PMT Database Function Documentation

The PMT Database contains a large number of functions that support various application & database functionality. This documentation contains a description of the function, its parameters and examples for usage. Please report any errors to the issues page.

Function Listing

pmt_clone_activity

pmt_contacts

pmt_countries

pmt_create_user **DEPRECATED Interation 8 and below: (see pmt_edit_user)**

pmt_data_groups **Updated for v10**

pmt_edit_activity

pmt_edit_activity_contact

pmt_edit_activity_taxonomy

pmt_edit_contact

pmt_edit_detail

pmt_edit_financial

pmt_edit_financial_taxonomy

pmt_edit_location

pmt_edit_location_taxonomy

pmt_edit_organization

pmt_edit_participation

pmt_edit_participation_taxonomy

pmt_edit_user **Updated for v10**

pmt_export **Updated for v10 (renamed from pmt_filter_csv)**

pmt_filter **New for v10**

pmt_filter_iati

pmt_filter_string **New for v10**

pmt_global_search **Updated for v10**

pmt_iati_import **Updated for v10**

pmt_is_data_group **New for v10**

pmt_isdate

pmt_isnumeric

pmt_locations **Updated for v10**

pmt_locations_for_boundaries **New for v10**

pmt_org_inuse **Updated for v10**

pmt_orgs **Updated for v10**

pmt_overview_stats **New for v10**

pmt_partner_pivot **New for v10**

pmt_partner_sankey **Updated for v10**

pmt_partner_sankey_activities **Updated for v10**

pmt_purge_activities **New for v10**

pmt_purge_activity **Updated for v10**

pmt_stat_activity_by_tax **Updated for v10**

pmt_stat_by_org **New for v10**

pmt_stat_invest_by_funder **New for v10**

pmt_statistic_data **New for v10**

pmt_statistic_indicators **New for v10**

pmt_tax_inuse

pmt_taxonomies

pmt_update_crosswalks

pmt_user_auth **Updated for v10**

pmt_user_salt **Updated for v10**

pmt_user

pmt_users **Updated for v10**

pmt_validate_activities **Updated for v10**

pmt_validate_activity

pmt_validate_boundary_feature **Updated for v10**

pmt_validate_classification **Updated for v10**

pmt_validate_classifications **Updated for v10**

pmt_validate_contact

pmt_validate_contacts

pmt_validate_detail

pmt_validate_financial

pmt_validate_location **Updated for v10**

pmt_validate_locations **Updated for v10**

pmt_validate_organization **Updated for v10**

pmt_validate_organizations

pmt_validate_participation

pmt_validate_participations

pmt_validate_role

pmt_validate_taxonomies

pmt_validate_taxonomy **Updated for v10**

pmt_validate_user

pmt_validate_user_authority **Updated for v10**

pmt_version **Updated for v10**

test_execute_unit_tests **New for v10**

---

pmt_2x2

Description

Returns 2x2 data for a requested country & region.

Parameter(s)

1. country (character varying) - **Required** the country name.
2. region (character varying) - **Required** the region name within the country for the 2x2 data.

Result

Json with the following:

1. country (character varying) – name of the country
2. regions (character varying[]) - array of region names with in the country that have 2x2 data

Example(s)

- Get all participating country & regions for the 2x2:

```
SELECT * FROM pmt_2x2_regions();
```


```
{
    "order":0,
    "category":"Low-Low",
    "districts":"Guji, Horo Guduru, Mirab Hararghe, Ilubabor, Jimma, Kelem
Wellega, Bale, Mirab Shewa, Mirab Welega, Misraq Harerge, Misraq Wellega,
North Shewa",
    "area":192455.44,
    "pop":22218885.238295598,
    "popden":115.4495047700163633,
    "povden":0.40458716053960334922
},
{
    "order":1,
    "category":"Low-Hi",
    "districts":"",
    "area":0,
    "pop":0,
    "popden":0,
    "povden":0
},
{
    "order":2,
    "category":"Hi-Low",
    "districts":"Debub Mirab Shewa, Arsi, Mirab Arsi, Borena",
    "area":83545.20,
    "pop":8065437.75257711,
    "popden":96.5398102174285297,
    "povden":0.38057243264723766297
}
{
    "order":3,
    "category":"Hi-Hi",
    "districts":"Misraq Shewa",
    "area":8370.9,
    "pop":1956499.81554151,
    "popden":233.7263395263962059,
    "povden":0.93239675542653717044
}
{
    "order":4,
    "category":"n/a",
    "districts":"",
    "area":0,
    "pop":0,
    "popden":0,
```

```
    "povden":0
}
```

↩ Back to Function List

pmt_2x2_regions

Description

Returns participating country & regions for the 2x2 tool.

Parameter(s)

No parameters

Result

Json with the following:

1. id (integer) – id of the gadm0 feature representing the country
2. country (character varying) – name of the country
3. extent (wkt) – polygon extent of the country
4. regions (json[]) - array of json objects
    1. id (integer) – id of the gadm1 feature representing the region within the country
    2. _name (character varying) - name of the region

Example(s)

- Get all participating country & regions for the 2x2:

```
SELECT * FROM pmt_2x2_regions();
```

```
"{"id":74,"_name":"Ethiopia","extent":"POLYGON((33.0015373229982
3.39882302284263,33.0015373229982 14.8454771041872,47.9582290649417
14.8454771041872,47.9582290649417 3.39882302284263,33.0015373229982
3.39882302284263))","regions":[{"id":15174,"_name":"Addi (...)"
{
    "id":74,
    "country":"Ethiopia",
    "extent":"POLYGON((33.0015373229982 3.39882302284263,33.0015373229982
14.8454771041872,47.9582290649417 14.8454771041872,47.9582290649417
3.39882302284263,33.0015373229982 3.39882302284263))"
    "regions":[
        {
            "id":889,
            "_name":"Addis Abeba"
        },
        {
            "id":890,
            "_name":"Afar"
```

```
        },
        ...
        {
            "id":899,
            "_name":"Tigray"
        }
    ]
},
{
    "id":227,
    "country":"Tanzania",
    "extent":"POLYGON((29.3271675109864 -11.7456951141355,29.3271675109864 -
0.985787510871774,40.445137023926 -0.985787510871774,40.445137023926 -
11.7456951141355,29.3271675109864 -11.7456951141355))"
    "regions":[
        {
            "id":2998,
            "_name":"Arusha"
        },
        {
            "id":2999,
            "_name":"Dar es Salaam"
        },
        {
            "id":3000,
            "_name":"Dodoma"
        },
        ...
        {
            "id":3027,
            "_name":"Zanzibar West"
        }
    ]
}
...
```


↩ Back to Function List

pmt_activate_activity ==========

Description

Activate/deactivate an activity and its related records (locations, financial, participation, detail, result).

Parameter(s)

1. user_id (integer) – **Required**. user_id of user requesting edit.
2. activity_id (integer) – **Required**. activity_id to activate/deactivate.
3. activate (boolean) - **Default is TRUE**. True to activate, false to deactivate.

