien code** :

```
private static final String T_RENDERER                = "t_renderer";
private static final String T_RENDERER_ID             = "id";
private static final String T_RENDERER_NAME           = "name";
private static final String T_RENDERER_BASE_URL       = "base_url";
private static final String T_RENDERER_ZOOM_MIN       = "zoom_min";
private static final String T_RENDERER_ZOOM_MAX       = "zoom_max";
private static final String T_RENDERER_TILE_SIZE_LOG = "tile_size_log";

private static final String T_FSCACHE_CREATE_COMMAND = "CREATE TABLE IF
↪  NOT EXISTS " + T_FSCACHE + " (" + T_FSCACHE_RENDERER_ID + "
↪  VARCHAR(255) NOT NULL,"
        + T_FSCACHE_ZOOM_LEVEL + " INTEGER NOT NULL," + T_FSCACHE_TILE_X
        ↪  + " INTEGER NOT NULL," + T_FSCACHE_TILE_Y + " INTEGER NOT
        ↪  NULL,"
        + T_FSCACHE_TIMESTAMP + " INTEGER NOT NULL," +
        ↪  T_FSCACHE_USAGECOUNT + " INTEGER NOT NULL DEFAULT 1," +
        ↪  T_FSCACHE_FILESIZE + " INTEGER NOT NULL,"
        + T_FSCACHE_DATA + " BLOB," + " PRIMARY KEY(" +
        ↪  T_FSCACHE_RENDERER_ID + "," + T_FSCACHE_ZOOM_LEVEL + "," +
        ↪  T_FSCACHE_TILE_X + "," + T_FSCACHE_TILE_Y
        + ")" + ");";

private static final String T_RENDERER_CREATE_COMMAND = "CREATE TABLE IF
↪  NOT EXISTS " + T_RENDERER + " (" + T_RENDERER_ID + " VARCHAR(255)
↪  PRIMARY KEY,"
        + T_RENDERER_NAME + " VARCHAR(255)," + T_RENDERER_BASE_URL + "
        ↪  VARCHAR(255)," + T_RENDERER_ZOOM_MIN + " INTEGER NOT NULL,"
        ↪  + T_RENDERER_ZOOM_MAX
        + " INTEGER NOT NULL," + T_RENDERER_TILE_SIZE_LOG + " INTEGER NOT
        ↪  NULL" + ");";
```

— **Nouveau code** :

```
private static final String T_FSCACHE_CREATE_COMMAND = "CREATE TABLE IF
↪  NOT EXISTS " + T_FSCACHE + " (" + T_FSCACHE_RENDERER_ID + "
↪  VARCHAR(255) NOT NULL,"
        + T_FSCACHE_ZOOM_LEVEL + " INTEGER NOT NULL," + T_FSCACHE_TILE_X
        ↪  + " INTEGER NOT NULL," + T_FSCACHE_TILE_Y + " INTEGER NOT
        ↪  NULL,"
        + T_FSCACHE_TIMESTAMP + " INTEGER NOT NULL," +
        ↪  T_FSCACHE_USAGECOUNT + " INTEGER NOT NULL DEFAULT 1," +
        ↪  T_FSCACHE_FILESIZE + " INTEGER NOT NULL,"
        + T_FSCACHE_DATA + " BLOB," + " PRIMARY KEY(" +
        ↪  T_FSCACHE_RENDERER_ID + "," + T_FSCACHE_ZOOM_LEVEL + "," +
        ↪  T_FSCACHE_TILE_X + "," + T_FSCACHE_TILE_Y
        + ")" + ");";
```

4.7 Scénario d'évolution n°7

4.7.1 Scénario

Ajout d'une colonne `rate_limit` à la table `apis`. Cette valeur permet de connaître le taux maximal de requête par seconde que l'API autorise.

4.7.2 Modifications

Tous les schémas sont concernés par cette modification, il faut donc les éditer pour faire propager les modifications faites dans le code.

