Etape 3: scénarios d'évolution

1- Enlever Tags de la table Stories puis créer la table story_tags

a- Remove story_tags from story table
b- Create Stroy_tags table
c- c - Add story_hash to Story tags table
d- Add tags to Story tags table

Ainsi, chaque Story est lié à un ou plusieurs Tag(s) via la clé étrangère story_hash présente dans story_tags. Cette approche offre plusieurs avantages :

Maintenabilité : Les mises à jour du tag d'une story se font de manière centralisée dans la table "story_tags", simplifiant la maintenance.

Extensibilité : Si un story peut avoir plusieurs tags (par exemple tag1,tag2, etc), cette structure permet d'ajouter facilement des relations supplémentaires.

L'utilisation d'une table distincte pour représenter les tags contribue à une meilleure organisation des données, surtout lorsque plusieurs entités peuvent partager la même tags.

Impact dans le code

```
246
247    static final String STORY_SQL = "CREATE TABLE " + STORY_TABLE + " (" +
248        STORY_HASH + TEXT + " PRIMARY KEY, " +
249        STORY_AUTHORS + TEXT + ", " +
250        STORY_CONTENT + TEXT + ", " +
251        STORY_SHORT_CONTENT + TEXT + ", " +
252        STORY_TIMESTAMP + INTEGER + ", " +
253        STORY_SHARED_DATE + INTEGER + ", " +
254        STORY_FEED_ID + INTEGER + ", " +
255        STORY_ID + TEXT + ", " +
256        STORY_INTELLIGENCE_AUTHORS + INTEGER + ", " +
257        STORY_INTELLIGENCE_FEED + INTEGER + ", " +
```

```
100    String STORY_SOCIAL_USER_ID =  socialUserId ;
101    String STORY_SOURCE_USER_ID = "sourceUserId";
102    String STORY_TAGS = "tags";
103    String STORY_USER_TAGS = "user_tags";
```

```
342
343    private static final String[] BASE_STORY_COLUMNS = {
344        STORY_AUTHORS, STORY_SHORT_CONTENT, STORY_TIMESTAMP, STORY_SHARED_DATE,
345        STORY_TABLE + "." + STORY_FEED_ID, STORY_TABLE + "." + STORY_ID,
346        STORY_INTELLIGENCE_AUTHORS, STORY_INTELLIGENCE_FEED, STORY_INTELLIGENCE_TAGS, STORY_INTELLIGENCE_
347        STORY_INTELLIGENCE_TITLE, STORY_PERMALINK, STORY_READ, STORY_STARRED, STORY_STARRED_DATE,
348        STORY_TAGS, STORY_USER_TAGS, STORY_TITLE,
349        STORY_SOCIAL_USER_ID, STORY_SOURCE_USER_ID, STORY_SHARED_USER_IDS, STORY_FRIEND_USER_IDS, STORY_HA
350        STORY_LAST_READ_DATE, STORY_THUMBNAIL_URL, STORY_HAS_MODIFICATIONS,
351    };
```

Ainsi que toutes les lignes où BASE_STORY_COLUMNS est utilisé.

```
16      @Override
17      public void onCreate(SQLiteDatabase db) {
18          db.execSQL(DatabaseConstants.FEED_SQL);
19          db.execSQL(DatabaseConstants.SOCIAL_FEED_SQL);
20          db.execSQL(DatabaseConstants.FOLDER_SQL);
21          db.execSQL(DatabaseConstants.USER_SQL);
22          db.execSQL(DatabaseConstants.STORY_SQL);
23          db.execSQL(DatabaseConstants.READING_SESSION_SQL);
24          db.execSQL(DatabaseConstants.STORY_TEXT_SQL);
25          db.execSQL(DatabaseConstants.COMMENT_SQL);
26          db.execSQL(DatabaseConstants.REPLY_SQL);
27          db.execSQL(DatabaseConstants.CLASSIFIER_SQL);
28          db.execSQL(DatabaseConstants.SOCIALFEED_STORIES_SQL);
29          db.execSQL(DatabaseConstants.STARREDCOUNTS_SQL);
30          db.execSQL(DatabaseConstants.SAVED_SEARCH_SQL);
31          db.execSQL(DatabaseConstants.ACTION_SQL);
32          db.execSQL(DatabaseConstants.NOTIFY_DISMISS_SQL);
33          db.execSQL(DatabaseConstants.FEED_TAGS_SQL);
34          db.execSQL(DatabaseConstants.FEED_AUTHORS_SQL);
35          db.execSQL(DatabaseConstants.SYNC_METADATA_SQL);
36      }
```