Result

Json with the following:

1. id (integer) – activity_id of the activity activated/deactivated.
2. message(character varying) - Message containing either "Success" for a successful transaction or containing an error description regarding the unsuccessful transaction. Some possible error messages:
   - Must include user_id and activity_id data parameters
   - User does NOT have authority to change the active status of this activity and its associated records.

Example(s)

- Activate the activity_id 15820 and its related records.

```
select * from pmt_activate_activity(34, 15820, true);
```

```
{"id":15820,"message":"Success"}
```

- Deactivate the activity_id 15820 and its related records.

```
select * from pmt_activate_activity(34, 15820, false);
```

```
{"id":15820,"message":"Success"}
```

pmt_activities

Description

Get a filterable list of activities.

Parameter(s)

1. data_group_ids (character varying) - comma seperated list of classification id(s) from the Data Group taxonomy to restrict data. If no data group id is provided, all data groups are included.
2. classification_ids (character varying) - comma seperated list of classification id(s) for any taxonomy (filter).
3. org_ids (character varying) - comma seperated list of organization id(s) for organizations for all participation types (filter).
4. imp_org_ids (character varying) - comma seperated list of organization id(s) for implementing organizations (filter).
5. fund_org_ids (character varying) - comma seperated list of organization id(s) for funding organizations (filter).
6. start_date (date) - start date for activities (filter).
7. end_date (date) - end date for activities (filter).
8. unassigned_taxonomy_ids (character varying) - comma seperated list of taxonomy id(s) for any taxonomy, will return activities that *DO NOT* have that taxonomy assigned (filter).

9. activity_ids (character varying) - comma seperated list of activity id(s) to restrict data aggregation to.
10. boundary_filter (json) - a json array of objects. Each object must contain "b" with a boundary id and "ids" with an array of feature ids (i.e. `[{"b":12,"ids":[2,3]},{"b":13,"ids":[73,85]}]`)

Result

Json with the following:

1. id (integer) – the activity id.
2. pid (integer) – the activity id of the parent activity.
3. dgid (integer) – the data group id for the activity.
4. dg (character varying) – the name of the data group for the activity
5. t (character varying) – title of activity.
6. a (numeric) – total investment amount for the activity.
7. sd (date) - the activity start date.
8. ed (date) - the activity end date.
9. f (character varying[]) - array of funding organizations for the activity.

Example(s)

- All activities for BMGF (data_group_id: 768) that have an Activity Status (taxonomy id: 18) of "Complete" (classification id: 797) or do not have an Activity Status (taxonomy id: 18) assigned between 1/1/2005 and 1/1/2020:

```
select * from
pmt_activities('768','797',null,null,null,'1/1/2005','1/1/2020','18',null);

...
{
    "id":1841,
    "pid":23728,
    "dgid":768,
    "dg":"BMGF",
    "t":"Tanzania Tropical Pesticides Research Institute",
    "a":200000.00,
    "sd":"2008-02-01",
    "ed":"2013-01-31",
    "f":{
        "Bill & Melinda Gates Foundation (BMGF)"
    }
},
{
    "id":1842,
    "pid":23546,
    "dgid":768,
    "dg":"BMGF",
    "t":"Quarterly review meeting",
```

```
    "a":null
    "sd":"2011-11-01",
    "ed":"2016-11-30",
    "f":{null}
},
{
    "id":1843,
    "pid":23546,
    "dgid":768,
    "dg":"BMGF",
    "t":"Equip women farmers and collective members with life skills",
    "a":null,
    "sd":"2011-11-01",
    "ed":"2016-11-30",
    "f":{null}
}
...
```


↩ Back to Function List

pmt_activities_by_polygon

Description

Select activities within a given polygon.

Parameter(s)

1. wktpolygon (text) – **Required**. Well-known text representation of a polygon.
2. data_group_ids (character varying) - comma seperated list of classification_id(s) from the Data Group taxonomy to restrict data aggregation to. If no data group id is provided, all data groups are included.
3. classification_ids (character varying) - comma seperated list of classification_id(s) for any taxonomy (filter).
4. org_ids (character varying) - comma seperated list of organization_id(s) for any organization regardless of role (filter).
5. imp_org_ids (character varying) - comma seperated list of organization_id(s) for implementing organizations (filter).
6. fund_org_ids (character varying) - comma seperated list of organization_id(s) for funding organizations (filter).
7. start_date (date) - start date for activities (filter).
8. end_date (date) - end date for activities (filter).
9. unassigned_taxonomy_ids (character varying) - comma seperated list of taxonomy id(s) for any taxonomy, will return activities that *DO NOT* have that taxonomy assigned (filter).
10. activity_ids (character varying) - comma seperated list of activity id(s) to restrict data aggregation to.

11. boundary_filter (json) - a json array of objects. Each object must contain "b" with a boundary id and "ids" with an array of feature ids (i.e. `[{"b":12,"ids":[2,3]},{"b":13,"ids":[73,85]}]`)

Result

Json with the following:

1. activity_ids (integer[]) – integer array of activity ids.

Example(s)

- Get activities for BMGF & AGRA (data_groups_id: 768,769) between 2001 and 2021 within a given polygon:

```
select * from pmt_activities_by_polygon('POLYGON((-16.473 13.522,-16.469
13.186,-16.764 13.185,-16.797 13.491,-16.472 13.517,-16.473
13.522))','768,769',null,null,null,null,'01-01-2001','12-31-2021',null,
null,null);
```

```
{
    "activity_ids":[14893,14895]
}
```

↩ Back to Function List

pmt_activity

Description

All information for a single activity.

Parameter(s)

1. activity_id (integer) - **Required.** Activity id.

Result

Json with the following:

1. id (integer) – activity id.
2. data_group_id (integer) – the data group id.
3. parent_id (integer) – the activity's parent id (if null the activity is a parent).
4. _title (character varying) – title of activity.
5. _label (character varying) – short title for activity.
6. _description (character varying) – description of activity.
7. _objective (character varying) – the objective for activity.
8. _content (character varying) – various content for activity.
9. _url (character varying) – url for activity.
10. _start_date (date) – start date of activity.
11. _planned_start_date (date) – planned start date of activity.

