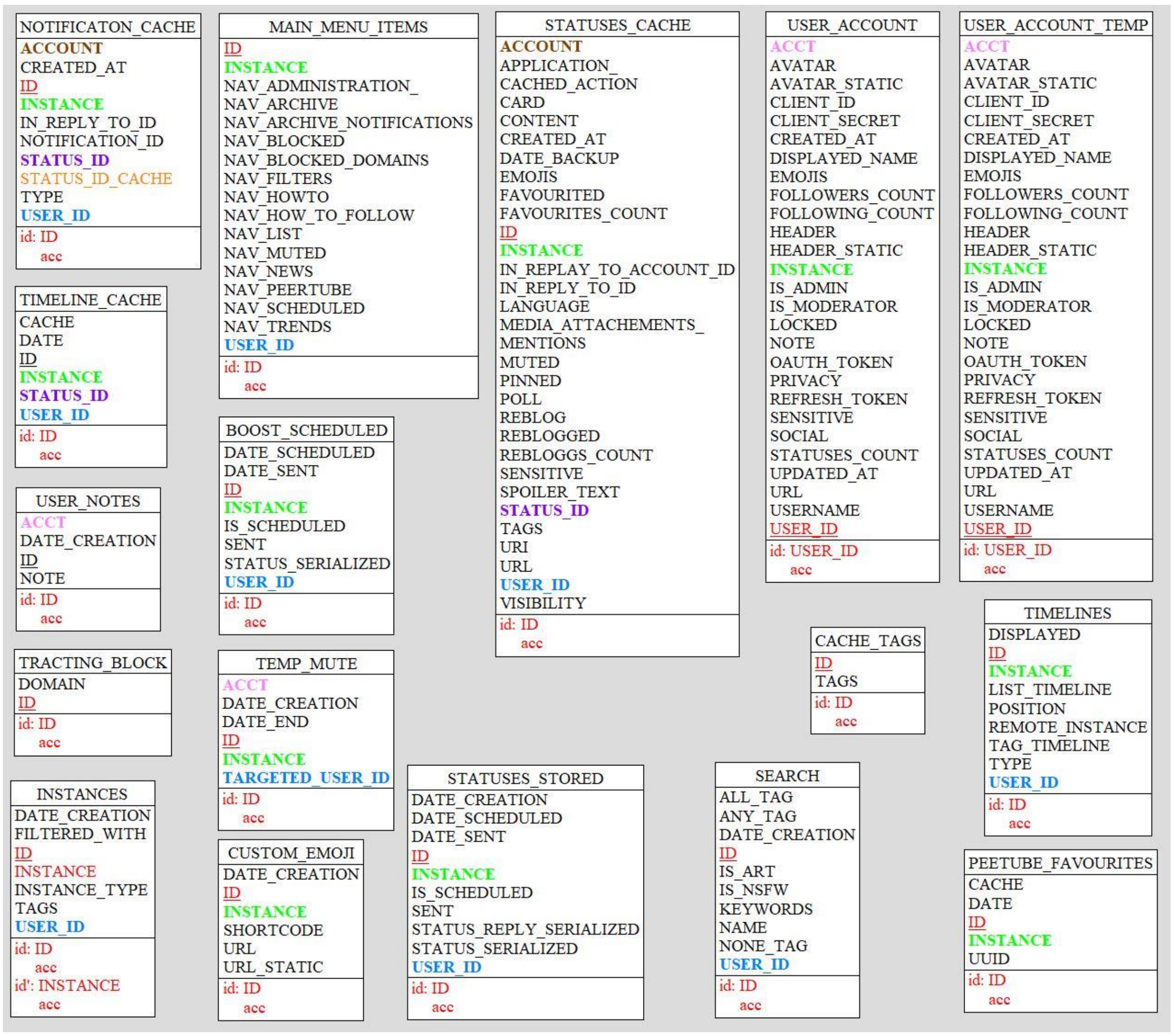
Fedilab database Schema

Enriched by implicit foreign keys from the schema and from the queries



USER_NOTES --> ACCT (Foreign key) ref USER_ACCOUNT

```
        ShowAccountActivity --> line 791
        UserNote userNote = new NotesDAO(ShowAccountActivity.this,
db).getUserNote(account.getAcct());
        ShowAccountActivity --> line 812
        UserNote userNote1 = new NotesDAO(ShowAccountActivity.this,
db).getUserNote(account.getAcct());
            if (userNote1 == null) {
                userNote1 = new UserNote();
                userNote1.setAcct(account.getAcct());
            }
            userNote1.setNote(input.getText().toString());
            new NotesDAO(ShowAccountActivity.this, db).insertInstance(userNote1);
```

