Automatically deactivating inactive Jira users

To save on licensing costs, it is sometimes useful to automatically deactivate Jira users who haven't logged in within a certain period, say 6 months. Here we review the options, and provide a ScriptRunner script (source in github) that does the job.

Which users can we deactivate?

First, it's worth thinking through the rules for which users you want to be automatically deactivated:

1. "**User has not logged in in X months**" is a good start.
2. How about users that have never logged in? It depends on the age of their user account: if it was created yesterday, but they haven't logged in yet, leave the account alone; if it was created last year and they haven't logged in, it should be deactivated. So let's also deactivate **users whose account was created more than X months ago, AND who have never logged in**.

    Incidentally, if you do a web search for Jira deactivate inactive users' you will see many solutions, like this ScriptRunner script from Adaptavist, that don't handle this edge case (probably because Jira's regular API doesn't expose the 'created' date).

3. Jira instances often have multiple directories. It's not possible to deactivate users in LDAP / AD User Directories, so let's add the criteria: **users are in the internal (id 1) or Crowd directory (e.g. id 10000)**.
4. Does your Jira have an 'admin' role account on the Internal directory, only used in emergencies when the external user provider (Crowd, LDAP, external Jira) is offline? This shouldn't be automatically deactivated. We must add the rule **exclude emergency access accounts**.
5. Does your Jira contain any 'role' accounts never log in, but are still valid? Perhaps a role account like 'qa' that is assigned issues so that qa@mycompany.com gets notified? If so, we need a **exclude role accounts that are used but never log in** rule to prevent these role accounts getting deactivated.