2 - Enlever Les attributs STORY_INTELLIGENCE_* de la table Stories puis creer la table STORY_INTELLIGENCE

a- Remove story_intelligence_* from story
b- Create story_intelligence table
c- add story_intelligence_*into story_intelligence table

STORY_INTELLIGENCE_AUTHORS, STORY_INTELLIGENCE_FEED

STORY_INTELLIGENCE_TAGS, STORY_INTELLIGENCE_TITLE

STORY_INTELLIGENCE_TOTAL

Ainsi, chaque Story est lié à attributs story_intelligence via la clé étrangère story_hash present dans story_intelligence. Cette approche offre plusieurs avantages :

Maintenabilité : Les mises à jour des attributs story_intelligence d'une story se font de manière centralisée dans la table "story_intelligence", simplifiant la maintenance.

L'utilisation d'une table distincte pour représenter les story_intelligence contribue à une meilleure organisation des données, surtout lorsque plusieurs entités peuvent partager la même tags.

Impact dans le code

```
246
247     static final String STORY_SQL = "CREATE TABLE " + STORY_TABLE + " (" +
248         STORY_HASH + TEXT + " PRIMARY KEY, " +
249         STORY_AUTHORS + TEXT + ", " +
250         STORY_CONTENT + TEXT + ", " +
251         STORY_SHORT_CONTENT + TEXT + ", " +
252         STORY_TIMESTAMP + INTEGER + ", " +
253         STORY_SHARED_DATE + INTEGER + ", " +
254         STORY_FEED_ID + INTEGER + ", " +
255         STORY_ID + TEXT + ", " +
256         STORY_INTELLIGENCE_AUTHORS + INTEGER + ", " +
257         STORY_INTELLIGENCE_FEED + INTEGER + ", " +
258         STORY_INTELLIGENCE_TAGS + INTEGER + ", " +
259         STORY_INTELLIGENCE_TITLE + INTEGER + ", " +
260         STORY_INTELLIGENCE_TOTAL + INTEGER + ", " +
```

```
43     private static final String[] BASE_STORY_COLUMNS = {
44         STORY_AUTHORS, STORY_SHORT_CONTENT, STORY_TIMESTAMP, STORY_SHARED_DATE,
45         STORY_TABLE + "." + STORY_FEED_ID, STORY_TABLE + "." + STORY_ID,
46         STORY_INTELLIGENCE_AUTHORS, STORY_INTELLIGENCE_FEED, STORY_INTELLIGENCE_TAGS, STORY_INTELLIGENCE_TOTAL,
47         STORY_INTELLIGENCE_TITLE, STORY_PERMALINK, STORY_READ, STORY_STARRED, STORY_STARRED_DATE,
48         STORY_TAGS, STORY_USER_TAGS, STORY_TITLE,
49         STORY_SOCIAL_USER_ID, STORY_SOURCE_USER_ID, STORY_SHARED_USER_IDS, STORY_FRIEND_USER_IDS, STORY_HASH,
50         STORY_LAST_READ_DATE, STORY_THUMBNAIL_URL, STORY_HAS_MODIFICATIONS,
51     };
52
53     public static String NOTIFY_FOCUS_STORY_QUERY =
54         STORY_QUERY_BASE_1 +
55         STORY_FEED_ID + " IN (SELECT " + FEED_ID + " FROM " + FEED_TABLE + " WHERE " + FEED_NOTIFICATION_FI
56         " AND " + STORY_INTELLIGENCE_TOTAL + " > 0 " +
57         STORY_QUERY_BASE_2 +
58         " ORDER BY " + STORY_TIMESTAMP + " DESC";
59
60     public static String NOTIFY_UNREAD_STORY_QUERY =
61         STORY_QUERY_BASE_1 +
62         STORY_FEED_ID + " IN (SELECT " + FEED_ID + " FROM " + FEED_TABLE + " WHERE " + FEED_NOTIFICATION_FI
63         " AND " + STORY_INTELLIGENCE_TOTAL + " >= 0 " +
64         STORY_QUERY_BASE_2 +
65         " ORDER BY " + STORY_TIMESTAMP + " DESC";
```

