Events Modeling

This section explains how to model your application data as events.

Entity: it's the real world object involved in the events. The entity may perform the events, or interact with other entity (which became `targetEntity` in an event).

For example, your application may have users and some items which the user can interact with. Then you can model them as two entity types: **user** and **item** and the entityId can uniquely identify the entity within each entityType (e.g. user with ID 1, item with ID 1).

An entity may perform some events (e.g user 1 does something), and entity may have properties associated with it (e.g. user may have gender, age, email etc). Hence, **events** involve **entities** and there are two types of events, respectively:

1. Generic events performed by an entity.
2. Special events for recording changes of an entity's properties

They are explained in details below.

1. Generic events performed by an entity

Whenever the entity performs an action, you can describe such event as `entity "verb" targetEntity with "some extra information"`. The *"targetEntity"* and *"some extra information"* can be optional.

The *"verb"* can be used as the name of the *"event"*. The *"some extra information"* can be recorded as `properties` of the event.

The following are some simple examples:

- user-1 sign-up

```
{
  "event" : "sign-up",
  "entityType" : "user",
  "entityId" : "1"
}
```

- user-1 views item-1 *(with targetEntity)*

```
{
  "event" : "view",
  "entityType" : "user",
  "entityId" : "1",
  "targetEntityType" : "item",
  "targetEntityId" : "1"
}
```

- user-1 buys item-1

```
{
  "event" : "buy",
  "entityType" : "user",
  "entityId" : "1",
  "targetEntityType" : "item",
  "targetEntityId" : "1",
}
```

2. Special events for recording changes of an entity's properties

The generic events described above are used to record general actions performed by the entity. However, an entity may have properties (or attributes) associated with it. Moreover, the properties of the entity may change over time (for example, user may have new address, item may have new categories). In order to record such changes of an entity's properties. Special events `$set` , `$unset` and `$delete` are introduced.

The following special events are reserved for updating entities and their properties:

- `"$set"` event: Set properties of an entity (also implicitly create the entity). To change properties of entity, you simply set the corresponding properties with value again. The `$set` events should be created only when:
    - The entity is *first* created (or re-create after `$delete` event), or
    - Set the entity's existing or new properties to new values (For example, user updates his email, user adds a phone number, item has a updated categories)
- `"$unset"` event: Unset properties of an entity. It means treating the specified properties as not existing anymore. Note that the field `properties` cannot be empty for `$unset` event.
- `”$delete"` event: delete the entity.

There is no `targetEntityId` for these special events.

For example, setting entity `user-1`'s properties `birthday` and `address`:

```
{
  "event" : "$set",
  "entityType" : "user",
  "entityId" : "1",
  "properties" : {
    "birthday" : "1984-10-11",
    "address" : "1234 Street, San Francisco, CA 94107"
}
```

Note that the properties values of the entity will be aggregated based on these special events and the eventTime. The state of the entity is different depending on the time you are looking at the data.

Event 1

For example, on `2018-09-09T...`, a user with ID "2" is newly added in your application. Also, this user has properties a = 3 and b = 4. To record such event, we can create a `$set` event for the user.

```
$ curl -X POST https://events.recomendo.ai/v1/events.json \
-H "Content-Type: application/json" \
-H "Authorization: Bearer 
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjcwMTU2OTViMmQ0MTRhNjg4OTA1MzJiOTgzNWQ1NWNkIiw
iand0VmVyc2lvbiI6IjAwMDEiLCJpYXQiOjE1MzY0MTcxMDQsImV4cCI6MTUzNjQyMDcwNH0.njWpeGBjFRFXS2Do5gml6ItbFLO
6Ab0CDPQvfti0SGY" \
-d '{
  "event" : "$set",
  "entityType" : "user",
  "entityId" : "2",
  "properties" : {
    "a" : 3,
    "b" : 4
  },
  "eventTime" : "2018-09-09T16:17:42.937-08:00"
}'
```

You should see something like the following, meaning the events are imported successfully.

```
HTTP/1.1 201 Created
Server: spray-can/1.3.2
Date: Tue, 02 Jun 2018 23:13:58 GMT
Content-Type: application/json; charset=UTF-8
Content-Length: 57

{“eventId":"PVjOIP6AJ5PgsiGQW6pgswAAAUhc7EwZpCfSj5bS5yg"}
```

After this eventTime, user-2 is created and has properties of a = 3 and b = 4.