12. _planned_end_date (date) – end date of activity.
13. _end_date (date) – planned end date of activity.
14. _tags (character varying) – tags or keywords of activity.
15. _iati_identifier (integer) – iati identifier or primary key of activity.
16. _iati_import_id (integer) – iati import id for the corresponding import record (if null the data was manually loaded).
17. _updated_by (character varying(50)) - last user to update activity information.
18. _updated_date (timestamp) - last date and time activity information was updated.
19. custom_fields (various) - any custom fields in the activity table that are not in the Core PMT will be returned as well.
20. data_group (character varying) – the data group name.
21. parent_title (character varying) – the activity's parent activity title.
22. ct (integer) - number of locations for activity.
23. taxonomy(object) - An object containing all associated taxonomy for the activity
    1. taxonomy_id (integer) - taxonomy id.
    2. taxonomy (character varying) - taxonomy name.
    3. classification_id (integer) - classification id.
    4. classification (character varying) - classification name.
    5. code (character varying) - classification code.
24. organizations(object) - An object containing all organizations participating in the activity
    1. organization (character varying) - organization name
    2. role (character varying) - the organization's role in the activity (Implementing, Funding, Accountable, Extending)
    3. url (character varying) - the url for the organization.
    4. address (character varying) - the organization's address.
    5. city (character varying) - the city of the organization.
    6. state_providence (character varying) - the state or providence of the organization.
    7. zip (character varying) - the zip code for the providence.
    8. country (character varying) - the country of the organization.
25. contacts (object) - An object containing all activity contacts.
    1. id (integer) - the contact id.
    2. _first_name (character varying) - contact's first name.
    3. _last_name (character varying) - contact's last name.
    4. _email (character varying) - contact's email address.
    5. organization_id (integer) - organization id.
    6. _name (character varying) - organization name the contact is associated with.
26. details (object) - An object containing all activity details.
    1. detail_id (integer) - detail id.
    2. _title (character varying) - the title of the detail.

3. _description (character varying) - description of the detail.
4. _amount (numeric (12,2)) - detail amount.

27. financials (object) - An object containing all activity financial data.
    1. id (integer) - financial id.
    2. _amount (numeric (100,2)) - financial amount.
    3. _start_date (date) – start date for financial amount.
    4. _end_date (date) – end date for financial amount.
    5. provider (character varying) - the name of the organization providing the financial amount.
    6. recipient (character varying) - the name of the organization receiving the financial amount.
    7. taxonomy(object) - An object containing all associated taxonomy for the financial record
        1. taxonomy_id (integer) - taxonomy id.
        2. taxonomy (character varying) - taxonomy name.
        3. classification_id (integer) - classification id.
        4. classification (character varying) - classification name.
        5. code (character varying) - classification code.
28. location_ids (int[]) - An array of location_ids associated to the activity.
29. locations (object) - An object containing all activity location data.
    1. id (integer) - location id.
    2. _admin1 (character varying) - the name of the administrative level 1 boundray for the location.
    3. _admin2 (character varying) - the name of the administrative level 2 boundray for the location.
    4. _admin3 (character varying) - the name of the administrative level 3 boundray for the location.
    5. taxonomy(object) - An object containing all associated taxonomy for the financial record
        1. taxonomy_id (integer) - taxonomy id.
        2. taxonomy (character varying) - taxonomy name.
        3. classification_id (integer) - classification id.
        4. classification (character varying) - classification name.
        5. code (character varying) - classification code.
    6. boundaries(object) - An object containing all intersected boundaries for the activity's locations
        1. boundary_id (integer) - boundary id.
        2. feature_id (integer) - feature id of boundary.
        3. _feature_name (character varying) - the name of the feature.
30. children (object) - An object containing all activity's children activities.

1. id (integer) - child activity id.
2. _title (character varying) - child activity title.

Example(s)

```
select * from pmt_activity(3);
```


```
{
    "activity_id":3
    ,"project_id":1
    ,"title":"UN collaborative Program for Reducing Emissions from
Deforestation and forest Degradation in developing
            countries, Tanzania (UN-REDD)"
    ,"label":null
    ,"description":"The project aims at strengthening Tanzania s readiness
for Reducing Emissions from Deforestation and
            forest Degradation (REDD) as a component of the Government s
evolving REDD Strategy, and integrate it
            with other REDD activities in the country"
    ,"content":null
    ,"start_date":"2009-10-01"
    ,"end_date":"2013-12-31"
    ,"tags":null
    ,"updated_by":"IATI XML Import"
    ,"updated_date":"2014-01-16 00:00:00"
    ,"iati_identifier":null
    ,"location_ct":1
    ,"admin_bnds":"United Republic of Tanzania,Singida,Manyoni"
    ,"taxonomy":[{
         "taxonomy_id":5
        ,"taxonomy":"Country"
        ,"classification_id":244
        ,"classification":"TANZANIA, UNITED REPUBLIC OF"
        ,"code":"TZ"
        }
        ,{
         "taxonomy_id":14
        ,"taxonomy":"Sector Category"
        ,"classification_id":552
        ,"classification":"Other multisector"
        ,"code":"430"
        }
        ,{
         "taxonomy_id":15
        ,"taxonomy":"Sector"
        ,"classification_id":729
        ,"classification":"Multisector aid"
        ,"code":"43010"
        }
        ,{
```

```
     "taxonomy_id":17
    ,"taxonomy":"Category"
    ,"classification_id":779
    ,"classification":"Training and Capacity Building"
    ,"code":null
    }
    ,{
     "taxonomy_id":18
    ,"taxonomy":"Sub-Category"
    ,"classification_id":792
    ,"classification":"Training and Capacity Building"
    ,"code":null
    }]
,"organizations":[{
     "organization_id":1
    ,"name":"FAO/ UNEP/ UNDP"
    ,"url":null
    ,"taxonomy":[{
        "taxonomy_id":10
        ,"taxonomy":"Organisation Role"
        ,"classification_id":496
        ,"classification":"Funding"
        ,"code":"Funding"
        }]
    }
    ,{
    "organization_id":56
    ,"name":"Tanzania Forestry Service (TFS)"
    ,"url":null
    ,"taxonomy":[{
        "taxonomy_id":10
        ,"taxonomy":"Organisation Role"
        ,"classification_id":497
        ,"classification":"Implementing"
        ,"code":"Implementing"
        }]
    }
    ,{
    "organization_id":2
    ,"name":"MNRT, TFS"
    ,"taxonomy":[{
        "taxonomy_id":10
        ,"taxonomy":"Organisation Role"
        ,"classification_id":497
        ,"classification":"Implementing"
        ,"code":"Implementing"
        }]
    }]
,"contacts":[{
    "contact_id":1
```

```
        ,"first_name":"Almas"
        ,"last_name":"Kashindye"
        ,"email":"Almas.Kashindye@fao.org"
        ,"organization_id":1
        ,"name":"FAO/ UNEP/ UNDP"
        }]
    ,"details": null
    ,"financials":[{
         "financial_id":13
        ,"amount":814972.00
        ,"taxonomy":[{
            "taxonomy_id":6
            ,"taxonomy":"Currency"
            ,"classification_id":422
            ,"classification":"US Dollar"
            ,"code":"USD"
            }]
        }]
    ,"locations":[{3}]
}
```


↩ Back to Function List

pmt_activity_by_invest

Description

Returns the top x number of activities, ordered by greatest investment amount.

Parameter(s)

1. data_group_ids (character varying) - comma seperated list of classification id(s) from the Data Group taxonomy to restrict data. If no data group id is provided, all data groups are included.
2. classification_ids (character varying) - comma seperated list of classification id(s) for any taxonomy (filter).
3. start_date (date) - start date for activities (filter).
4. end_date (date) - end date for activities (filter).
5. boundary_id (integer) - the boundary id referenced by the feature_id (filter).
6. feature_id (integer) - the featuer id to restrict activities to (filter).
7. limit_records (integer) - number of records to return.
8. field_list (character varying) - list of additional activity fields to return. Example: 'opportunity_id,_description'

Result

Json with the following:

1. id (integer) – the activity id.