3- DÉCOMPOSER SAVE_SEARCH_ADDRESS EN PLUSIEURS ATTRIBUTS(RUE, VILLE, PAYS, CODE POSTAL)

a- Delete saved_search_address from save_search
b- add saved_search_city into saved_search
c- add saved_search_postcode into saved_search
d- add saved_search_street into saved_search
e- add saved_search_country into saved_search

Chaque enregistrement dans cette table représenterait un save_search avec toutes les informations nécessaires, et l'adresse serait décomposée en plusieurs colonnes.

Cette approche facilite la recherche, la mise à jour et la gestion des données d'adresse. Elle suit également les principes de la normalisation des bases de données en évitant la redondance d'informations. On pourrait facilement faire une recherche en fonction d' un pays, d' une ville, d'un code postal ou alors d'une rue.

Impact dans le code

```
static final String SAVED_SEARCH_SQL = "CREATE TABLE " + SAVED_SEARCH_TABLE + " (" +
        SAVED_SEARCH_FEED_TITLE + TEXT + ", " +
        SAVED_SEARCH_FAVICON + TEXT + ", " +
        SAVED_SEARCH_ADDRESS + TEXT + ", " +
        SAVED_SEARCH_QUERY + TEXT + ", " +
        SAVED_SEARCH_FEED_ID +
        ")";
```

```
153     public static final String SAVED_SEARCH_FEED_TITLE = "saved_search_title";
154     public static final String SAVED_SEARCH_FAVICON = "saved_search_favicon";
155     public static final String SAVED_SEARCH_ADDRESS = "saved_search_address";
156     public static final String SAVED_SEARCH_QUERY = "saved_search_query";
157     public static final String SAVED_SEARCH_FEED_ID = "saved_search_feed_id";
158
```

```
public Cursor getSavedSearchCursor(CancellationSignal cancellationSignal) {
    return query(false, DatabaseConstants.SAVED_SEARCH_TABLE, null, null, null, null,  null, null, null,
}
```

```
16      @Override
17      public void onCreate(SQLiteDatabase db) {
18          db.execSQL(DatabaseConstants.FEED_SQL);
19          db.execSQL(DatabaseConstants.SOCIAL_FEED_SQL);
20          db.execSQL(DatabaseConstants.FOLDER_SQL);
21          db.execSQL(DatabaseConstants.USER_SQL);
22          db.execSQL(DatabaseConstants.STORY_SQL);
23          db.execSQL(DatabaseConstants.READING_SESSION_SQL);
24          db.execSQL(DatabaseConstants.STORY_TEXT_SQL);
25          db.execSQL(DatabaseConstants.COMMENT_SQL);
26          db.execSQL(DatabaseConstants.REPLY_SQL);
27          db.execSQL(DatabaseConstants.CLASSIFIER_SQL);
28          db.execSQL(DatabaseConstants.SOCIALFEED_STORIES_SQL);
29          db.execSQL(DatabaseConstants.STARREDCOUNTS_SQL);
30          db.execSQL(DatabaseConstants.SAVED_SEARCH_SQL);
31          db.execSQL(DatabaseConstants.ACTION_SQL);
32          db.execSQL(DatabaseConstants.NOTIFY_DISMISS_SQL);
33          db.execSQL(DatabaseConstants.FEED_TAGS_SQL);
34          db.execSQL(DatabaseConstants.FEED_AUTHORS_SQL);
35          db.execSQL(DatabaseConstants.SYNC_METADATA_SQL);
36      }
```