Event 2

Then, on `2018-09-10T...`, let's say the user has updated the properties b = 5 and c = 6. To record such property change, create another `$set` event. Run the following command:

```
$ curl -X POST https://events.recomendo.ai/v1/events.json \
-H "Content-Type: application/json" \
-H "Authorization: Bearer 
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjcwMTU2OTViMmQ0MTRhNjg4OTA1MzJiOTgzNWQ1NWNkIiw
iand0VmVyc2lvbiI6IjAwMDEiLCJpYXQiOjE1MzY0MTcxMDQsImV4cCI6MTUzNjQyMDcwNH0.njWpeGBjFRFXS2Do5gml6ItbFLO
6Ab0CDPQvfti0SGY" \
-d '{
  "event" : "$set",
  "entityType" : "user",
  "entityId" : "2",
  "properties" : {
    "b" : 5,
    "c" : 6
  },
  "eventTime" : "2018-09-10T13:12:04.937-08:00"
}’
```

After this eventTime, user-2 has properties of a = 3, b = 5 and c = 6. Note that property `b` is updated with latest value.

Event 3

Then, let's say on `2018-09-11T...`, the user's properties 'b' is removed for some reasons. To record such event, create `$unset` event for user-2 with properties b:

```
$ curl -X POST https://events.recomendo.ai/v1/events.json \
-H "Content-Type: application/json" \
-H "Authorization: Bearer 
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjcwMTU2OTViMmQ0MTRhNjg4OTA1MzJiOTgzNWQ1NWNkIiw
iand0VmVyc2lvbiI6IjAwMDEiLCJpYXQiOjE1MzY0MTcxMDQsImV4cCI6MTUzNjQyMDcwNH0.njWpeGBjFRFXS2Do5gml6ItbFLO
6Ab0CDPQvfti0SGY" \
-d '{
  "event" : "$unset",
  "entityType" : "user",
  "entityId" : "2",
  "properties" : {
    "b" : null
  },
  "eventTime" : "2018-09-11T14:17:42.456-08:00"
}’
```

After this eventTime, user-2 has properties of a = 3, and c = 6. Note that property `b` is removed.

Event 4

Then, on `2018-09-12T...`, the user is removed from the application data. To record such event, create `$delete` event:

```
$ curl -i -X POST https://events.recomendo.ai/v1/events.json \
-H "Content-Type: application/json" \
-H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjcwMTU2OTViMmQ0MTRhNjg4OTA1MzJiOTgzNWQ1NWNkIiw
iand0VmVyc2lvbiI6IjAwMDEiLCJpYXQiOjE1MzY0MTcxMDQsImV4cCI6MTUzNjQyMDcwNH0.njWpeGBjFRFXS2Do5gml6ItbFLO
6Ab0CDPQvfti0SGY" \
-d '{
  "event" : "$delete",
  "entityType" : "user",
  "entityId" : "2",
  "eventTime" : "2014-09-12T16:13:41.452-08:00"
}’
```

After this eventTime, user-2 is removed.

Event 5

Then, on `2018-09-13T...`, let's say we want to add back the user-2 into the application again for some reasons. To record such event, create `$set` event for user-2 with empty properties:

```
$ curl -i -X POST https://events.recomendo.ai/v1/events.json \
-H "Content-Type: application/json" \
-H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjcwMTU2OTViMmQ0MTRhNjg4OTA1MzJiOTgzNWQ1NWNkIiw
iand0VmVyc2lvbiI6IjAwMDEiLCJpYXQiOjE1MzY0MTcxMDQsImV4cCI6MTUzNjQyMDcwNH0.njWpeGBjFRFXS2Do5gml6ItbFLO
6Ab0CDPQvfti0SGY" \
-d '{
  "event" : "$set",
  "entityType" : "user",
  "entityId" : "2",
  "eventTime" : "2018-09-13T16:17:42.143-08:00"
}'
```

After this eventTime, user-2 is created again with empty properties.