TEMP_MUTE --> ACCT (Foreign key) ref USER_ACCOUNT

```
        public void insert(Account account, String targeted_id, Date date) {
    ContentValues values = new ContentValues();
    values.put(Sqlite.COL_TARGETED_USER_ID, targeted_id);
    values.put(Sqlite.COL_ACCT, account.getAcct());
    values.put(Sqlite.COL_INSTANCE, account.getInstance());
    values.put(Sqlite.COL_DATE_CREATION, Helper.dateToString(new Date()));
    values.put(Sqlite.COL_DATE_END, Helper.dateToString(date));
    //Inserts temp mute
    try {
      db.insert(Sqlite.TABLE_TEMP_MUTE, null, values);
    } catch (Exception ignored) {
    }
  }
```

INSTANCES --> USER_ID (Foreign key) ref USER_ACCOUNT

```
        static final String COL_USER_ID = "USER_ID";
        public void insertInstance(String instanceName, String id, String type) {
                        ContentValues values = new ContentValues();
                        values.put(Sqlite.COL_INSTANCE, instanceName.trim());
                        values.put(Sqlite.COL_USER_ID, id);
                        values.put(Sqlite.COL_INSTANCE_TYPE, type);
                        values.put(Sqlite.COL_DATE_CREATION, Helper.dateToString(new Date()));
                        //Inserts search
                        try {
                                db.insert(Sqlite.TABLE_INSTANCES, null, values);
                        } catch (Exception ignored) {
                        }
                }
```

TIMELINES --> USER_ID (Foreign key) ref USER_ACCOUNT

```
        static final String COL_USER_ID = "USER_ID";
        public void insert(ManageTimelines timeline) {
    SharedPreferences sharedpreferences = context.getSharedPreferences(Helper.APP_PREFS,
Context.MODE_PRIVATE);
    String userId = sharedpreferences.getString(Helper.PREF_KEY_ID, null);
    String instance = Helper.getLiveInstance(context);
    ContentValues values = new ContentValues();
    values.put(Sqlite.COL_TYPE, ManageTimelines.typeToDb(timeline.getType()));
    values.put(Sqlite.COL_DISPLAYED, timeline.isDisplayed());
    values.put(Sqlite.COL_POSITION, timeline.getPosition());
    values.put(Sqlite.COL_USER_ID, userId);
    values.put(Sqlite.COL_INSTANCE, instance);
    if (timeline.getTagTimeline() != null)
      values.put(Sqlite.COL_TAG_TIMELINE,
Helper.tagTimelineToStringStorage(timeline.getTagTimeline()));
    if (timeline.getRemoteInstance() != null)
      values.put(Sqlite.COL_REMOTE_INSTANCE,
Helper.remoteInstanceToStringStorage(timeline.getRemoteInstance()));
    if (timeline.getListTimeline() != null)
      values.put(Sqlite.COL_LIST_TIMELINE,
Helper.listTimelineToStringStorage(timeline.getListTimeline()));
```

```
        try {
            db.insert(Sqlite.TABLE_TIMELINES, null, values);
        } catch (Exception ignored) {
        }
    }
```

TIMELINE_CACHE --> USER_ID (Foreign key) ref USER_ACCOUNT
TIMELINE_CACHE --> INSTANCE (Foreign key) ref USER_ACCOUNT

```
            API.java --> line 6291
            JSONObject resobj = jsonArray.getJSONObject(i);
        Status status = parseStatuses(context, resobj);
        Status alreadyCached = new TimelineCacheDAO(context, db).getSingle(status.getId());
        if (alreadyCached == null && account != null && account.getId() != null &&
account.getInstance() != null) {
            new TimelineCacheDAO(context, db).insert(status.getId(), resobj.toString(), account.getId(),
account.getInstance());
        }
```

TIMELINE_CACHE --> STATUS_ID_CACHE (Foreign key) ref STATUSES_CACHE

```
            API.java --> line 6291
                    Status status = parseStatuses(context, resobj);
        Status alreadyCached = new TimelineCacheDAO(context, db).getSingle(status.getId());
        if (alreadyCached == null && account != null && account.getId() != null &&
account.getInstance() != null) {
            new TimelineCacheDAO(context, db).insert(status.getId(), resobj.toString(), account.getId(),
account.getInstance());
        }
```

NOTIFICATION_CACHE --> STATUS_ID_CACHE (Foreign key) ref STATUSES_CACHE

```
            NotificationCacheDAO.java --> line 69
                    method insertNotification(Notification notification, String userId, String instance)
```

STATUSES_CACHE --> STATUS_ID_CACHE (Foreign key) ref STATUSES_CACHE

```
            StatusCacheDAO.java --> line 74
                    insertStatus(int cacheType, Status status, String userId, String instance)
```