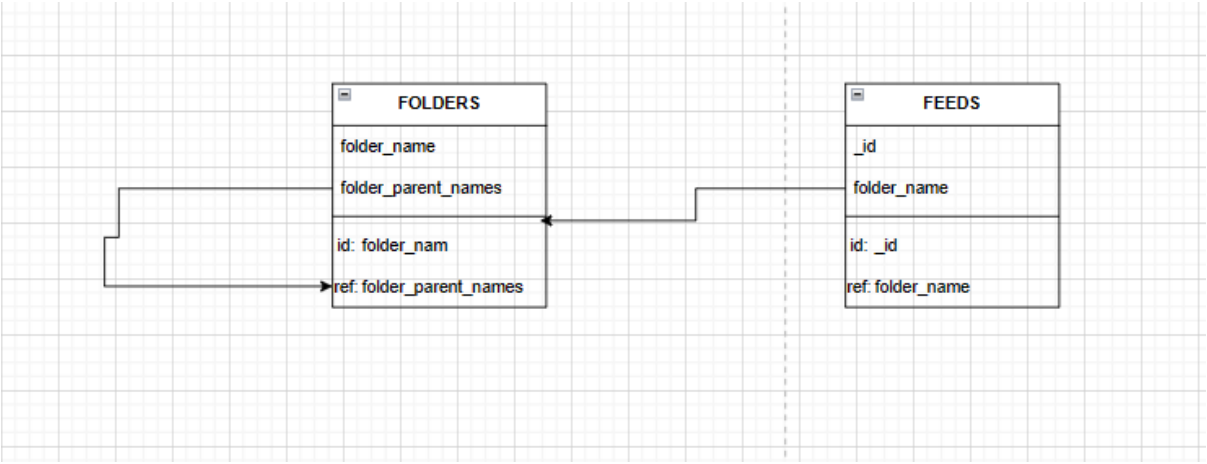
4- MODIFICATION DES TABLES FOLDER ET FEEDS:

a- Delete folder_children_names from Folders
b- Delete folder_feed_ids from Folders
a- c- add folder_name into Feeds table

Lorsqu'on se fie à l'interface de l'application, on se rend compte que lorsqu'on crée un Feed on l'associe à un folder ceci nous permet de comprendre que la table folder peut contenir des foder enfants ainsi qu'une liste de Feeds. Lui-même pouvant être contenu dans un folder parent

Ainsi on modifierait le code de telle sorte que la table folder aurait les attributs folder_name(primary key) et Parent_folder_name(foreign). Puis la table Feeds aurait un attribut folder_name faisant référence ici au folder dans lequel il est situé, ainsi pour avoir les parents d' un folder il faudrait utiliser

un code prend récursivement les parents de folder_parent_names. puis la liste de tous les feeds ayant pour folder_names permettrait d'avoir tous les feeds présents dans un folder.