1. **La colonne doit être ajoutée à la requête de création de la table apis présente dans le fichier `AdvancedPrefDatabase.java`.**
   — **Fichier concerné** : `AdvancedPrefDatabase.java`
   — **Emplacement** : Ligne 129
   — **Ancien code** :

```
db.execSQL(
"CREATE TABLE apis (id TEXT, name TEXT, url TEXT, readonlyurl TEXT,
↪  notesurl TEXT, user TEXT, pass TEXT, preset TEXT, showicon INTEGER
↪  DEFAULT 1, oauth INTEGER DEFAULT 0, accesstoken TEXT,
↪  accesstokensecret TEXT)");
```

   — **Nouveau code** :

```
db.execSQL(
"CREATE TABLE apis (id TEXT, name TEXT, url TEXT, readonlyurl TEXT,
↪  notesurl TEXT, user TEXT, pass TEXT, preset TEXT, showicon INTEGER
↪  DEFAULT 1, oauth INTEGER DEFAULT 0, accesstoken TEXT,
↪  accesstokensecret TEXT, rate_limit INTEGER DEFAULT NULL)");
```

   — Une valeur `NULL` par défaut permet d'enregistrer une API sans taux limite. Il est possible qu'une api ne dévoile pas son taux limite ou une API peut être utilisée pour une requête et donc ce taux limite ne sera jamais atteint.

2. **En plus de cette modification, la méthode `addAPI` du fichier `AdvancedPrefDatabase` doit également être adaptée afin de rajouter l'insertion de la donnée `rate_limit`. Une constante doit être définie au début du fichier pour le nom de la nouvelle colonne.**

— **Fichier concerné** : `AdvancedPrefDatabase.java`

— **Emplacement** : Ligne 450

— **Ancien code** :

```
/**
 * Adds a new API with the given values to the supplied database
 *
 * @param db a writeable SQLiteDatabase
 * @param id the internal id for this entry
 * @param name the name of the entry
 * @param url the read / write url
 * @param readonlyurl a read only url or null
 * @param notesurl a note url or null
 * @param user OSM display name
 * @param pass OSM password
 * @param auth authentication method
 */
private synchronized void addAPI(@NonNull SQLiteDatabase db, @NonNull
↪  String id, @NonNull String name, @NonNull String url, @Nullable
↪  String readonlyurl,
        @Nullable String notesurl, @Nullable String user, @Nullable
        ↪  String pass, @NonNull Auth auth) {
    ContentValues values = new ContentValues();
    values.put(ID_COL, id);
    values.put(NAME_COL, name);
    values.put(URL_COL, url);
    values.put(READONLYURL_COL, readonlyurl);
    values.put(NOTESURL_COL, notesurl);
    values.put(USER_COL, user);
    values.put(PASS_COL, pass);
    values.put(AUTH_COL, auth.ordinal());
    db.insert(APIS_TABLE, null, values);
}
```

— **Nouveau code** :

```
private synchronized void addAPI(@NonNull SQLiteDatabase db, @NonNull
↪  String id, @NonNull String name, @NonNull String url, @Nullable
↪  String readonlyurl, @Nullable String notesurl, @Nullable String
↪  user, @Nullable String pass, @NonNull Auth auth, @Nullable int
↪  rateLimit) {
```

```
    ContentValues values = new ContentValues();
    values.put(ID_COL, id);
    values.put(NAME_COL, name);
    values.put(URL_COL, url);
    values.put(READONLYURL_COL, readonlyurl);
    values.put(NOTESURL_COL, notesurl);
    values.put(USER_COL, user);
    values.put(PASS_COL, pass);
    values.put(AUTH_COL, auth.ordinal());
    values.put(RATE_LIMIT, rateLimit); // ajout ici
    db.insert(APIS_TABLE, null, values);
}
```

— Il faudra bien entendu modifier l'appel de cette méthode addAPI pour envoyer la valeur `rate_limit`. Une autre solution est de créer une méthode sans le paramètre rateLimit qui fera appel à la méthode ci-dessus avec null comme valeur pour l'argument rateLimit.

Une méthode de modification de la valeur `rate_limit` pour une API donnée peut également être créée au besoin.