2. title (character varying) – the activity title.
3. amount (numeric) – the investment amount for the activity.
4. fund (character varying) - comma seperated listing of funding organization names for the activity.
5. imp (character varying) - comma seperated listing of implementing organization names for the activity.
6. acct (character varying) - comma seperated listing of accountable organization names for the activity.
7. additional requested fields listed in the "field_list" option.

Example(s)

- Top 5 activities by investment, including the opportunity_id field for BMGF data group (data_group_id:768) in Ethiopia (feature_id: 74) using GADM 0 boundary (boundary_id: 15):

```
SELECT * FROM
pmt_activity_by_invest('768',null,null,null,15,74,5,'opportunity_id');
```


```
{
    "id":24905,
    "_title":"AGRA Soil Health Program",
    "amount":148135881.00,
    "fund":"Bill & Melinda Gates Foundation (BMGF)",
    "imp":"Alliance for a Green Revolution in Africa (AGRA)",
    "acct":null,
    "opportunity_id":"OPP48790"
},
...
{
    "id":25130,
    "_title":"Renewal: STRASA Phase 3 - Stress-Tolerant Rice for Africa and
South Asia",
    "amount":32770000.00,
    "fund":"Bill & Melinda Gates Foundation (BMGF)",
    "imp":"International Rice Research Institute (IRRI)",
    "acct":null,
    "opportunity_id":"OPP10888"
}
```


↩ Back to Function List

pmt_activity_count

Description

Returns count of parent activities for a filter.

Parameter(s)

1. data_group_ids (character varying) - comma seperated list of classification_id(s) from the Data Group taxonomy to restrict data aggregation to. If no data group id is provided, all data groups are included.
2. classification_ids (character varying) - comma seperated list of classification_id(s) for any taxonomy (filter).
3. org_ids (character varying) - comma seperated list of organization_id(s) for any organization regardless of role (filter).
4. imp_org_ids (character varying) - comma seperated list of organization_id(s) for implementing organizations (filter).
5. fund_org_ids (character varying) - comma seperated list of organization_id(s) for funding organizations (filter).
6. start_date (date) - start date for activities (filter).
7. end_date (date) - end date for activities (filter).
8. unassigned_taxonomy_ids (character varying) - comma seperated list of taxonomy id(s) for any taxonomy, will return activities that *DO NOT* have that taxonomy assigned (filter).
9. activity_ids (character varying) - comma seperated list of activity id(s) to restrict data aggregation to.
10. boundary_filter (json) - a json array of objects. Each object must contain "b" with a boundary id and "ids" with an array of feature ids (i.e. `[{"b":12,"ids":[2,3]},{"b":13,"ids":[73,85]}]`)

Result

Json with the following:

1. ct(integer) – total number of activities for a given filter.

Example(s)

- Number of activities for BMGF data (data_group_id: 768) where Activity Status is Complete (classification_id: 797) and activities occur between 1/1/2012 and 12/31/2018:

```
SELECT * FROM
pmt_activity_count('768','797',null,null,'1/1/2012','12/31/2018',null);
```

```
{
    "ct":67
}
```

- Number of activities for BMGF data (data_group_id: 768) where Activity Status is Complete (classification_id: 797) and activities occur between 1/1/2012 and 12/31/2018:

```
SELECT * FROM
pmt_activity_count('768','797',null,null,null,'1/1/2012','12/31/2018',null,nu
ll,null);
```

```
{
    "ct":67
}
```

↩ Back to Function List

pmt_activity_count_by_participants

Description

Returns count of activities by organization for a specific organization role.

Parameter(s)

1. classification_id (integer) - **Required** classification id of the Organisation Role taxonomy.
2. activity_ids (character varying) - **Required** comma seperated list of activity id(s) to be counted per organization.

Result

Json with the following:

1. organizations(object[]) – array of objects containing organizations with specified Organisation Role.
    1. name (character varying) - name of organization.
    2. activity_ct (integer) - number of activities organization is participating in from provided list of activity ids.

Example(s)

- Number of activities each Implmenting (classification_id: 497) organization is participating in.

```
SELECT * FROM pmt_activity_count_by_participants(497, '1767,3188');
```

```
{
    "organizations":[
        {
            "name":"Local Government",
            "activity_ct":1
        },
        {
            "name":"Ministry of Agriculture Food Security and
Cooperatives (MAFC)",
            "activity_ct":1
        },
        {
```

```
            "name":"Private Sector Players",
            "activity_ct":1
        }
    ]
}
```

↩ Back to Function List

pmt_activity_count_by_taxonomy

Description

Returns count of activities within a list of activity ids for a specified taxonomy.

Parameter(s)

1. tax_id (integer) - **Required** taxonomy id for count of activities that have a classification assignment.
2. activity_ids (character varying) - **Required** comma seperated list of activity id(s) to be counted for association to specified taxonomy.

Result

Json with the following:

1. taxonomy (character varying) – the name of the requested taxonomy.
2. classification (character varying) – the name of the classification within the requested taxonomy.
3. activity_ct (integer) - the number of activities that have the classification assignment within the provided list of activities.

Example(s)

- Number of activities in list that are assigned to the Initiative taxonomy (taxonomy_id: 23).

```
SELECT * FROM pmt_activity_count_by_taxonomy(23, '19030,14893,14895,24268');
```

```
{
    "taxonomy":"Initiative",
    "classification":"Research & Development",
    "activity_ct":2
}
```

↩ Back to Function List

pmt_activity_details ==========

Description

Activity details for a single activity.

Parameter(s)

1. a_id (integer) – **Required**. activity_id.

Result

Json with the following:

1. a_id (integer) – activity_id.
2. title (character varying) – title of activity.
3. desc (character varying) – description of activity.
4. start_date (date) – start date of activity.
5. end_date (date) – end date of activity.
6. tags (character varying) – tags of activity.
7. amount (integer) - total amount of activity.
8. taxonomy (object) – object containing taxonomy/classification for all related taxonomies
    1. taxonomy (character varying) – name of taxonomy
    2. classification (character varying) – name of classification
    3. org (character varying) – name of organization (provided only where taxonomy is *Organisation Role*)
9. locations (object) – object containing all related locations.
    1. location_id (integer) - location_id of location
    2. gaul0_name (character varying) - name of GAUL 0 administrative boundary
    3. gaul1_name (character varying) - name of GAUL 1 administrative boundary
    4. gaul2_name (character varying) - name of GAUL 2 administrative boundary
    5. lat (decimal degrees) - latitude of location
    6. long (decimal degrees) - longitude of location

Example(s)