Generic ScriptRunner Solution

Our first generic solution is a ScriptRunner for Jira Groovy script. It deactivates users matching rules 1, 2 and 3, namely **users in the Internal Directory (1) who have not logged in X months, or who have never logged in to an account created more than X months ago.**

```
/**
 * Script that deactivates users who have not logged in within the last 6 months.
 * See https://www.redradishtech.com/pages/viewpage.action?pageId=11796495
 * Loosely based on Adaptavist's sample at https://www.adaptavist.com/doco/display/SFJ
/Automatically+deactivate+inactive+JIRA+users
 * Adaptavist's script has a bug where if a user has *never* logged in, they will never be deactivated. We fix
this by checking the user creation date too.
 *
 * Note: I suggest using the SQL variant (deactivate-inactive-jira-users.groovy) of this script in production.
 *
 * jeff@redradishtech.com, 5/Jun/19
 * v1.0
*/
import com.atlassian.crowd.embedded.api.User
import com.atlassian.crowd.embedded.api.CrowdService
import com.atlassian.crowd.embedded.api.UserWithAttributes
import com.atlassian.crowd.embedded.impl.ImmutableUser
import com.atlassian.crowd.embedded.api.SearchRestriction
import com.atlassian.jira.bc.user.UserService
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.user.ApplicationUser
import com.atlassian.jira.user.ApplicationUsers
import com.atlassian.crowd.search.query.entity.restriction.constants.UserTermKeys
import com.atlassian.crowd.search.query.entity.restriction.constants.DirectoryTermKeys
import com.atlassian.crowd.search.builder.Restriction
import com.atlassian.crowd.search.builder.QueryBuilder
import com.atlassian.crowd.search.query.entity.EntityQuery
import com.atlassian.crowd.search.EntityDescriptor
import com.atlassian.jira.crowd.embedded.ofbiz.OfBizUser

import org.joda.time.DateTime;
import org.joda.time.Period;

CrowdService crowdService = ComponentAccessor.crowdService

// In a perfect world Jira would let us find exactly the users we want to deactivate with CQL expression
'lastLogin > -6m OR (!lastLogin AND createdDate<-6m)'. Sadly 'lastLogin.lastLoginMillis' is considered a
'secondary' property which Crowd CQL doesn't support (https://developer.atlassian.com/server/crowd/crowd-query-
language/). Crowd CQL also doesn't support relative dates like '-6m'. Nor does it support finding users from a
particular directory (some of ours may be read-only).
//
// So instead we search for all active users, and manually check the lastLogin/create date.
//
// First we search for active users. We don't use UserUtil.getUsers() (unlike every other example on the web),
as that returns ApplicationUsers for which it is impossible to get the underlying Ofbiz object, which we need
to get the created_date. Instead we use CrowdService.search(), which returns OfBizUsers (https://docs.atlassian.
com/software/jira/docs/api/7.2.0/com/atlassian/jira/crowd/embedded/ofbiz/OfBizUser.html).
// QueryBuilder has excellent Javadocs at https://docs.atlassian.com/atlassian-crowd/3.2.3/com/atlassian/crowd
/search/builder/QueryBuilder.html
// This returns an iterable of OfBizUsers (https://docs.atlassian.com/software/jira/docs/api/7.2.0/com/atlassian
/jira/crowd/embedded/ofbiz/OfBizUser.html) actually
def SearchRestriction active = Restriction.on(UserTermKeys.ACTIVE).exactlyMatching(Boolean.TRUE)
def foundUsers = crowdService.search(
        QueryBuilder.queryFor(User.class, EntityDescriptor.user()).with(active).returningAtMost(EntityQuery.
ALL_RESULTS)
        ) as ArrayList<OfBizUser>;

log.info "Checking ${foundUsers.size()} active users for possible deactivation-due-to-inactivity"

def shouldDeactivate(User user, DateTime lastUsed) {
        def INACTIVITY_PERIOD = Period.parse("P1Y") // Period of inactivity after which user is deactivated.
The format is https://en.wikipedia.org/wiki/ISO_8601#Durations
        // JodaTime 'time ago' calculation: https://stackoverflow.com/a/3859313/7538322
        def expiryDate = lastUsed.plus(INACTIVITY_PERIOD);
        log.info "User ${user.name} will be deactivated after ${expiryDate}";
        return expiryDate.isBeforeNow();
}

def deactivate(User user) {
        UserService userService = ComponentAccessor.getComponent(UserService)
        ApplicationUser updateUser = ApplicationUsers.from(ImmutableUser.newUser(user).active(false).toUser());
        UserService.UpdateUserValidationResult updateUserValidationResult = userService.validateUpdateUser
(updateUser);
        if (updateUserValidationResult.isValid()) {
                // Comment out this line to do a dry run:
                //userService.updateUser(updateUserValidationResult)
                return true
        } else {
                log.error "Update of ${user.name} failed: ${updateUserValidationResult.getErrorCollection().
getErrors().entrySet().join(',')}";
                return false
        }
}

long count = 0
// Restrict to our Internal directory, with ID 1, otherwise we'll get errors trying to modify read-only LDAP
users.
foundUsers.findAll { ofbizUser -> ofbizUser.directoryId == 1 }.each { ofbizUser ->
        def UserWithAttributes user = crowdService.getUserWithAttributes(ofbizUser.getName());
        // FIXME: also need to consider 'lastAuthenticated' if Confluence or an external client uses Jira for
auth, and you want these authentications to be considered 'activity'
        String lastLoginMillis = user.getValue('login.lastLoginMillis');
        if (lastLoginMillis?.isNumber()) {
                DateTime lastLogin = new DateTime(Long.parseLong(lastLoginMillis));
                if (shouldDeactivate(user, lastLogin) && deactivate(user)) {
                        log.warn "Deactivated ${user.name}, who was last active on ${lastLogin}";
                        count++
                }
        } else if (!lastLoginMillis) {
                DateTime created = new DateTime(ofbizUser.getCreatedDate());
                if (shouldDeactivate(user, created) && deactivate(user)) {
                        log.warn "Deactivated ${user.name}, who has never logged in and was created on
${created}";
                        count++;
                }
        }
}

"${count} inactive users automatically deactivated.\n"
```

To use this script to automatically deactivate users:

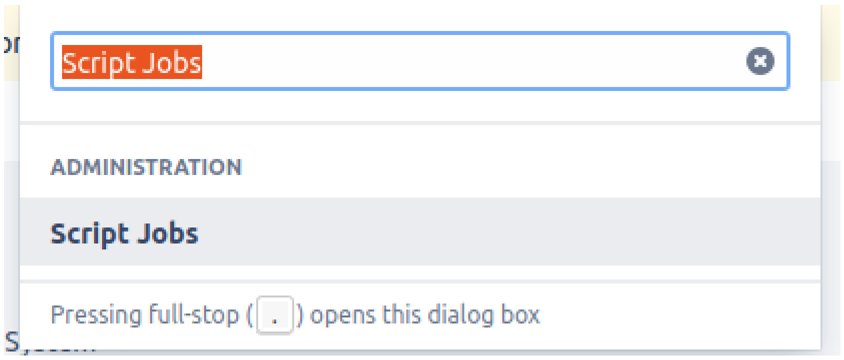
- Checkout the script from the github repository to `$JIRAHOME/scripts`:

  ```
  cd $JIRAHOME/scripts
  git clone https://github.com/redradishtech/jira-user-deactivator-groovy
  chgrp -R jira jira-user-deactivator-groovy    # Ensure Jira has read access.
  ```

- If you first want to see what *would* happen without deactivating anyone, edit `deactivate-inactive-jira-users-nonsql.groovy` and comment out the `updateUser` line:

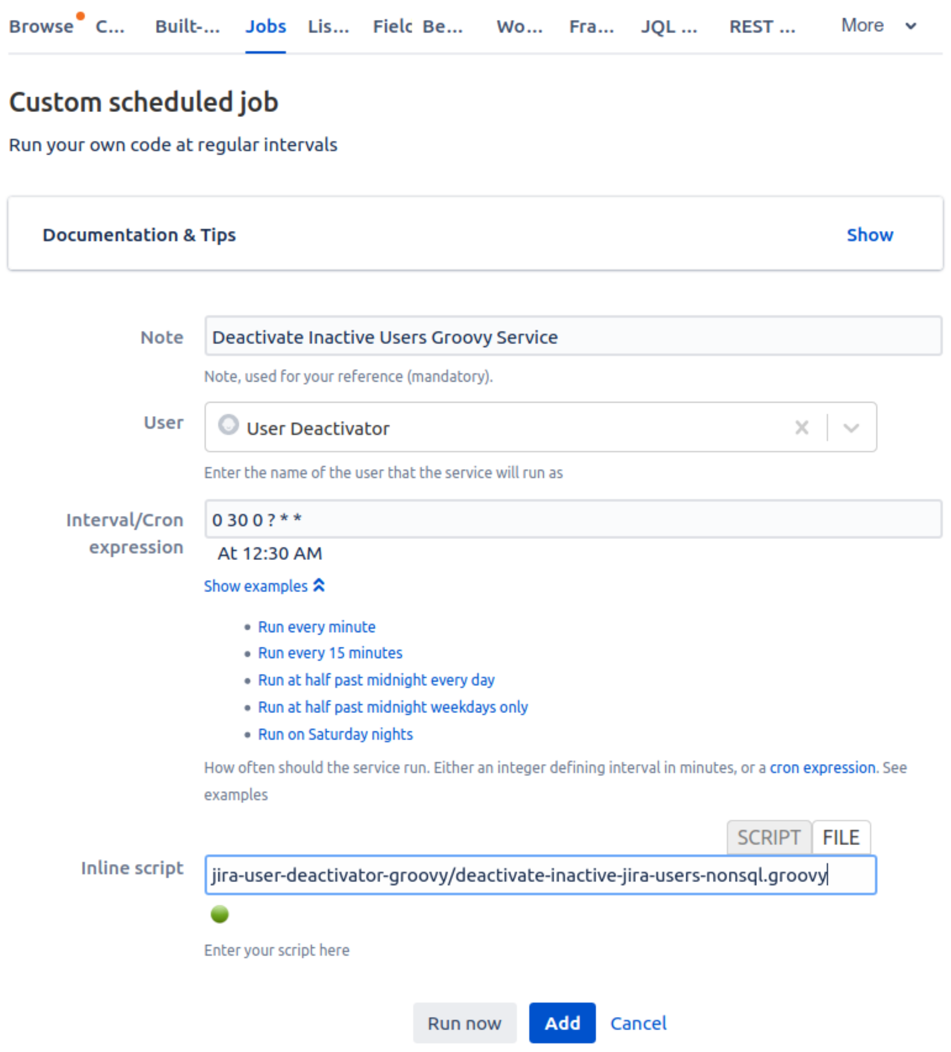
  ```
  // Comment out this line to do a dry run:
  // userService.updateUser(updateUserValidationResult)
  ```

- Go to the ScriptRunner **Jobs** tab, e.g. by typing 'gg' then 'Script jobs':

  (ScriptRunner Jobs is just a nice UI around Jira Services. In the past one would have created a **com.onresolve.jira.groovy.GroovyService** Jira Service directly)

- Create a *Custom Scheduled Job:



Custom scheduled job

Run your own code at regular intervals

Documentation & Tips — Show

Note: Deactivate Inactive Users Groovy Service

Note, used for your reference (mandatory).