Impact dans le Code

```
    static final String FOLDER_SQL = "CREATE TABLE " + FOLDER_TABLE + " (" +
        FOLDER_NAME + TEXT + " PRIMARY KEY, " +
        FOLDER_PARENT_NAMES + TEXT + ", " +
        FOLDER_CHILDREN_NAMES + TEXT + ", " +
        FOLDER_FEED_IDS + TEXT +
        ")";
    ) ;
```

```
static final String FEED_SQL = "CREATE TABLE " + FEED_TABLE + " (" +
    FEED_ID + INTEGER + " PRIMARY KEY, " +
    FEED_ACTIVE + TEXT + ", " +
    FEED_ADDRESS + TEXT + ", " +
    FEED_FAVICON_COLOR + TEXT + ", " +
    FEED_FAVICON_URL + TEXT + ", " +
    FEED_POSITIVE_COUNT + INTEGER + ", " +
    FEED_NEGATIVE_COUNT + INTEGER + ", " +
    FEED_NEUTRAL_COUNT + INTEGER + ", " +
    FEED_FAVICON_FADE + TEXT + ", " +
    FEED_FAVICON_TEXT + TEXT + ", " +
    FEED_FAVICON_BORDER + TEXT + "  " +
```

```
public static final String FOLDER_TABLE = "folders";
public static final String FOLDER_NAME = "folder_name";
public static final String FOLDER_PARENT_NAMES = "folder_parent_names";
public static final String FOLDER_CHILDREN_NAMES = "folder_children_names";
public static final String FOLDER_FEED_IDS = "folder_feedids";
```

```
public void updateFeed(Feed feed) {
    synchronized (RW_MUTEX) {
        dbRW.insertWithOnConflict(DatabaseConstants.FEED_TABLE, null, feed.getValues(), SQLiteDatabase
    }
}
```

```
                for (Feed feed : apiResponse.feeds) {
                    feedValues.add(feed.getValues());
                }
                bulkInsertValuesExtSync(DatabaseConstants.FEED_TABLE, feedValues);
            }

    public Folder getFolder(String folderName) {
        String[] selArgs = new String[] {folderName};
        String selection = DatabaseConstants.FOLDER_NAME + " = ?";
        Cursor c = dbRO.query(DatabaseConstants.FOLDER_TABLE, null, selection, selA
        if (c.getCount() < 1) {
            closeQuietly(c);
            return null;
        }
        Folder folder = Folder.fromCursor(c);
    }

    public Cursor getFoldersCursor(CancellationSignal cancellationSignal) {
        return query(false, DatabaseConstants.FOLDER_TABLE, null, null, null, null, null, null, null, cancel
    }

    public Cursor getFeedsCursor(CancellationSignal cancellationSignal) {
        return query(false, DatabaseConstants.FEED_TABLE, null, null, null, null, null, "UPPER(" + DatabaseC
    }
```

5- SUPPRIMER L'ATTRIBUT SUBSCRIBERS DE FEEDS PUIS CREER UNE TABLE FEED_SUBSCRIBERS

a- Remove subscribers from Feeds table
b- create Feed_suscribers table
c- Add Feed_id column to Feed_suscribers table
d- Add suscribers column to Feed_suscribers table

Ainsi, chaque Feed est lié à un ou plusieurs subscriber(s) via la clé étrangère feed_id présentée dans Feed_suscribers. Cette approche offre plusieurs avantages :

Maintenabilité : Les mises à jour du suscriber d'un feed se font de manière centralisée dans la table "Feed_suscribers", simplifiant la maintenance.

Extensibilité : Si un feeds peut avoir plusieurs suscribers (par exemple subs1,subs2, etc), cette structure permet d'ajouter facilement des relations supplémentaires.

L'utilisation d'une table distincte pour représenter les subscribers contribue à une meilleure organisation des données, surtout lorsque plusieurs entités peuvent partager la même subscriber.

Impact dans le code

on aura besoin de creer des constantes pour chaque colonne de la nouvelle table Feed_suscribers ainsi qu'une contante qui permet de representer la requete permettant de creer une table Feed_subscribers

Puis toutes les lignes de codes permettant de recuper toutes les informations liees a un feed

```
    public static final String FEED_SUBSCRIBERS = "subscribers";
    public static final String FEED_OPENS = "opens";
```

```
    static final String FEED_SQL = "CREATE TABLE " + FEED_TABLE + " (" +
        FEED_ID + INTEGER + " PRIMARY KEY, " +
        FEED_ACTIVE + TEXT + ", " +
        FEED_ADDRESS + TEXT + ", " +
```

```
public void updateFeed(Feed feed) {
    synchronized (RW_MUTEX) {
        dbRW.insertWithOnConflict(DatabaseConstants.FEED_TABLE, null, feed.getValues(), SQLiteDatabase.CON
    }
}
```

```
79      // not vended by API, but used locally for UI
80      public boolean fetchPending;
81
82      public ContentValues getValues() {
83          ContentValues values = new ContentValues();
```

6- DECOMPOSER FEED_ADDRESS EN PLUSIEURS ATTRIBUTS (RUE, VILLE, PAYS, CODE POSTAL)

a- Delete feed_address from feeds
b- add feed_city into feeds
c- add feed_postcode into feeds
d- add feed_street into feeds
e- add feed_country into feeds

Chaque enregistrement dans cette table représenterait un feed avec toutes les informations nécessaires, et l'adresse serait décomposée en plusieurs colonnes.