Creating Your First Event

The following shows how one can create an event involving a single entity.

```
$ curl -i -X POST http://localhost:7070/events.json \
-H "Content-Type: application/json" \
-H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjcwMTU2OTViMmQ0MTRhNjg4OTA1MzJiOTgzNWQ1NWNkIiw
iand0VmVyc2lvbiI6IjAwMDEiLCJpYXQiOjE1MzY0MTcxMDQsImV4cCI6MTUzNjQyMDcwNH0.njWpeGBjFRFXS2Do5gml6ItbFLO
6Ab0CDPQvfti0SGY" \
-d '{
  "event" : "$set",
  "entityType" : "user",
  "entityId" : "uid",
  "properties" : {
    "prop1" : 1,
    "prop2" : "value2",
    "prop3" : [1, 2, 3],
    "prop4" : true,
    "prop5" : ["a", "b", "c"],
    "prop6" : 4.56
  }
  "eventTime" : "2017-12-13T21:39:45.618-07:00"
}'
```

For example, the following shows how one can create an event involving two entities (with `targetEntity`).

```
$ curl -i -X POST http://localhost:7070/events.json \
-H "Content-Type: application/json" \
-H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjcwMTU2OTViMmQ0MTRhNjg4OTA1MzJiOTgzNWQ1NWNkIiw
iand0VmVyc2lvbiI6IjAwMDEiLCJpYXQiOjE1MzY0MTcxMDQsImV4cCI6MTUzNjQyMDcwNH0.njWpeGBjFRFXS2Do5gml6ItbFLO
6Ab0CDPQvfti0SGY" \
-d '{
  "event" : "my_event",
  "entityType" : "user",
  "entityId" : "uid",
  "targetEntityType" : "item",
  "targetEntityId" : "iid",
  "properties" : {
    "someProperty" : "value1",
    "anotherProperty" : "value2"
  },
  "eventTime" : "2017-12-13T21:39:45.618Z"
}'
```

Sample response:

```
HTTP/1.1 201 Created
Server: spray-can/1.2.1
Date: Wed, 10 Sep 2017 22:51:33 GMT
Content-Type: application/json; charset=UTF-8
Content-Length: 41

{"eventId":"AAAABAAAAQDP3-jSlTMGVu0waj8"}
```

Note About Properties

Note that `properties` can be:

1. Associated with an *generic event*: The `properties` field provide additional information about this event

2. Associated with an *entity*: The `properties` field is used to record the changes of an entity's properties with special events `$set`, `$unset` and `$delete.`

Debugging Recipes

The following API are mainly for development or debugging purpose only. They should not be supported by SDK nor used by real application under normal circumstances and they are subject to changes.

Instead of using `curl`, you can also install JSON browser plugins such as **JSONView** to pretty-print the JSON on your browser. With the browser plugin you can make the `GET` queries below by passing in the URL. Plugins like **Postman - REST Client** provide a more advanced interface for making queries.

Replace `<your_eventId>` by a real one in the following:

Get an Event

```
$ curl -i -X GET -H "Authorization: Bearer 
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjcwMTU2OTViMmQ0MTRhNjg4OTA1MzJiOTgzNWQ1NWNkIiw
iand0VmVyc2lvbiI6IjAwMDEiLCJpYXQiOjE1MzY0MTcxMDQsImV4cCI6MTUzNjQyMDcwNH0.njWpeGBjFRFXS2Do5gml6ItbFLO
6Ab0CDPQvfti0SGY" \
https://events.recomendo.ai/v1/events/<your_eventId>.json
```