User: User Deactivator

Enter the name of the user that the service will run as

Interval/Cron expression: 0 30 0 ? * *

At 12:30 AM

Show examples

- Run every minute
- Run every 15 minutes
- Run at half past midnight every day
- Run at half past midnight weekdays only
- Run on Saturday nights

How often should the service run. Either an integer defining interval in minutes, or a cron expression. See examples

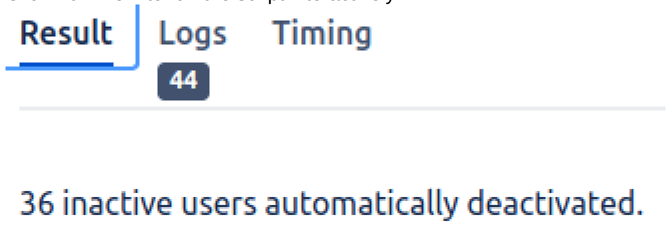
SCRIPT FILE

Inline script: jira-user-deactivator-groovy/deactivate-inactive-jira-users-nonsql.groovy

Enter your script here

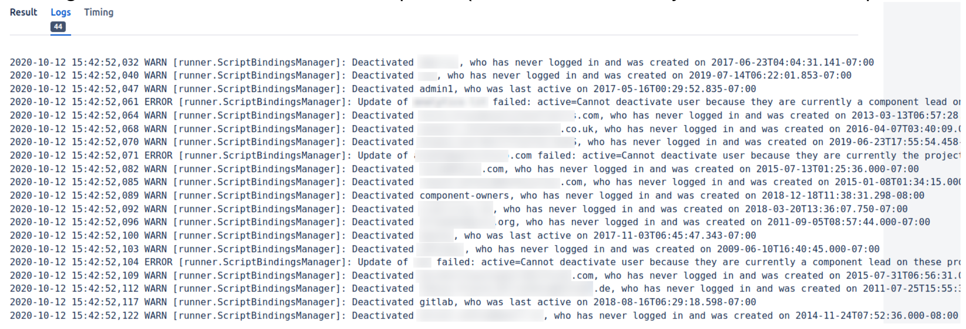
For **User** pick an account with the **Jira Administrators** global permission. You might like to create a dedicated role account ('deactivator') as I have in the screenshot, so that the Job isn't tied to a user account, but this does cost a license slot.

- Click **Run Now** to run the script interactively.



36 inactive users automatically deactivated.