3. **La méthode qui permet de sélectionner les API doit également être adaptée.**
   — **Fichier concerné** : `AdvancedPrefDatabase.java`
   — **Emplacement** : Ligne 503
   — **Ancien code** :

```
@NonNull
private synchronized API[] getAPIs(@NonNull SQLiteDatabase db, @Nullable
↪  String id) {
    Cursor dbresult = db.query(
            APIS_TABLE, new String[] { ID_COL, NAME_COL, URL_COL,
            ↪  READONLYURL_COL, NOTESURL_COL, USER_COL, PASS_COL,
            ↪  "preset", "showicon", AUTH_COL,
                    ACCESSTOKEN_COL, ACCESSTOKENSECRET_COL },
            id == null ? null : WHERE_ID, id == null ? null : new
            ↪  String[] { id }, null, null, null, null);
    API[] result = new API[dbresult.getCount()];
    dbresult.moveToFirst();
    for (int i = 0; i < result.length; i++) {
        Auth auth = Auth.BASIC;
```

```
        try {
            auth = API.Auth.values()[dbresult.getInt(9)];
        } catch (IndexOutOfBoundsException ex) {
            Log.e(DEBUG_TAG, "No auth method for " + dbresult.getInt(9));
        }
        result[i] = new API(dbresult.getString(0),
        ↪  dbresult.getString(1), dbresult.getString(2),
        ↪  dbresult.getString(3), dbresult.getString(4),
                dbresult.getString(5), dbresult.getString(6), auth,
                ↪  dbresult.getString(10), dbresult.getString(11));
        dbresult.moveToNext();
    }
    dbresult.close();
    return result;
}
```

— Il faut ajouter `RATE_LIMIT` dans le tableau de String à la ligne 505, 506.

— Il faut également ajouter la valeur récupérée par le `SELECT` dans le constructeur de l'objet `API` à la ligne 517, 528.

— Une fois ces modifications terminées, il faut ajouter un attribut à la classe `API` pour lier l'information à l'objet.

4.8 Scénario d'évolution n°8

4.8.1 Scénario

Suppression de la colonne description dans la table `presets`.

4.8.2 Modifications

Tous les schémas sont concernés par cette modification, il faut donc les éditer pour faire propager les modifications faites dans le code.

Cette modification entrainera la perte de donnée pour les presets. L'affichage devra également être adapté dans l'application.

Les modifications sont principalement centrées dans AdvancedPrefDatabase.

Tout d'abord, il faut supprimer la constante `DESCRIPTION_COL` à la ligne 74. Cette dernière correspond au nom de la colonne qu'on souhaite supprimer.

Grâce à cette constante, les IDE modernes aident grandement à trouver les endroits où la constante était utilisée. Dans le pire des cas, le compilateur prendra la relève et indiquera les endroits à modifier.

Voici toutefois les endroits importants à modifier.

1. Dans la méthode getActivePresets, il faut supprimer la constante `DESCRIPTION_COL`
   - **Fichier concerné** : `AdvancedPrefDatabase.java`
   - **Emplacement** : Ligne 662
   - **Ancien code** :

```
@NonNull
public PresetInfo[] getActivePresets() {
        SQLiteDatabase db = getReadableDatabase();
        Cursor dbresult = db.query(PRESETS_TABLE,
        new String[] { ID_COL, NAME_COL, VERSION_COL,
        ↪  SHORTDESCRIPTION_COL, DESCRIPTION_COL, URL_COL,
        ↪  LASTUPDATE_COL, ACTIVE_COL, USETRANSLATIONS_COL },
        "active=1", null, null, null, POSITION_COL);
        PresetInfo[] result = new PresetInfo[dbresult.getCount()];
        Log.d(DEBUG_TAG, "#prefs " + result.length);
        dbresult.moveToFirst();
        for (int i = 0; i < result.length; i++) {
                Log.d(DEBUG_TAG, "Reading pref " + i + " " +
                ↪  dbresult.getString(1));
                result[i] = new PresetInfo(dbresult.getString(0),
                ↪  dbresult.getString(1), dbresult.getString(2),
                ↪  dbresult.getString(3), dbresult.getString(4),
```

```
                dbresult.getString(5), dbresult.getString(6),
                ↪  dbresult.getInt(7) == 1, dbresult.getInt(8) == 1);
                dbresult.moveToNext();
            }
        dbresult.close();
        db.close();
        return result;
    }
```