- Activity_id 2039:

```
SELECT * FROM  pmt_activity_details(2039);
```


```
{   "a_id":2039
    ,"title":"ASDP"
    ,"desc":"repair of hides and skin banda"
    ,"start_date":"2011-01-01"
    ,"end_date":"2012-12-31"
    ,"tags":null
    ,"amount":3500000.00
    ,"taxonomy":
        [{  "taxonomy":"Country"
            ,"classification":"TANZANIA, UNITED REPUBLIC OF"
            ,"org":null
         }
        ,{ "taxonomy":"Data Group"
```

```
            ,"classification":"ASDP"
            ,"org":null
          }
         ,{ "taxonomy":"Category"
            ,"classification":"Post Harvest"
            ,"org":null
          }
         ,{ "taxonomy":"Organisation Role"
            ,"classification":"Implementing"
            ,"org":"LGAs"
          }
         ,{ "taxonomy":"Organisation Role"
            ,"classification":"Funding"
            ,"org":"DLDF (District Local Development Fund)"
          }
         ,{ "taxonomy":"Sector"
            ,"classification":"Plant and post-harvest protection and pest
control"
            ,"org":null
          }
         ,{ "taxonomy":"Sector Category"
            ,"classification":"AGRICULTURE"
            ,"org":null
          }
         ,{ "taxonomy":"Sub-Category"
            ,"classification":"Commodity Value Chain"
            ,"org":null
          }]
    ,"locations":
          [{    "location_id":2039
            ,"gaul0_name":"United Republic of Tanzania"
            ,"gaul1_name":"Shinyanga"
            ,"gaul2_name":"Kahama"
            ,"lat":-3.950000
            ,"long":32.03333
          }]
}
```


pmt_activity_ids_by_boundary

Description

Returns parent activity ids and titles for a given feature and boundary.

Parameter(s)

1. boundary_id (integer) - **Required**. boundary id of the boundary layer that the feature belongs to.
2. feature_id (integer) - **Required**. feature id of the feature in which to query activities for.

3. data_group_ids (character varying) - comma seperated list of classification_id(s) from the Data Group taxonomy to restrict data aggregation to. If no data group id is provided, all data groups are included.
4. classification_ids (character varying) - comma seperated list of classification_id(s) for any taxonomy (filter).
5. org_ids (character varying) - comma seperated list of organization_id(s) for any organization regardless of role (filter).
6. imp_org_ids (character varying) - comma seperated list of organization_id(s) for implementing organizations (filter).
7. fund_org_ids (character varying) - comma seperated list of organization_id(s) for funding organizations (filter).
8. start_date (date) - start date for activities (filter).
9. end_date (date) - end date for activities (filter).
10. unassigned_taxonomy_ids (character varying) - comma seperated list of taxonomy id(s) for any taxonomy, will return activities that *DO NOT* have that taxonomy assigned (filter).
11. activity_ids (character varying) - comma seperated list of activity id(s) to restrict data aggregation to.
12. boundary_filter (json) - a json array of objects. Each object must contain "b" with a boundary id and "ids" with an array of feature ids (i.e. `[{"b":12,"ids":[2,3]},{"b":13,"ids":[73,85]}]`)

Result

Json array with the following:

1. id (integer) – id of activity
2. _title (character varying) - the activity title

Example(s)

- Activity information for Africa (feature id: 4) in the continent boundary (boundary id: 8) for BMGF data (data_group_id: 768) where Focus Crop is Maize or Cassava (classification ids: 816,819) OR the activity has **NO** Focus Crops (taxonomy id: 22) and activities occur between 1/1/2012 and 12/31/2018:

```
SELECT * FROM pmt_activity_ids_by_boundary(8,
4,'768','816,819','','','1/1/2012','12/31/2018','22',null);

...
{
    "id":15795,
    "_title":"OLAM"
},
{
    "id":15796,
    "_title":"Cocoa Research Institute of Ghana"
},
```

```
{   "id":15797,
    "_title":"Ministry of Food & AGRIC, Ghana"
}
...
```


- Activity information for Gambela (feature_id: 6) in the to the UNOCHA administrative level 1 boundary (boundary_id: 12) for RED&FS data (data_group_id: 2237) where Program is "Stand Alone Project" (classification_id: 2239) and activities occur between 1/1/2002 and 12/31/2020 within the Oromia region (boundary_id: 12 feature_id: 8) or in the Majang zone (boundary_id: 13 feature_id: 38) within the Gambela region :

```
SELECT * FROM pmt_activity_ids_by_boundary(12,
6,'2237','2239',null,null,'1/1/2002','12/31/2020',null,'[{"b":12,"ids":[8]},{
"b":13,"ids":[38]}]');
```


```
{
    "id":26234,
    "_title":"UNICEF-Accelarating Progress MDGS on W&S Ethiopia"
},
{
    "id":26283,
    "_title":"Rural Capacity Building Project (RCBP)"
},
{
    "id":26301,
    "_title":"Disaster Risk Reduction and Early Recovery Project"
}
```


↩ Back to Function List

pmt_activity_titles

Description

Provides a listing of activity titles for a given set of activity ids.

Parameter(s)

1. activity_ids (integer[]) – **Required**. Array of activity ids.

Result

Json with the following:

1. id (integer) – activity id.
2. _title (character varying) – title of activity.

Example(s)

- Activity titles for the following activity ids (26271,26283,26284,26286,26287):

```
SELECT * FROM pmt_activity_titles(ARRAY[26271,26283,26284,26286,26287]);
```

```
{
    "id":26325,
    "_title":"East Africa Agriculture Productivity Project"
},
{
    "id":26284,
    "_title":"Ethiopia Productive Safety Net APL III Project (P113220)"
},
{
    "id":26271,"_title":"African Stockpiles Project"
},
{
    "id":26283,
    "_title":"Rural Capacity Building Project (RCBP)"
},
{
    "id":26286,
    "_title":"Tana Beles Integrated Water Resources Development Project
(TBIWRDP)"
},
{
    "id":26287,
    "_title":"Agricultural Growth"
}
```


↩ Back to Function List

pmt_auto_complete

Description

Function accepting columns, of data type text or character varying, for the activity table and compiles a list of unique data from those fields for use in an autocomplete or type ahead function.

Parameter(s)

1. filter_fields (character varying) – **Required**. Comma separated string of activity table column names.

Result

A text array of distinct values from all provided columns. Each text element returned is restricted to 100 characters.

Example(s)

- Tags and opportunity_id (custom instance field) from the activity table:

```
SELECT * FROM pmt_auto_complete('_tags,opportunity_id');
```

```
{
  "autocomplete":
  [
    "4h",
    "accelerate country action",
    "access",
    "accountability",
    "adopt",
    "adoption",
    "advocacy",
    "advocacy and campaigns",
    "advocate for fctc measures",
    ...
  ]
}
```

↩ Back to Function List

pmt_boundaries_by_point

Description

Accepts a well-known text point and returns all intersected boundary features.

Parameter(s)

1. wktpoint (character varying) – **Required**. Well-known text representation of a point feature.