The **Logs** tab will show what actions the script took (or would have taken if you commented out `updateUser`):



```
2020-10-12 15:42:52,032 WARN [runner.ScriptBindingsManager]: Deactivated [redacted], who has never logged in and was created on 2017-06-23T04:04:31.141-07:00
2020-10-12 15:42:52,040 WARN [runner.ScriptBindingsManager]: Deactivated [redacted], who has never logged in and was created on 2019-07-14T06:22:01.853-07:00
2020-10-12 15:42:52,047 WARN [runner.ScriptBindingsManager]: Deactivated admin1, who was last active on 2017-05-16T00:29:52.835-07:00
2020-10-12 15:42:52,061 ERROR [runner.ScriptBindingsManager]: Update of [redacted] failed: active=Cannot deactivate user because they are currently a component lead or
2020-10-12 15:42:52,064 WARN [runner.ScriptBindingsManager]: Deactivated [redacted].com, who has never logged in and was created on 2013-03-13T06:57:28
2020-10-12 15:42:52,068 WARN [runner.ScriptBindingsManager]: Deactivated [redacted].co.uk, who has never logged in and was created on 2016-04-07T03:40:09.
2020-10-12 15:42:52,070 WARN [runner.ScriptBindingsManager]: Deactivated [redacted], who has never logged in and was created on 2019-06-23T17:55:54.458
2020-10-12 15:42:52,071 ERROR [runner.ScriptBindingsManager]: Update of [redacted].com failed: active=Cannot deactivate user because they are currently the project
2020-10-12 15:42:52,082 WARN [runner.ScriptBindingsManager]: Deactivated [redacted].com, who has never logged in and was created on 2015-07-13T01:25:36.000-07:00
2020-10-12 15:42:52,085 WARN [runner.ScriptBindingsManager]: Deactivated [redacted].com, who has never logged in and was created on 2015-01-08T01:34:15.000
2020-10-12 15:42:52,089 WARN [runner.ScriptBindingsManager]: Deactivated component-owners, who has never logged in and was created on 2018-12-18T11:38:31.298-08:00
2020-10-12 15:42:52,092 WARN [runner.ScriptBindingsManager]: Deactivated [redacted], who has never logged in and was created on 2018-03-20T13:36:07.750-07:00
2020-10-12 15:42:52,096 WARN [runner.ScriptBindingsManager]: Deactivated [redacted].org, who has never logged in and was created on 2011-09-05T08:57:44.000-07:00
2020-10-12 15:42:52,100 WARN [runner.ScriptBindingsManager]: Deactivated [redacted], who was last active on 2017-11-03T06:45:47.343-07:00
2020-10-12 15:42:52,103 WARN [runner.ScriptBindingsManager]: Deactivated [redacted], who has never logged in and was created on 2009-06-10T16:40:45.000-07:00
2020-10-12 15:42:52,104 ERROR [runner.ScriptBindingsManager]: Update of [redacted] failed: active=Cannot deactivate user because they are currently a component lead on these pr
2020-10-12 15:42:52,109 WARN [runner.ScriptBindingsManager]: Deactivated [redacted].com, who has never logged in and was created on 2015-07-31T06:56:31.
2020-10-12 15:42:52,112 WARN [runner.ScriptBindingsManager]: Deactivated [redacted].de, who has never logged in and was created on 2011-07-25T15:55:
2020-10-12 15:42:52,117 WARN [runner.ScriptBindingsManager]: Deactivated gitlab, who was last active on 2018-08-16T06:29:18.598-07:00
2020-10-12 15:42:52,122 WARN [runner.ScriptBindingsManager]: Deactivated [redacted], who has never logged in and was created on 2014-11-24T07:52:36.000-08:00
```

- If all looks good, click **Add** to permanently add the Job.

ScriptRunner Solution with SQL Rules

How about if your rules for who to deactivate need to be more sophisticated than just 'user hasn't logged in in 6 months'?

Consider the use of role accounts, as would exist if you crowdsource the triaging of issues. Role accounts are assigned issues, but never log in. The script above would deactivate role accounts, causing chaos.

So we need to refine our rule for which accounts can be deactivated. For role accounts, we know they are being frequently assigned issues. So we can use the "date of last assign" as another indicator that the account is used.

Figuring out our last login date in code was painful enough: calculating the last assign is a bridge too far. This is a job for SQL, not code.

Our solution is as follows:

- Create a SQL View identifying accounts that can be deactivated. This SQL will take into account when the user last logged in AND when last they were assigned an issue. Any other rules you like can be added to the SQL.
- We modify the Groovy script to read usernames from the SQL View, and deactivate those accounts in code.