— **Nouveau code** :

```
@NonNull
public PresetInfo[] getActivePresets() {
        SQLiteDatabase db = getReadableDatabase();
        Cursor dbresult = db.query(PRESETS_TABLE,
        new String[] { ID_COL, NAME_COL, VERSION_COL,
        ↪  SHORTDESCRIPTION_COL, URL_COL, LASTUPDATE_COL, ACTIVE_COL,
        ↪  USETRANSLATIONS_COL },
        "active=1", null, null, null, POSITION_COL);
        PresetInfo[] result = new PresetInfo[dbresult.getCount()];
        Log.d(DEBUG_TAG, "#prefs " + result.length);
        dbresult.moveToFirst();
        for (int i = 0; i < result.length; i++) {
                Log.d(DEBUG_TAG, "Reading pref " + i + " " +
                ↪  dbresult.getString(1));
                result[i] = new PresetInfo(dbresult.getString(0),
                ↪  dbresult.getString(1), dbresult.getString(2),
                ↪  dbresult.getString(3), dbresult.getString(4),
                dbresult.getString(5), dbresult.getString(6),
                ↪  dbresult.getInt(7) == 1, dbresult.getInt(8) == 1);
                dbresult.moveToNext();
            }
        dbresult.close();
        db.close();
        return result;
    }
```

2. Il faut également supprimer dbresult.getString(5) dans l'appel au constructeur de PresetInfo. De plus, il faut également décaler les index des autres appels à getString(val) car la supprimer d'une colonne sélectionnée décale tout

le reste.

3. La même modification que ci-dessus est à faire pour la méthode getPresets
4. La méthode setPresetAdditionalFields est à adapter car elle utilise également la description lors de la modification d'un preset. Pour ce faire, il faut supprimer les lignes 776 à 778. De ce fait, la colonne `description` ne sera plus utilisée par le programme pour interagir avec la base de données.
5. La dernière modification à effectuer est la suppression de l'attribut `description` de la classe PresetInfo.
6. Les modifications engendrées par la suppression de cet attribut ne sont pas expliquées ici mais le compilateur permet de repérer les endroits où l'attribut `description` était utilisé. Il est donc aisé d'effectuer les derniers changements.

4.9 Scénario d'évolution n°9

4.9.1 Scénario

Ajouter une clef étrangère entre la colonne `photo.dir` et `directories.dir`

4.9.2 Modifications

Tous les schémas sont concernés par cette modification, il faut donc les éditer pour faire propager les modifications faites dans le code.

Etant donné qu'une nouvelle contrainte référentielle va être ajoutée, certaines lignes déjà existantes pourraient poser problème. Pour qu'une contrainte de clef étrangère puisse exister, il est impératif que la colonne référencée ne contiennent que des valeurs uniques et qu'elles soient de même type. Si besoin, certaines entrées de la table directories devront être supprimées pour respecter la contrainte d'unicité.

Pour l'instant, cette contrainte n'est pas totalement existante. En effet, il existe déjà une relation entre ces 2 colonnes mais basée sur un `LIKE`. Il n'y a donc aucune contrainte explicitement définie.

1. Adapter la requête de création de la table
   - **Fichier concerné** : `PhotoIndex.java`
   - **Emplacement** : ligne 88
   - **Méthode** : `public synchronized void onCreate(SQLiteDatabase db)`

   Pour créer une contrainte, il faut tout d'abord adapter la requête de création de la table en y ajoutant la contraite de référence. Les 2 colonnes ayant le même type, il n'y a pas d'autres modifications à apporter afin de pouvoir créer la clef étrangère.

```
CREATE TABLE IF NOT EXISTS photos (lat int, lon int, direction int DEFAULT
↪  NULL, dir VARCHAR, name VARCHAR, source VARCHAR DEFAULT NULL);
```

   doit être changée en y ajoutant la contrainte référentielle comme suit :

```
CREATE TABLE IF NOT EXISTS photos (lat int, lon int, direction int DEFAULT
↪  NULL, dir VARCHAR, name VARCHAR, source VARCHAR DEFAULT NULL) FOREIGN
↪  KEY(dir) REFERENCES directories(dir);
```