Result

An array of json objects containing all intersected boundary features:

1. boundary_id (integer) – id of the boundary intersected.
2. boundary_name (character varying) – the name of the boundary intersected.
3. feature_id (integer) – id of the feature of the boundary intersected.
4. feature_name (character varying) – the name of the feature of the boundary intersected.

Example(s)

- Determine what boundries are under the point 'POINT(38.758578 8.942346)':

```
SELECT * FROM pmt_boundaries_by_point('POINT(38.758578 8.942346)');
```

```
[
    {
        "boundary_id":1,
        "boundary_name":"gaul0",
        "feature_id":86,
        "feature_name":"Ethiopia"
    },
```

```
    {
        "boundary_id":2,
        "boundary_name":"gaul1",
        "feature_id":951,
        "feature_name":"Addis Ababa"
    },
    {
        "boundary_id":3,
        "boundary_name":"gaul2",
        "feature_id":15117,
        "feature_name":"Addis Ababa Zone3"
    },
    ...
    {
        "boundary_id":18,
        "boundary_name":"gadm3",
        "feature_id":33778,
        "feature_name":"Akaki - Kalit"
    }
```


↩ Back to Function List

pmt_boundary_extents

Description

Get boundary extent of a boundary feature(s).

Parameter(s)

1. boundary_table (character varying) - **Required** the name of the boundary table.
2. feature_names (character varying) - **Required** comma delimited list of features to include in extent.

Result

Json with the following:

1. extent (wkt polygon) – well-known text representation of a polygon as the extent of the requested features

Example(s)

- Get extent of Ethiopia and Mali from the gamd0 (spatial table: gadm0) boundary:

```
SELECT * FROM pmt_boundary_extents('gadm0','Ethiopia,Mali'); ;
```


```
{
    "extent":"POLYGON(
        (-12.2389116287231 3.39882302284263,
        -12.2389116287231 25.0000000000001,
        47.9582290649417 25.0000000000001,
        47.9582290649417 3.39882302284263,
```

```
        -12.2389116287231 3.39882302284263))"
}
```

↩ Back to Function List

pmt_boundary_feature

Description

Returns all administrative level names for a give boundary feature.

Parameter(s)

1. boundary_id (integer) - **Required** the boundary id for the feature.
2. feature_id (integer) - **Required** the feature id for the feature to query.

Result

Json with the following:

1. 0_name (character varying) – the country level name (if exists)
2. 1_name (character varying) – the admin 1 level name (if exists)
3. 2_name (character varying) – the admin 2 level name (if exists)
4. 3_name (character varying) – the admin 3 level name (if exists)
5. admin_level (character varying) – the number for the administrative level: 0 (country), 1 (admin 1), 2 (admin 2) or 3 (admin 3)

Example(s)

- Get all administrative level names for Ethiopia (feature_id: 74) in GADM 0 (boundary_id: 15):

```
SELECT * FROM  pmt_boundary_feature(15, 74);
```

```
{
    "0_name":"Ethiopia",
    "admin_level":"0"
}
```

- Get all administrative level names for Addis Abeba(feature_id: 15174) in GADM 2 (boundary_id: 17):

```
SELECT * FROM  pmt_boundary_feature(17, 15174);
```

```
{
    "0_name":"Ethiopia",
    "1_name":"Addis Abeba",
    "2_name":"Addis Abeba",
    "admin_level":"2"
}
```

↩ Back to Function List

pmt_boundary_hierarchy

Description

Creates a nested boundary hierarchy (tree view structure) of boundary feature ids and names.

Parameter(s)

1. boundary_type (character varying) - **Required** The boundary type for the created hierarchy. Options: gaul, gadm, unocha, nbs.
2. admin_levels (character varying) - a comma delimited list of admin levels to include. Options: 0,1,2,3
3. filter_features (character varying) - a comma delimited list of names of features in the highest admin level to restrict data to.
4. data_group_ids (character varying) - a comma delimited list of data group ids to filter returned boundaries to. Only boundary features where data group(s) have location information will be returned.

Result

Json with the following:

1. b0-b3 (integer) - the boundary id of each level of the hierarchy (number does not correlate to admin level).
2. boundaries (object[]) - array of objects containing the boundary hierarchy
    1. id (integer) - the feature id (first boundary level of hierarchy)
    2. n (character varying) - the feature name (first boundary level of hierarchy)
    3. b (object[]) - array of objects containing the next boundary level
        1. id (integer) - the feature id (second boundary level of hierarchy)
        2. n (character varying) - the feature name (second boundary level of hierarchy)
        3. b (object[]) - array of objects containing the next boundary level
            1. pattern continued for each level of boundary hierarchy (based on requested number of admin levels)

Example(s)

- Create a boundary hierarchy using GADM for admin levels 0-2 for Ethiopia:

```
SELECT * FROM pmt_boundary_hierarchy('gadm','0,1,2','Ethiopia');
```


```
{
    "b0":15,
    "b1":16,
    "b2":17,
    "boundaries":[
    {
        "id":74,
        "n":"Ethiopia",
```

```
        "b1":[
        {
            "id":896,
            "n":"Oromia",
            "b2":[
            {
                "id":15203,
                "n":"Debub Mirab Shewa"
            },
            {
                "id":15200,
                "n":"Arsi"
            },
            {
                "id":15204,
                "n":"Guji"
            },
            ...
            {
                "id":15216,
                "n":"North Shewa"
            }]
        },
        ...
        {
            "id":890,
            "n":"Afar",
            "b2":[
            {
                "id":15178,
                "n":"Afar Zone 4"
            },
            {
                "id":15179,
                "n":"Afar Zone 5"
            },
            ...
            {
                "id":15177,
                "n":"Afar Zone 3"
            }]
        }]
    }]
}
```


↩ Back to Function List

pmt_boundary_filter

Description

Filters boundary features.

Parameter(s)

1. boundary_table (character varying) - **Required** the name of the boundary table to filter.
2. query_field (character varying) - **Required** the name of the field in the boundary table to apply query to.
3. query (character varying) - **Required** comma delimited list of values in the query field to restrict features to.

Result

Json with the following:

1. id (integer) – the feature id.
2. _name (character varying) - the name of the feature

Example(s)

- Get all gadm regions (spatial table: gadm1) in Ethiopia and Mali:

```
SELECT * FROM pmt_boundary_filter('gadm1','_gadm0_name','Ethiopia,Mali'); ;
```

```
{
    "id":1768,
    "_name":"Gao"
},
{
    "id":896,
    "_name":"Oromia"
},
{
    "id":891,
    "_name":"Amhara"
}
...
{
    "id":1773,
    "_name":"Ségou"
},
{
    "id":1774,
    "_name":"Sikasso"
}
```

↩ Back to Function List

pmt_boundary_pivot

Description

Function to create a filterable pivot table showing organization represation within a single taxonomy and boundary (row and column) based on a single organization role.