Here is Postgres-flavoured SQL, creating a `queries.inactive_users` view, of users that can be deactivated (source at https://github.com/redradishtech/jira-user-deactivator-groovy/blob/master/active_users.sql):

```
-- Creates a queries.inactive_users view in a Jira database, listing inactive user accounts that might be
deactivated by deactivate-inactive-jira-users.groovy
--
-- Last updated: 14/Oct/24
-- See https://www.redradishtech.com/display/KB/Automatically+deactivating+inactive+Jira+users

-- @provides queries.inactive_users_all
drop view if exists queries.inactive_users_all cascade;
create schema if not exists queries;
create view queries.inactive_users_all AS
WITH userlogins AS (
        SELECT DISTINCT ON (user_name) -- If LDAP is used there will be 2 directories ('LDAP' and 'Jira
Internal Directory'), each with a duplicate set of cwd_user rows. The "DISTINCT ON (user_name) ... ORDER BY
user_name, cwd_directory.directory_position ASC" gets us only the first cwd_user record by directory
'position', i.e. the one actually authenticated against that will have up-to-date lastLogin stats.
                cwd_user.directory_id
        , user_name
        , email_address
        , cwd_user.created_date
        , timestamp with time zone 'epoch'+lastlogins.attribute_value::numeric/1000 * INTERVAL '1 second' AS
lastlogin
        , timestamp with time zone 'epoch'+lastauths.attribute_value::numeric/1000 * INTERVAL '1 second' AS
lastauth   -- REST queries count as authentications, not logins
        FROM
        cwd_user
        JOIN (select * from cwd_directory WHERE active=1) as cwd_directory ON cwd_user.directory_id =
cwd_directory.id
        JOIN cwd_membership ON (cwd_membership.lower_child_name=cwd_user.lower_user_name and cwd_membership.
directory_id=cwd_directory.id)
        JOIN (
                select * from globalpermissionentry WHERE permission IN ('USE', 'ADMINISTER')
             ) AS globalpermissionentry ON cwd_membership.lower_parent_name=globalpermissionentry.group_id
             LEFT JOIN LATERAL (select * from cwd_user_attributes WHERE directory_id=cwd_directory.id AND
attribute_name in ('login.lastLoginMillis')) lastlogins ON lastlogins.user_id=cwd_user.id
             LEFT JOIN LATERAL (select * from cwd_user_attributes WHERE directory_id=cwd_directory.id AND
attribute_name in ('lastAuthenticated')) lastauths ON lastauths.user_id=cwd_user.id
        WHERE cwd_user.active=1
                -- Note that we cannot have any further WHERE clauses, e.g. limiting email_address to a certain
pattern. Say user 'jjsmith' Internal user has email jjsmith@foo.com, and is shadowed by 'jjsmith' in LDAP, with
email jjsmith@bar.com. If we limit to 'not like '%@bar.com' then only the inactive foo.com username's
attributes would be found. Instead our caller must do the filtering.
                -- Specific exceptions can be added to the 'never-deactivate' group.
                ORDER BY user_name, cwd_directory.directory_position ASC
)
, lastassigns AS (
        SELECT DISTINCT
        newvalue AS user_name
        , max(created) AS lastassign
        FROM changegroup cg
        JOIN changeitem ci ON cg.id = ci.groupid
        WHERE field='assignee' group by 1
)
, lastwatch AS (
        SELECT cwd_user.user_name
                   , max(userassociation.created) AS lastwatch
        FROM app_user
        LEFT JOIN userassociation ON userassociation.source_name=app_user.user_key
        JOIN cwd_user USING (lower_user_name)
        WHERE association_type='WatchIssue'
        GROUP BY user_name )
, lastreactivate AS (
        -- Check the audit log for account reactivations.
        -- If an admin recently reactivated a dormant account, we don't want to deactivate it due to the user's
inactivity
        select
                app_user.lower_user_name AS user_name
                ,max(to_timestamp("ENTITY_TIMESTAMP"/1000)::date) AS lastreactivate
         from "AO_C77861_AUDIT_ENTITY" JOIN app_user ON app_user.user_key="PRIMARY_RESOURCE_ID" where
"PRIMARY_RESOURCE_TYPE"='USER' AND "CHANGE_VALUES" ~ '"from":"Inactive","to":"Active"}]$'
        group by user_name
)
, neverdeactivate AS (
        select cwd_user.user_name from cwd_user JOIN cwd_membership ON cwd_user.id=cwd_membership.child_id JOIN
cwd_group ON cwd_membership.parent_id=cwd_group.id WHERE cwd_group.group_name='never-deactivate'
)
SELECT distinct
        user_name
        , directory_id
        , email_address
        , to_char(created_date, 'YYYY-MM-DD') AS created
        , to_char(lastlogin, 'YYYY-MM-DD') AS lastlogin
        , to_char(lastauth, 'YYYY-MM-DD') AS lastauth
        , to_char(lastassign, 'YYYY-MM-DD') AS lastassign
        , to_char(lastwatch, 'YYYY-MM-DD') AS lastwatch
        , to_char(lastreactivate, 'YYYY-MM-DD') AS lastreactivate
        , (select count(*) from jiraissue where assignee=userlogins.user_name) AS assigneecount
FROM userlogins
LEFT JOIN lastassigns USING (user_name)
LEFT JOIN lastreactivate USING (user_name)
LEFT JOIN lastwatch USING (user_name)
 WHERE
        (created_date < now() - '3 months'::interval)
        AND ((lastlogin < now() - '3 months'::interval) OR lastlogin is null)
        AND ((lastauth < now() - '3 months'::interval) OR lastauth is null)
        AND ((lastassign < now() - '3 months'::interval) OR lastassign is null)
        AND ((lastreactivate < now() - '3 months'::interval) OR lastreactivate is null)
        -- Note that we don't filter on 'lastwatch' (although we include it), as it is not a good sign of user
liveness - at least, no better than lastlogin (people need to be logged in to watch issues). lastwatch is
displayed for informational purposes - if the date is later than lastlogin, it means someone other than
user_name added user_name as a watcher after they last logged in.
        AND NOT EXISTS (select * from neverdeactivate where user_name=userlogins.user_name)
ORDER BY lastlogin desc nulls last ;
GRANT select on queries.inactive_users to jira_ro;

-- @provides queries.inactive_users
-- Inactive customer accounts
drop view if exists queries.inactive_users;

CREATE VIEW queries.inactive_users AS
SELECT user_name,
       email_address,
       created,
       lastlogin,
       lastauth,
       lastassign,
       lastwatch,
       lastreactivate
FROM queries.inactive_users_all
WHERE email_address not like '%@mycompany.com'
  AND directory_id=1;
```