2. Modifier la requête de sélection des photos liée aux directories.
   — **Fichier concerné** : `PhotoIndex.java`
   — **Emplacement** : ligne 236
   — **Méthode** : `private void indexDirectories()`

```
private void indexDirectories() {
    Log.d(DEBUG_TAG, "scanning directories");
    // determine at least a few of the possible mount points
    File sdcard = Environment.getExternalStorageDirectory(); // NOSONAR
    List<String> mountPoints = new ArrayList<>();
    mountPoints.add(sdcard.getAbsolutePath());
    mountPoints.add(sdcard.getAbsolutePath() +
    ↪  Paths.DIRECTORY_PATH_EXTERNAL_SD_CARD);
    mountPoints.add(Environment.getExternalStoragePublicDirectory(Environme
    ↪  nt.DIRECTORY_DOWNLOADS).getAbsolutePath()); //
    ↪  NOSONAR
    File storageDir = new File(Paths.DIRECTORY_PATH_STORAGE);
    File[] list = storageDir.listFiles();
    if (list != null) {
        for (File f : list) {
            if (f.exists() && f.isDirectory() &&
            ↪  !sdcard.getAbsolutePath().equals(f.getAbsolutePath())) {
                Log.d(DEBUG_TAG, "Adding mount point " +
                ↪  f.getAbsolutePath());
                mountPoints.add(f.getAbsolutePath());
            }
        }
    }

    SQLiteDatabase db = null;
    Cursor dbresult = null;

    try {
        db = getWritableDatabase();
        dbresult = db.query(SOURCES_TABLE, new String[] { URI_COLUMN,
        ↪  LAST_SCAN_COLUMN, TAG_COLUMN }, TAG_COLUMN + " is NULL", null,
        ↪  null, null, null,
                null);
        int dirCount = dbresult.getCount();
        dbresult.moveToFirst();
        // loop over the directories configured
        for (int i = 0; i < dirCount; i++) {
```

```
            String dir = dbresult.getString(0);
            long lastScan = dbresult.getLong(1);
            Log.d(DEBUG_TAG, dbresult.getString(0) + " " +
            ↪  dbresult.getLong(1));
            // loop over all possible mount points
            for (String m : mountPoints) {
                File indir = new File(m, dir);
                Log.d(DEBUG_TAG, "Scanning directory " +
                ↪  indir.getAbsolutePath());
                if (indir.exists()) {
                    Cursor dbresult2 = null;
                    try {
                        dbresult2 = db.query(PHOTOS_TABLE, new String[] {
                        ↪  "distinct dir" }, "dir LIKE '" +
                        ↪  indir.getAbsolutePath() + "%'", null, null,
                        ↪  null,
                                null, null);
                        int dirCount2 = dbresult2.getCount();
                        dbresult2.moveToFirst();
                        for (int j = 0; j < dirCount2; j++) {
                            String photo = dbresult2.getString(0);
                            File pDir = new File(photo);
                            if (!pDir.exists()) {
                                Log.d(DEBUG_TAG, "Deleting entries for gone
                                ↪  photo " + photo);
                                db.delete(PHOTOS_TABLE, URI_WHERE, new
                                ↪  String[] { photo });
                            }
                            dbresult2.moveToNext();
                        }
                        dbresult2.close();
                        scanDir(db, indir.getAbsolutePath(), lastScan);
                        updateSources(db, indir.getName(), null,
                        ↪  System.currentTimeMillis());
                    } finally {
                        close(dbresult2);
                    }
                    continue;
                }
                Log.d(DEBUG_TAG, "Directory " + indir.getAbsolutePath() + "
                ↪  doesn't exist");
                // remove all entries for this directory
```

```
                db.delete(PHOTOS_TABLE, URI_WHERE, new String[] {
                ↪  indir.getAbsolutePath() });
                db.delete(PHOTOS_TABLE, "dir LIKE ?", new String[] {
                ↪  indir.getAbsolutePath() + "/%" });
            }
            dbresult.moveToNext();
        }
    } catch (SQLiteException ex) {
        // Don't crash just report
        ACRAHelper.nocrashReport(ex, ex.getMessage());
    } finally {
        close(dbresult);
        SavingHelper.close(db);
    }
}
```

Etant donné qu'une contrainte référentielle existe, il n'est plus nécessaire l'utiliser le mot clef `LIKE` pour récupérer les photos qui appartient à un dossier mais un simple select suffit.