Delete an Event

```
$ curl -i -X DELETE -H "Authorization: Bearer 
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjcwMTU2OTViMmQ0MTRhNjg4OTA1MzJiOTgzNWQ1NWNkIiw
iand0VmVyc2lvbiI6IjAwMDEiLCJpYXQiOjE1MzY0MTcxMDQsImV4cCI6MTUzNjQyMDcwNH0.njWpeGBjFRFXS2Do5gml6ItbFLO
6Ab0CDPQvfti0SGY" \
https://events.recomendo.ai/v1/events/<your_eventId>.json
```

Get Events of an App

```
$ curl -i -X GET -H "Authorization: Bearer 
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjcwMTU2OTViMmQ0MTRhNjg4OTA1MzJiOTgzNWQ1NWNkIiw
iand0VmVyc2lvbiI6IjAwMDEiLCJpYXQiOjE1MzY0MTcxMDQsImV4cCI6MTUzNjQyMDcwNH0.njWpeGBjFRFXS2Do5gml6ItbFLO
6Ab0CDPQvfti0SGY" \
https://events.recomendo.ai/v1/events.json
```

By default, it returns at most 20 events. Use the `limit` parameter to specify how many events returned (see below). Use cautiously!

In addition, the following *optional* parameters are supported:

- `startTime`: time in ISO8601 format. Return events with `eventTime >= startTime`.
- `untilTime`: time in ISO8601 format. Return events with `eventTime < untilTime`.
- `entityType`: String. The entityType. Return events for this `entityType` only.
- `entityId`: String. The entityId. Return events for this `entityId` only.
- `event`: String. The event name. Return events with this name only.
- `targetEntityType`: String. The targetEntityType. Return events for this `targetEntityType` only.

- `targetEntityId`: String. The targetEntityId. Return events for this `targetEntityId` only.
- `limit`: Integer. The number of record events returned. Default is 20. -1 to get all.
- `reversed`: Boolean. **Must be used with both `entityType` and `entityId` specified**, returns events in reversed chronological order. Default is false.

If you are using `curl` with the `&` symbol, you should quote the entire URL by using single or double quotes.

Depending on the size of data, you may encounter timeout when querying with some of the above filters. Event server uses `entityType` and `entityId` as the key so any query without both `entityType` and `entityId` specified might result in a timeout.

For example, get all events of an app with `eventTime >= startTime`

1

```
$ curl -i -X GET -H "Authorization: Bearer 
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjcwMTU2OTViMmQ0MTRhNjg4OTA1MzJiOTgzNWQ1NWNkIiw
iand0VmVyc2lvbiI6IjAwMDEiLCJpYXQiOjE1MzY0MTcxMDQsImV4cCI6MTUzNjQyMDcwNH0.njWpeGBjFRFXS2Do5gml6ItbFLO
6Ab0CDPQvfti0SGY" \
"https://events.recomendo.ai/v1/events.json?startTime=<time in ISO8601 format>" 
```

For example, get all events of an app with `eventTime < untilTime`:

```
$ curl -i -X GET -H "Authorization: Bearer 
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjcwMTU2OTViMmQ0MTRhNjg4OTA1MzJiOTgzNWQ1NWNkIiw
iand0VmVyc2lvbiI6IjAwMDEiLCJpYXQiOjE1MzY0MTcxMDQsImV4cCI6MTUzNjQyMDcwNH0.njWpeGBjFRFXS2Do5gml6ItbFLO
6Ab0CDPQvfti0SGY" \
"https://events.recomendo.ai/v1/events.json?untilTime=<time in ISO8601 format>" 
```

For example, get all events of an app with `eventTime >= startTime` and `eventTime < untilTime`:

```
$ curl -i -X GET -H "Authorization: Bearer 
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjcwMTU2OTViMmQ0MTRhNjg4OTA1MzJiOTgzNWQ1NWNkIiw
iand0VmVyc2lvbiI6IjAwMDEiLCJpYXQiOjE1MzY0MTcxMDQsImV4cCI6MTUzNjQyMDcwNH0.njWpeGBjFRFXS2Do5gml6ItbFLO
6Ab0CDPQvfti0SGY" \
"https://events.recomendo.ai/v1/events.json?startTime=<time in ISO8601 format>&untilTime=<time in 
ISO8601 format>"
```