Cette approche facilite la recherche, la mise à jour et la gestion des données d'adresse. Elle suit également les principes de la normalisation des bases de données en évitant la redondance d'informations. on pourrait facilement faire une recherche en fonction d un pays d une ville d'un code postal ou alors d'une rue.

Impact dans le code

remove FEED_ADDRESS et ADD FEED_CITY , FEED_POSTCODE etc..

```
    public static final String FEED_TABLE = "feeds";
    public static final String FEED_ID = BaseColumns._ID;
    public static final String FEED_TITLE = "feed_name";
    public static final String FEED_LINK = "link";
    public static final String FEED_ADDRESS = "address";
    public static final String FEED_SUBSCRIBERS = "subscribers";
    public static final String FEED_OPENS = "opens";
    public static final String FEED_LAST_STORY_DATE = "last_story_date";
    public static final String FEED_AVERAGE_STORIES_PER_MONTH = "average_stories_per_month";
    public static final String FEED_UPDATED_SECONDS = "updated_seconds";
```

Le code de creation de la table Feed serait impacter

```
        ")";

    static final String FEED_SQL = "CREATE TABLE " + FEED_TABLE + " (" +
        FEED_ID + INTEGER + " PRIMARY KEY, " +
        FEED_ACTIVE + TEXT + ", " +
        FEED_ADDRESS + TEXT + ", " +
```

getValues() de feed

```
79      // not vended by API, but used locally for UI
80      public boolean fetchPending;
81
82      public ContentValues getValues() {
83          ContentValues values = new ContentValues();
84          values.put(DatabaseConstants.FEED_ID, feedId);
```

Et toute autre ligne de code manipulant l'adresse d'un feed

7- AJOUT D'UNE LIAISON(FOREIGN KEY) ENTRE SUBSCRIBER ET USER_TABLE

a- Add subscribers as foreign key faisant reference a user_table._id
b- Add primary = (feed_id , subscriber)

Etant donné qu'on a suggère au **point 5** la création d'une table Feed_subscribers pour gérer la liste des subscribers d'un feed, on se rend compte aussi au travers de l'interface de l'application que pour utiliser ou exploiter les fonctionnalites de l'app faudrait d'apport creer un compte ainsi un subscriber représente un utilisateur de la table user_table.

D'où l'attribut subscriber de la table feed_subscriber serait une clé etrangere faisant reference a _id de la table user_table. Puis la clé primaire de la classe serait une composition de feed_id et subscriber. feed_id faisant reference a feeds._id

Ainsi on n'aurait pas des subscribers inexistantes dans le système.

Impact dans le code

En plus des impacts présenter au **point 5**, on aura essentiellement des ajouts de ligne de codes ainsi que la modification des fonctions permettant d'exploiter toutes les données de la table feed.

8- Enlever l'attribut comment_liking_users de la table comment et creation de la table comment_liking_users

a- Remove comment_liking_users from comments
b- Creation de comment_liking_users table
c- Add comment_id to comment_liking_users foreign key to coments table
d- Add user_id to comment_liking_users foreign key to user_table

Ainsi, chaque comment_liking_users est lié à comments via la clé étrangère comment_id et lie a user_table via user_id. Ainsi on pourrait facilement gerer la liste des utilisateurs qui ont liker un commentaire. Cette approche offre plusieurs avantages :

Maintenabilité : Les mises à jour des likes des commentaires par un user se font de manière centralisée dans la table "comment_liking_users ", simplifiant la maintenance.