3. Modifier la méthode d'insertion

— **Fichier concerné** : `PhotoIndex.java`

— **Emplacement** : ligne 515

— **Méthode** : `private void insertPhoto(@NonNull SQLiteDatabase db, @NonNull Photo photo, @N`

```
private void insertPhoto(@NonNull SQLiteDatabase db, @NonNull Photo photo,
↪  @NonNull String name, @Nullable String source) {
    try {
        ContentValues values = new ContentValues();
        values.put(LAT_COLUMN, photo.getLat());
        values.put(LON_COLUMN, photo.getLon());
        if (photo.hasDirection()) {
            values.put(DIRECTION_COLUMN, photo.getDirection());
        }
        values.put(URI_COLUMN, photo.getRef());
        values.put(NAME_COLUMN, name);
        if (source ≠ null) {
            values.put(SOURCE_COLUMN, source);
        }
        db.insert(PHOTOS_TABLE, null, values);
    } catch (SQLiteException sqex) {
```

```
            Log.d(DEBUG_TAG, sqex.toString());
            ACRAHelper.nocrashReport(sqex, sqex.getMessage());
        } catch (NumberFormatException bfex) {
            // ignore silently, broken pictures are not our business
        } catch (Exception ex) {
            Log.d(DEBUG_TAG, ex.toString());
            ACRAHelper.nocrashReport(ex, ex.getMessage());
        } // ignore
}
```

Cette dernière s'occupe d'insérer une photo. Dorénavent, il faut s'assurer que la propriété `dir` de la photo qu'on veut insérer existe dans la table `directories`. Une vérification avec un select permet de résoudre ce problème, sinon il est possible d'utiliser l'exception générée par la requete d'insertion.

L'adaptation du code pourra donc se faire :

— dans la méthode ci-dessus, sans rien adapter. La méthode comporte déjà un `try/catch` qui empéchera à l'application de planter. Il serait toutefois intéressant de renvoyer l'information pour déterminer si l'insertion a eu lieu ou pas.

— dans la méthode ci-dessus, en y ajoutant un sélect afin de déterminer que le `java` existe bien dans la table `directories`

— dans la méthode ci-dessus, en renvoyant une exception qui informe la méthode appellante qu'une erreur s'est produite. Il faudra également modifier l'appel à la méthode `insertPhoto` dans les autres classes qui l'utilise pour l'entourer d'un `try/catch` et ainsi éviter des crashs.

4.10 Scénario d'évolution n°10

4.10.1 Scénario

Ajouter une clef primaire composite sur la table `filterentries` pour les colonnes `filter`, `type` et `value`.

4.10.2 Modifications

1. **Ajout d'une clé primaire lors de la création de la table `filterentries` dans la méthode `onCreate` du fichier `TagFilterDatabaseHelper.java`**
   - **Fichier concerné** : `TagFilterDatabaseHelper.java`
   - **Emplacement** : Ligne 32
   - **Ancien code** :

```
@Override
public void onCreate(SQLiteDatabase db) {
    try {
        db.execSQL("CREATE TABLE filters (name TEXT)");
        db.execSQL("INSERT INTO filters VALUES ('Default')");
        db.execSQL(
                "CREATE TABLE filterentries (filter TEXT, include INTEGER
                ↪  DEFAULT 0, type TEXT DEFAULT '*', key TEXT DEFAULT
                ↪  '*', value TEXT DEFAULT '*', active INTEGER DEFAULT
                ↪  0, FOREIGN KEY(filter) REFERENCES filters(name))");
    } catch (SQLException e) {
        Log.w(DEBUG_TAG, "Problem creating database", e);
    }
}
```