For example, get all events of a specific entity with `eventTime < untilTime`:

1

```
$ curl -i -X GET -H "Authorization: Bearer 
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbGllbnRJZCI6IjcwMTU2OTViMmQ0MTRhNjg4OTA1MzJiOTgzNWQ1NWNkIiw
iand0VmVyc2lvbiI6IjAwMDEiLCJpYXQiOjE1MzY0MTcxMDQsImV4cCI6MTUzNjQyMDcwNH0.njWpeGBjFRFXS2Do5gml6ItbFLO
6Ab0CDPQvfti0SGY" \
"https://events.recomendo.ai/v1/events.json?
&entityType=<your_entityType>&entityId=<your_entityId>&untilTime=<time in ISO801 format>"
```

Prediction Queries

The Recomendo API has a reasonable set of defaults so queries can be very simple or, when the need arises, very flexible. The configuration parameters control much of what is returned so

Simple Personalized Query

```
{
  "user": "xyz"
}
```

Simple Item Set / Shopping Cart Query

Technically this is to find the things missing in the itemSet. Another term for this is "Complimentary Purchase" but the idea applies to a wide variety of lists, like watch-lists, favorites, shopping carts, wish-lists, etc. To get the missing items you will need to train a model with itemSets. For instance items purchased at one time or the contents of a wishlist whenever it changes.

```
{
  "itemSet": ["item-1", "item-5", "item-300", "item-2", ...]
}
```

When using a model trained on itemSets this returns the "missing items" or things that go with the items in the list. This type of model and query combination returns "complimentary items". The list in the query is the current contents of the shopping cart, wishlist, etc.

The query can also be used with the typical model of indicators, in which case the results are more like, "what items are similar to the ones in the query". This may be useful since it is easy to get from an existing model but will not return items missing from the set /complimentary items.

Trending Items

An empty query returns only popular items. All returned scores will be 0 but the order will be based on relative popularity. Property-based biases for boosts and filters can also be applied as with any other query.

```
{

}
```

Full Query Parameters

Query fields determine what data is used to match when returning recommendations.

```
{
  "user": "xyz",
  "userBias": -maxFloat..maxFloat,
  "item": "53454543513",
  "itemBias": -maxFloat..maxFloat,
  "itemSet": ["cd53454543513", "lg1", "vf23423432", "af87634"],
  "itemSetBias": -maxFloat..maxFloat,
  "from": 0,
  "num": 4,
  "fields": [
    {
      "name": "fieldname"
      "values": ["fieldValue1", ...],
      "bias": -maxFloat..maxFloat
    },...
  ]
  "dateRange": {
    "name": "dateFieldname",
    "before": "2015-09-15T11:28:45.114-07:00",
    "after": "2015-08-15T11:28:45.114-07:00"
  },
  "currentDate": "2015-08-15T11:28:45.114-07:00",
  "blacklistItems": ["itemId1", "itemId2", ...]
  "returnSelf": true | false,
}
```

- **user**: optional, contains a unique id for the user. This may be a user not in the **training**: data, so a new or anonymous user who has an anonymous id. All user history captured in near realtime can be used to influence recommendations, there is no need to retrain to enable this.

- **userBias**: optional (use with great care), the amount to favor the user's history in making recommendations. The user may be anonymous as long as the id is unique from any authenticated user. This tells the recommender to return recommendations based on the user's event history. Used for personalized recommendations.

- **item**: optional, contains the unique item identifier

- **itemBias**: optional (use with great care), the amount to favor similar items in making recommendations. This tells the recommender to return items similar to this the item specified. Use for "people who liked this also liked these".

- **itemSet**: optional, contains a list of unique item identifiers

- **itemBias**: optional (use with great care), the amount to favor itemSets in making recommendations. Mixing itemSet queries with user and item queries is not recommended and it is difficult to predict what it will return in the final mixed results.