Here is a corresponding Groovy script that reads usernames from the view, and deactivates those accounts (source):

```
/**
 * Script that deactivates users who have not logged in within the last X months, based on a SQL query.
 * See See https://www.redradishtech.com/pages/viewpage.action?pageId=11796495
 *
 * Loosely based on Adaptavist's sample at https://www.adaptavist.com/doco/display/SFJ
/Automatically+deactivate+inactive+JIRA+users
 *
 * Instead of trying to figure out which users to deactivate in code, we instead rely on a queries.
inactive_users table or view being defined in the Jira database. The SQL can then be as fancy or customized as
needed: e.g. we might want to avoid deactivating role accounts which are assigned issues but never log in. The
only requirement for our table or view is that a 'user_name' column must exist.
 *
 * jeff@redradishtech.com, 19/Dec/2019
 * v1.0
*/
import com.atlassian.jira.user.ApplicationUser
import com.atlassian.jira.user.ApplicationUsers
import com.atlassian.jira.bc.user.UserService
import com.atlassian.crowd.embedded.api.User
import com.atlassian.crowd.embedded.api.UserWithAttributes
import com.atlassian.crowd.embedded.api.CrowdService
import com.atlassian.crowd.embedded.impl.ImmutableUser


/** Deactivate a user.
 * @return null on success, or a String error message.
 */
def String deactivate(String user_name) {
        CrowdService crowdService = ComponentAccessor.crowdService
        def UserWithAttributes user = crowdService.getUserWithAttributes(user_name);
        if (!user.active) return "Already inactive";
        UserService userService = ComponentAccessor.getComponent(UserService)
        ApplicationUser updateUser = ApplicationUsers.from(ImmutableUser.newUser(user).active(false).toUser());
        UserService.UpdateUserValidationResult updateUserValidationResult = userService.validateUpdateUser
(updateUser);
        if (updateUserValidationResult.isValid()) {
                // Comment out this line to do a dry run:
                userService.updateUser(updateUserValidationResult)
                return null
        } else {
                return updateUserValidationResult.getErrorCollection().getErrors().entrySet().join(',')
        }
}

// https://scriptrunner.adaptavist.com/latest/jira/recipes/misc/connecting-to-databases.html
import com.atlassian.jira.component.ComponentAccessor
import groovy.sql.Sql
import org.ofbiz.core.entity.ConnectionFactory
import org.ofbiz.core.entity.DelegatorInterface

import java.sql.Connection

def delegator = (DelegatorInterface) ComponentAccessor.getComponent(DelegatorInterface)
String helperName = delegator.getGroupHelperName("default")

def sqlStmt = """select * from queries.inactive_users;"""

Connection conn = ConnectionFactory.getConnection(helperName)
Sql sql = new Sql(conn)

log.warn "Beginning inactive user deactivation run"
long count = 0
try {
    sql.eachRow(sqlStmt) {
    // https://stackoverflow.com/questions/50041526/how-to-read-each-row-in-a-groovy-sql-statement
        def errmsg = deactivate(it['user_name'] as String);
        if (!errmsg) {
                log.warn "Deactivated ${it['user_name']}: ${it}";
                count++
        } else {
                log.error "Failed to deactivate ${it['user_name']}: ${errmsg}";
        }
    }
}
finally {
    sql.close()
}
"${count} inactive users automatically deactivated.\n"
```