   - **Nouveau code** :

```
@Override
public void onCreate(SQLiteDatabase db) {
    try {
        db.execSQL("CREATE TABLE filters (name TEXT)");
        db.execSQL("INSERT INTO filters VALUES ('Default')");
        db.execSQL(
            "CREATE TABLE filterentries " +
            "(filter TEXT, " +
            "include INTEGER DEFAULT 0, " +
```

```
            "type TEXT DEFAULT '*', " +
            "key TEXT DEFAULT '*', " +
            "value TEXT DEFAULT '*', " +
            "active INTEGER DEFAULT 0, " +
            "FOREIGN KEY(filter) REFERENCES filters(name), " +
            "PRIMARY KEY(filter, type, value))"
        );
    } catch (SQLException e) {
        Log.w(DEBUG_TAG, "Problem creating database", e);
    }
}
```

2. **Comme on utilise une clef primaire, la colonne implicite rowid n'est plus crée par sqlite donc on doit utiliser la clé primaire composite**
   - **Fichier concerné** : `TagFilterDatabaseHelper.java`
   - **Emplacement** : Ligne 62
   - **Ancien code** :

```
private static final String QUERY = "SELECT rowid as _id, active,
↪  include, type, key, value FROM filterentries WHERE filter = '";
```

   - **Nouveau code** :

```
private static final String QUERY = "SELECT filter, type, value, active,
↪  include, key FROM filterentries WHERE filter = '";
```

3. **Modification accès à la base de données pour les requêtes de mise à jour**
   - **Fichier concerné** : `TagFilterActivity.java`
   - **Emplacement** : Ligne 206
   - **Ancien code**

```
db.update(FILTERENTRIES_TABLE, values, "rowid=" + id, null);
```

   - **Nouveau code**

```
db.update(FILTERENTRIES_TABLE, values, "filter=? AND type=? AND
↪  value=?", new String[] { filter, type, value });
```

4. **Modification accès à la base de données pour les requêtes de suppression**
   — **Fichier concerné** : `TagFilterActivity.java`
   — **Emplacement** : Ligne 370
   — **Ancien code**

```
db.delete(FILTERENTRIES_TABLE, "rowid=" + id, null);
```

   — **Nouveau code**

```
db.delete(FILTERENTRIES_TABLE, "filter=? AND type=? AND value=?", new
↪  String[] { filter, type, value });
```

5. **Modification**
   — **Fichier concerné** : `TagFilterActivity.java`
   — **Emplacement** : Ligne 333
   — **Ancien code**

```
final int id = cursor.getInt(cursor.getColumnIndexOrThrow("_id"));
vh.id = id;
```

   — **Nouveau code**

```
final String filter =
↪  cursor.getString(cursor.getColumnIndexOrThrow(FILTER_COLUMN));
final String type =
↪  cursor.getString(cursor.getColumnIndexOrThrow(TYPE_COLUMN));
final String value =
↪  cursor.getString(cursor.getColumnIndexOrThrow(VALUE_COLUMN));
vh.id = filter + ":" + type + ":" + value;
```

6. **Modification**
   — **Fichier concerné** : `TagFilterActivity.java`
   — **Emplacement** : Ligne 396

— **Ancien code**

```
private void newCursor() {
    Cursor newCursor = db.rawQuery(QUERY + filter + "'", null);
    Cursor oldCursor = filterAdapter.swapCursor(newCursor);
    oldCursor.close();
}
```

— **Nouveau code**

```
private void newCursor() {
    Cursor newCursor = db.rawQuery("SELECT filter, type, value, active,
    ↪  include, key FROM filterentries WHERE filter=?", new String[] {
    ↪  filter });
    Cursor oldCursor = filterAdapter.swapCursor(newCursor);
    oldCursor.close();
}
```

5 Partie 4

5.1 Schéma de la base de données

Mérites

— **Nommage explicite :** Les tables et les colonnes ont des noms descriptifs, donc on sait directement de quoi il s'agit.

— **Utilisation de clés étrangères :** Certaines relations entre les tables utilisent des clés étrangères, assurant l'intégrité référentielle (par exemple, entre les tables `filters` et `filterentries`).

— **Segmentation claire des données :** Les tables sont divisées correctement en fonction de leurs fonctionnalités (filtres, photos, headers, etc.).