Parameter(s)

1. pivot_boundary_id (integer) - **Required** boundary id for the boundary that will represent either rows (y axis of pivot) or columns (x axis of pivot). The boundary used to pivot **must** be of the same type as the boundary to filter (i.e both must be gadm). The pivot boundary cannot be the same level as the filter. For example, if the filter is admin level 0 then the pivot must an admin level higher (1, 2, ect).
2. pivot_taxonomy_id (integer) - **Required** taxonomy id for taxonomy that will represent either rows (y axis of pivot) or columns (x axis of pivot).
3. boundary_as_row (boolean) - T/F will the boundary be represented as rows. Default is false, meaning the boundary is represented as a column and the taxonomy is the row.
4. org_role_id (integer) - **Required** classification id for the Organisation Role to be used to select participating organizations (table data).
5. data_group_ids (character varying) - comma seperated list of classification_id(s) from the Data Group taxonomy to restrict data. If no data group id is provided, all data groups are included (filter).
6. classification_ids (character varying) - comma seperated list of classification_id(s) for any taxonomy (filter).
7. start_date (date) - start date for activities (filter).
8. end_date (date) - end date for activities (filter).
9. boundary_id (integer) - **Required** the boundary id referenced by the feature_id (filter).
10. feature_id (integer) - **Required** the featuer id to restrict activities to (filter).

Result

Json with the following:

1. c# (character varying) - each object will contain a value for each column in a row numbered 1 - x
    1. f1 (character varying) - organization label
    2. f2 (integer) - organization id
    3. f3 (character varying) organization name

Example(s)

- Partner pivot table data to include only EthATA data (data_group_id: 2237) where Program is the row/y-axis (pivot_taxonomy_id: 68) and GADM Level 1 is the column/x-axis (pivot_boundary_id: 16 & boundary_as_row: false) and the organizations represented are Funding (org_role_id: 496) for all activities in Ethiopia (feature_id: 74) using GADM Level 0 (boundary_id: 15):

```
SELECT * FROM
pmt_boundary_pivot(16,68,false,496,'2237',null,null,null,15,74);
```


```
{
    "c1":"",
    "c2":"Oromia",
    "c3":"Amhara",
    "c4":"Benshangul-Gumaz",
    "c5":"Dire Dawa",
    "c6":"Addis Abeba",
    "c7":"Gambela Peoples",
    "c8":"Harari People",
    "c9":"Somali",
    "c10":"Southern Nations, Nationalities and Peoples",
    "c11":"Tigray",
    "c12":"Afar"
}
{
    "c1":"Agricultural Growth Program",
    "c2":[
        {
            "f1":"CIDA",
            "f2":3135,
            "f3":"Canadian International Development Agency (CIDA)"
        },

        {
            "f1":"Global Agriculture and Food Security Program",
            "f2":2588,
            "f3":"Global Agriculture and Food Security Program"
        },
        ...

        {
            "f1":"GoE",
            "f2":3148,
            "f3":"Government of Ethiopia (GoE)"
        }
    ],
    "c3":[
        {
            "f1":"CIDA",
            "f2":3135,
            "f3":"Canadian International Development Agency (CIDA)"
        },

        {
            "f1":"Global Agriculture and Food Security Program",
            "f2":2588,
            "f3":"Global Agriculture and Food Security Program"
```

```
    },
    ...

    {
        "f1":"GoE",
        "f2":3148,
        "f3":"Government of Ethiopia (GoE)"
    }
],
"c4":[null],
"c5":[null],
"c6":[null],
"c7":[null],
"c8":[null],
"c9":[null],
"c10":[
    {
        "f1":"CIDA",
        "f2":3135,
        "f3":"Canadian International Development Agency (CIDA)"
    },

    {
        "f1":"Global Agriculture and Food Security Program",
        "f2":2588,
        "f3":"Global Agriculture and Food Security Program"
    },
    ...

    {
        "f1":"GoE",
        "f2":3148,
        "f3":"Government of Ethiopia (GoE)"
    }
],
"c11":[
    {
        "f1":"CIDA",
        "f2":3135,
        "f3":"Canadian International Development Agency (CIDA)"
    },

    {
        "f1":"Global Agriculture and Food Security Program",
        "f2":2588,
        "f3":"Global Agriculture and Food Security Program"
    },
    ...
```

```
        {
            "f1":"GoE",
            "f2":3148,
            "f3":"Government of Ethiopia (GoE)"
        }
    ],
    "c12":[null]
},
{
    "c1":"Concern Livlihoods Program",
    "c2":[null],
    "c3":[null],
    "c4":[null],
    "c5":[null],
    "c6":[null],
    "c7":[null],
    "c8":[null],
    "c9":[null],
    "c10":[null],
    "c11":[null],
    "c12":[
        {
            "f1":"Concern Ethiopia",
            "f2":3166,
            "f3":"Concern Ethiopia"
        },

        {
            "f1":"EU",
            "f2":1027,
            "f3":"European Union (EU)"
        },

        {
            "f1":"WRDA",
            "f2":3222,
            "f3":"Wonnta Rural Development Association (WRDA)"
        }
    ]
},
{
    "c1":"Food Security Program",
    "c2":[null],
    "c3":[null],
    "c4":[null],
    "c5":[null],
    "c6":[null],
    "c7":[null],
    "c8":[null],
    "c9":[null],
```

```
    "c10":[null],
    "c11":[null],
    "c12":[
        {
            "f1":"CIDA",
            "f2":3135,
            "f3":"Canadian International Development Agency (CIDA)"
        },

        {
            "f1":"DEFID",
            "f2":3170,
            "f3":"DEFID"
        },

        {
            "f1":"Department of Foreign and International Affairs",
            "f2":3171,
            "f3":"Department of Foreign and International Affairs"
        }
    ]
},
{
    "c1":"Stand Alone Project",
    "c2":[
        {
            "f1":"GDC",
            "f2":3149,
            "f3":"German Development Cooperation (GDC)"
        }
    ],
    "c3":[
        {
            "f1":"ITAL",
            "f2":3100,
            "f3":"Italian Development Cooperation (ITAL)"
        }
    ],
    "c4":[
        {
            "f1":"EU",
            "f2":1027,
            "f3":"European Union (EU)"
        },
        {
            "f1":"ITAL",
            "f2":3100,
            "f3":"Italian Development Cooperation (ITAL)"
        },
```

```
        {
            "f1":"UNDP",
            "f2":3093,
            "f3":"United Nations Development Program (UNDP)"
        }
    ],
    "c5":[null],
    "c6":[null],
    "c7":[null],
    "c8":[null],
    "c9":[null],
    "c10":[null],
    "c11":[null],
    "c12":[
        {
            "f1":"ADA",
            "f2":3163,
            "f3":"Austrian Development Agency (ADA)"
        }
    ]
```


Results can then be rendered as a table:

	Oromia	Amhara	Benshangul-Gumaz	Dire Dawa	Addis Abeba	Gambela Peoples	Hareri People	Somali	Southern Nations, Nationalities and Peoples	Tigray	Afar
Agricultural Growth	CIDA,Global Agriculture and Food Security Program, GoE,Local,NETH,SPAN,USAID,WB,	CIDA,Global Agriculture and Food Security Program,GoE,Local,NETH,SPAN,USAID,WB,							CIDA,Global Agriculture and Food Security Program,GoE, Local,NETH,SPAN,USAID,WB,	CIDA, Global Agriculture and Food Security Program,GoE,Local, NETH, SPAN, USAID, WB,	