The script should be installed in `$JIRAHOME/scripts/jira-user-deactivator-groovy/deactivate_inactive_users.groovy` ande invoked automatically as a service, as described above.

Other options

Before writing the ScriptRunner Groovy scripts above, I considered (and discarded) a few other options.

Plugins

As of 06 Jun 2019, the only relevant plugin is Manage Inactive Users. This free plugin also supports deactivating users in external user bases like Okta and Google Apps.

~~I am waiting on feedback from the author before passing judgement.~~ 17 Jul 2019 The MIU plugin author released new versions that IMO bring the plugin into the realms of usability (previously even the definition of 'inactive' was completely opaque and unmodifiable). For users not keen on Groovy, I suggest giving this plugin a serious try.

REST Script

Without any plugins, the cleanest solution would be a script utilizing Jira's REST interface. The script would search for inactive users with Crowd CQL, then deactivate them. A REST solution would have the advantage of also working on Cloud Jira.

As a preliminary experiment, here is a demonstration of running Crowd Query Language against Jira:

```
# curl --silent --get -u cli:cli http://jira.localhost/rest/usermanagement/1/search -d 'entity-type=user' --
data-urlencode 'restriction=active=true and email=jeff@redradishtech.com and createdDate>2013-09-02'   --header
'Accept: application/json'  | jq .
{
  "expand": "user",
  "users": [
    {
      "link": {
        "href": "http://jira.localhost/rest/usermanagement/1/user?username=jturner",
        "rel": "self"
      },
      "name": "jturner"
    }
  ]
}
```

(create the 'cli' username/password in JIra's "User Server" admin page)

In a perfect world Jira would let us find exactly the users we want to deactivate with Crowd Query Language expression `lastLogin > -6m OR (! lastLogin AND createdDate<-6m)`. Sadly 'lastLogin.lastLoginMillis' is considered a 'secondary' property which Crowd CQL doesn't support. Crowd CQL also doesn't support relative dates like '-6m'.

Without decent CQL support, our REST script would need to retrieve *every* active user, iterate through them, and check each user's last login date / created date. This may be slow and memory-intensive.

Another spanner in the works: Jira only gained a user deactivate REST method in JIRA 8.3+. See **JRASERVER-44801** - Getting issue details... STATUS . Users of earlier releases would have to write their own REST endpoint using ScriptRunner: https://www.mos-eisley.dk/display/ATLASSIAN/Deactivate+a+User+via+REST

Given the potential slowness, and lack of REST support, I didn't pursue this route too far.

Direct database hackery

We have SQL identifying exactly what accounts we want to deactivate. Couldn't we just change the SELECT to an UPDATE that sets `active=0` , and do the deactivation directly in the database?

Atlassian apps generally have caching layers that prevent direct database changes from working, but in my experience, Crowd picks up changes to `cwd_user` immediately, so this approach could work. The Crowd Query Language (CQL) is presumably implemented with Lucene, and would have stale results. Is this critical?

I haven't researched this much further, as instances I work with all have ScriptRunner available.

What about Confluence?

05 Aug 2021 There is now a Confluence version of the `inactive_users` SQL at https://github.com/redradishtech/jira-user-deactivator-groovy/blob/master/inactive_users_confluence.sql. Note that the SQL doesn't limit itself to Internal directories yet. I haven't made a Groovy deactivation script based around it yet.

Conclusion

Using ScriptRunner, we have implemented a means for Jira to automatically deactivate inactive users, thus saving license slots. This is (to my knowledge, as of 07 Jun 2019) the only implementation that handles never-logged-in users. Users who require more flexibility can use the SQL-augmented approach.