Inconvénients

1. **Tables et colonnes non utilisées :**
   — La table `t_renderer` et sa colonne `id` ne sont pas utilisées.
2. **Colonnes de clés étrangères nullables :**
   — Plusieurs champs de clés étrangères autorisent les valeurs nulles, ce qui peut entraîner des problèmes d'intégrité des données.
3. **Relations implicites :**
   — Certaines relations sont implicites, comme le lien entre `photos` et `directories` via une requête `LIKE` plutôt qu'une clé étrangère définie.
4. **Données dénormalisées :**
   — Des tables surchargées comme `photos` stockent à la fois des métadonnées et des informations géospatiales, réduisant la clarté et la normalisation.
5. **Conception des clés primaires :**
   — La clé primaire pour `tiles` est composite, ce qui peut compliquer l'indexation et les requêtes.

Recommandations

— Supprimer les tables et colonnes inutilisées, comme `t_renderer`, pour réduire la complexité du schéma et améliorer la maintenabilité.

— Définir des clés étrangères explicites pour les relations implicites, comme `photos.dir` et `directories.dir`, afin d'assurer la cohérence des données.

— Normaliser les données en scindant les tables surchargées (par exemple, diviser `photos` en `photo_metadata` et `photo_location`).
— S'assurer que toutes les clés étrangères font référence à des champs uniques et non nuls pour éviter les problèmes d'intégrité.
— Réévaluer l'utilisation des clés primaires composites et envisager d'introduire des clés substitutives là où c'est pertinent.

5.2 Code de manipulation de la base de données

Avantages

— **Gestion centralisée des requêtes SQL :** Les requêtes sont bien regroupées dans des fichiers Java spécifiques, facilitant la navigation et les mises à jour du code.
— **Requêtes lisibles :** La plupart des requêtes SQL sont simples et directes, facilitant leur compréhension.

Inconvénients

1. **Requêtes codées en dur :**
   — Les requêtes sont souvent codées en dur sous forme de chaînes, ce qui peut poser des défis de maintenance et des risques d'injection SQL.
2. **Utilisation limitée des requêtes paramétrées :**
   — Certaines requêtes utilisent des chaînes concaténées au lieu de la paramétrisation, augmentant les risques pour la sécurité.
3. **Absence de tests unitaires pour les requêtes :**
   — Le manque de tests automatisés pour les instructions SQL peut entraîner des bugs non détectés pendant le développement.
4. **Évolution manuelle du schéma :**
   — Les mises à jour du schéma nécessitent des modifications manuelles des requêtes, augmentant les risques d'erreurs pendant l'évolution du schéma.

Recommandations

— Passer à des requêtes paramétrées ou à un framework ORM (par exemple, `Room` pour Android) pour améliorer la sécurité et la maintenabilité.
— Développer une suite de tests complète pour les opérations sur la base de données afin d'assurer leur exactitude.

— Introduire des scripts de migration de schéma ou des bibliothèques comme `Flyway` ou le système de migration intégré de `Room` pour gérer les changements de schéma de manière systématique.
— Consolider la logique des requêtes SQL en utilisant des classes ou méthodes utilitaires pour réduire la duplication et centraliser les modifications.

6 Conclusion

Ce projet a permis d'approfondir les concepts de rétro-ingénierie appliqués à une base de données d'application Android, en l'occurrence Vespucci. En retraçant les schémas physique, logique et conceptuel, nous avons mis en lumière la structure sous-jacente de la base de données et identifié des opportunités d'amélioration.

L'étude des clés étrangères a révélé leur rôle central dans l'intégrité référentielle des données. Si certaines contraintes étaient implicites, notamment dans les relations entre les tables, le processus de rétro-ingénierie a permis de proposer des améliorations, telles que l'ajout explicite de clés étrangères. Ces ajustements ont renforcé la cohérence des données et simplifié les requêtes SQL nécessaires pour interagir avec elles.

En conclusion, cette analyse a mis en évidence l'importance des bonnes pratiques en conception de bases de données, notamment dans l'utilisation des clés étrangères pour établir des relations solides et explicites entre les entités. De plus, ce projet a souligné le rôle crucial de la documentation dans la compréhension et l'évolution des bases de données. Une documentation claire et détaillée facilite non seulement la rétro-ingénierie, mais aussi la maintenance et l'adaptabilité des systèmes face aux besoins changeants. Elle constitue un pilier fondamental pour assurer la pérennité et la cohérence des bases de données dans des environnements complexes et collaboratifs.