- **fields**: optional, array of fields values and biases to use in this query.

- **name** field name for metadata stored in the EventStore with $set and $unset events.

- **values** an array on one or more values to use in this query. The values will be looked for in the field name.

- **bias** will either boost the importance of this part of the query or use it as a filter. Positive biases are boosts any negative number will filter out any results that do not contain the values in the field name. .

- **from**: optional rank/position to start returning recommendations from. Used in pagination. The rank/position is 0 based, 0 being the highest rank/position. Default: 0, since 0 is the first or top recommendation. Unless you are paginating skip this param.

- **num**: optional max number of recommendations to return. There is no guarantee that this number will be returned for every query.

- **blacklistItems**: optional. It can be used to remove duplicates when items are already shown in a specific context. This is called anti-flood in recommender use.

- **dateRange** optional, default is not range filter. One of the bound can be omitted but not both. Values for the `before` and `after` are strings in ISO 8601 format. Overrides the **currentDate** if both are in the query.

- **currentDate** optional, must be specified if used. If **dateRange** is included then **currentDate** is ignored.

- **returnSelf**: optional boolean asking to include the item that was part of the query (if there was one) as part of the results. Defaults to false.

- Defaults are either noted or taken from algorithm values, which themselves may have defaults. This allows very simple queries for the simple, most used cases.

- The query returns personalized recommendations, similar items, or a mix including backfill. The query itself determines this by supplying item, user, both, or neither. Some examples are:

Contextual Personalized

This returns items based on user "xyz" history filtered by categories and boosted to favor more genre specific items. The values for fields have been attached to items with $set events where the "name" corresponds to a doc field and the "values" correspond to the contents of the field. The "bias" is used to indicate a filter or a boost.

```
{
  "user": "xyz",
  "fields": [
    {
      "name": "categories"
      "values": ["series", "mini-series"],
      "bias": -1 // filter out all except 'series' or 'mini-series'
    },{
      "name": "genre",
      "values": ["sci-fi", "detective"]
      "bias": 1.5 // boost/favor recommendations with the 'genre' = 'sci-fi' or 'detective'
    }
  ]
}
```

Date Ranges As Query Filters

When the a date is stored in the items properties it can be used in a date range query. This is most often used by the app server since it may know what the range is, while a client query may only know the current date and so use the "Current Date" filter below.

```
{
  "user": "xyz", 
  "fields": [
    {
      "name": "categories"
      "values": ["series", "mini-series"],
      "bias": -1 }// filter out all except 'series' or 'mini-series'
    },{
      "name": "genre",
      "values": ["sci-fi", "detective"]
      "bias": 1.5 // boost/favor recommendations with the 'genre' = 'sci-fi' or 'detective'
    }
  ],
  "dateRange": {
    "name": "availableDate",
    "before": "2015-08-15T11:28:45.114-07:00",
    "after": "2015-08-20T11:28:45.114-07:00
  }
}
```

Items are assumed to have a field of the same `name` that has a date associated with it using a `$set` event. The query will return only those recommendations where the date field is in range. Either date bound can be omitted for a on-sided range. The range applies to all returned recommendations, even those for popular items.

Contextual Personalized With Similar Items

```
{
  "user": "xyz", 
  "userBias": 2, // favor personal recommendations
  "item": "53454543513", // fallback to contextual recommendations
  "fields": [
    {
      "name": "categories"
      "values": ["series", "mini-series"],
      "bias": -1 }// filter out all except 'series' or 'mini-series'
    },{
      "name": "genre",
      "values": ["sci-fi", "detective"]
      "bias": 1.02 // boost/favor recommendations with the 'genre' = 'sci-fi' or 'detective'
    }
  ]
}
```

This returns items based on user *xyz* history or similar to item 53454543513 but favoring user history recommendations. These are filtered by categories and boosted to favor more genre specific items.

Note:This query should be considered **experimental**. mixing user history with item similarity is possible but may have unexpected results. If you use this you should realize that user and item recommendations may be quite divergent and so mixing the them in query may produce nonsense.