Normalisation : Les données d'adresse ne sont stockées qu'une seule fois, ce qui réduit la redondance et améliore l'efficacité de la base de données.

L'utilisation d'une table distincte pour représenter les comment_liking_users contribue à une meilleure organisation des données.

Impact dans le code

On prend en compte le fait que plusieurs ligne de code seraient ajouter pour creer la nouvelle table , ainsi la constante COMMENT_LIKING_USERS serait utilisee comme nom de la nouvelle table et non comme attribut de la table comments.

```
public static final String COMMENT_TABLE = "comments";
public static final String COMMENT_ID = BaseColumns._ID;
public static final String COMMENT_STORYID = "comment_storyid";
public static final String COMMENT_TEXT = "comment_text";
public static final String COMMENT_DATE = "comment_date";
public static final String COMMENT_SOURCE_USERID = "comment_source_user";
public static final String COMMENT_LIKING_USERS = "comment_liking_users";
public static final String COMMENT_SHAREDDATE = "comment_shareddate";
public static final String COMMENT_BYFRIEND = "comment_byfriend";
public static final String COMMENT_USERID = "comment_userid";
public static final String COMMENT_ISPSEUDO = "comment_ispseudo";
public static final String COMMENT_ISPLACEHOLDER = "comment_isplaceholder";
```

```
static final String COMMENT_SQL = "CREATE TABLE " + COMMENT_TABLE + " (" +
    COMMENT_DATE + TEXT + ", " +
    COMMENT_SHAREDDATE + TEXT + ", " +
    COMMENT_SOURCE_USERID + TEXT + ", " +
    COMMENT_ID + TEXT + " PRIMARY KEY, " +
    COMMENT_LIKING_USERS + TEXT + ", " +
    COMMENT_BYFRIEND + TEXT + ", " +
    COMMENT_STORYID + TEXT + ", " +
    COMMENT_TEXT + TEXT + ", " +
    COMMENT_USERID + TEXT + ", " +
    COMMENT_ISPSEUDO + TEXT + ", " +
    COMMENT_ISPLACEHOLDER + TEXT +
    "\".
```

9- AJOUT D'UNE COLONNE START_DATE_TIME DANS LA TABLE READING_SESSION

a- Add start_date_time to reading_session table

La table reading_session permet de retenir les informations concernant une session de lecture d'une story , il serait plus intéressant d'enregistrer la date et le temps de début d'une session(éventuellement date_time de fin)de lecture ceci pourrait être utile plus tard lors des sessions d'analyse de données . On pourrait par exemple chercher a savoir le temps moyen qu un utilisateur mets pour lire une story avant de passer a autres chose.

Impact dans le code

Ajout d'une ligne permettant de créer la constante START_DATE_TIME

```
public static final String READING_SESSION_TABLE = "reading_session";
public static final String READING_SESSION_STORY_HASH = "session_story_hash";
```

```
static final String READING_SESSION_SQL = "CREATE TABLE " + READING_SESSION_TABLE + " (" +
    READING_SESSION_STORY_HASH + TEXT +
    ")";
```

10- RENOMER L'ATTRIBUT LOCATION DE LA TABLE USER_TABLE PAR ADRESSE

a- rename location with adress from user_table

"location" permet d'enregistrer l'adresse d'un utilisateur (rue, ville, pays, code postal) il serait plus parlant et plus intuitif de stocker ces informations dans une chaine de caractère.

Impact dans le code

```
public static final String USER_TABLE = "user_table";
public static final String USER_USERID = BaseColumns._ID;
public static final String USER_USERNAME = "username";
public static final String USER_LOCATION = "location";
public static final String USER_PHOTO_URL = "photo_url";
```

```
static final String USER_SQL = "CREATE TABLE " + USER_TABLE + " (" +
    USER_PHOTO_URL + TEXT + ", " +
    USER_USERID + INTEGER + " PRIMARY KEY, " +
    USER_USERNAME + TEXT + ", " +
    USER_LOCATION + TEXT +
    ")";
```