Program							
Concern Livlihoods Program		Concern Ethiopia,				Concern Ethiopia,EU,WRDA,	
Food Security Prog	CIDA,DEFID,Department of Foreign and International Affairs,EC,GoE,IREAID,NETH, SWED,USAID,WB, WFP,	CIDA,DEFID,Department of Foreign and International Affairs,EC,GoE,IREAID,NETH,SWED,USAID, WB,WFP,	Department of Foreign and Interna	Department of Foreign and Interna	CIDA,DEFID,EC,IREAID,NETH,SWED,USAID,WB,WFP,	CIDA,DEFID,Department of Foreign and International Affairs, EC,GoE, IREAID, NETH,SWED,USAID,WB,WFP,	CIDA, DEFID,Department of Foreign and International Affairs,EC,GoE,IREAID,NETH,SWED,US

ram				tional Affairs,GoE,IREAID,WB,			tional Affairs,GoE,IREAID,WB,			AID,WB,WFP,	
Stand Alone Project	African Development Bank,Belgium Cooperation,CIDA,Department of Foreign and International Affairs,EU,FIN,GDC,GIZ,GoE,IFAD,IREAID,ITAL,JICA,Korea,Local,NETH,Regional Government,Royal Norwegian Embassy,Russia,SPAN,UNDP,UNICEF,USAID,WB,WFP,	ADA,African Development Bank,CIDA,DANIDA,Department of Foreign and International Affairs,EU,FIN,GoE,IFAD,IREAID,ITAL,JICA,Korea,Local,MEDA,Regional Government,Russia,SIDA,SPAN,UNDP,UNICEF,USAID,WB,WFP,	CIDA,Department of Foreign and International Affairs,EU,FIN,GoE,WB,	EU,	CIDA,GoE,Local,MEDA,UNDP,USAID,	CIDA,EU,UNDP,WB,	CIDA,EU,GoE,IFAD,International Rescue Committee,IREAID,JICA,Korea,Local,Russia,SPAN,UNDP,USAID,WB,WFP,	African Development Bank,Christian Aid,CIDA,Department of Foreign and International Affairs,EU,GoE,IFAD,IREAID,JICA,Korea,Local,MEDA,Regional Government,Royal Norwegian Embassy,Russia,SPAN,UNDP,UNICEF,USAID,	African Development Bank,CIDA,Department of Foreign and International Affairs,EU,GoE,IFAD,IREAID,ITAL,JICA,Korea,Regional Government,Royal Norwegian Embassy,Russia,SPAN,UNDP,UNICEF,U	CIDA,Department of Foreign and International Affairs,EU,GoE,IFAD,IIRR,Local,Norway Development Fund,OCHA,SPAN,UNDP,USAID,WB

				WB,WFP,	SAID, WB,WFP,
Sustainable Land Management Program	CIDA,DED,EU,GIZ,IFAD,KFW, MASHAV, WB,	CIDA,DED,EU,FIN,GIZ,IFAD,KFW, MASHAV,WB,	CIDA,DED,EU,FIN,GIZ,IFAD, KFW, WB,	CIDA,DED,EU, GIZ,IFAD,KFW, MASHAV,WB,	CIDA, DED,EU,GIZ,IFAD,KFW,MASHAV,WB,

↩ Back to Function List

pmt_bytea_import ==========

Description

Converts text into bytea. Used in combination with PostgreSQL convert_from() to import xml documents as an xml data type. (Jack Douglas)[1]

Parameter(s)

1. (text) – any text, or document.

Result

Text as bytea.

Example(s)

- Convert utf-8 formatted xml file in the temp directory called file.xml into the xml data type:

```
convert_from(pmt_bytea_import('/temp/file.xml'), 'utf-8')::xml
```

pmt_category_root ==========

Description

A taxonomy can have a taxonomy category and a taxonomy category can have a taxonomy category. This function returns the base or root taxonomy_id of any taxonomy.

Parameter(s)

1. id (integer) – **Required**. The category taxonomy.
2. data_group (character varying) – Optional. Comma separated list of data group classification id(s).

Result

Integer of root taxonomy_id.

Example(s)

- Return the root taxonomy for the PMT Sector Category (taxonomy_id:16) taxonomy category:

```
SELECT pmt_category_root(16, null);
```

15

pmt_clone_activity ==========

Description

Clone an existing activity and all it's related records. The clone will replicate the following activity records: * contacts * taxonomies * participation records (and associated taxonomies) * financial records (and associated taxonomies) * detail records (and associated taxonomies) * result records (and associated taxonomies)

Parameter(s)

1. user_id (integer) – **Required**. User_id of user requesting edits.
2. activity_id (integer) – **Required**. Activity_id of activity to clone.
3. project_id (integer) – Optional. Project_id of newly cloned activity.
4. json (json) - Optional. Key/value pair as json of field/values to edit. Activity is cloned, then columns submitted in json are updated. Column names are case sensitive. Enclose all values in double quotes, including null. **If your text values include a single quote,**

use two adjacent single quotes, e.g., 'Dianne''s dog'. This is because PostgreSQL uses the single quote to encase string constants. You can include any existing field that exists for activity, even custom fields. The following fields cannot be edited evenif included:

- activity_id
- project_id
- active
- retired_by
- created_by
- created_date
- updated_by
- updated_date

Json key/value format:

```
{"column_name": "value"}
```

Result

Json with the following:

1. id (integer) – activity_id of then new activity.
2. message(character varying) - Message containing either "Success" for a successful transaction or containing an error description regarding the unsuccessful transaction. Some possible error messages:
   - user_id and activity_id are required parameters for all operations
   - Invalid user_id
   - Invalid activity_id
   - User does NOT have authority to create new records on this project

Example(s)

- User sparadee (user_id: 34) to clone activity "Advancing through Sustainable Diets" (activity_id: 45) changing the title and description. (*By not providing the project_id, the new activity will remain on the same project as the activity it was cloned from*)

```
select * from pmt_clone_activity(34, 45, null,'{"title": "Advancing through
Sustainable Diets: Phase II",
"description":"Phase II of the activity."}');
```

```
{"id":863,"message":"Success"}
```

- User sparadee (user_id: 34) to clone activity "Rwanda Super Foods" (activity_id: 661) for project "Improving Rwanda Food Supplies Phase II" (project_id: 756):

```
select * from pmt_clone_activity(34, 661, 756 ,null);
```

```
{"id":967,"message":"Success"}
```

- Delete activity_id 15821.

```
select * from pmt_edit_activity(34, 15821, null, null, true);
```

```
{"id":15821,"message":"Success"}
```

pmt_contacts ==========

Description

Get all contacts.

Parameter(s)

No parameters.

Result

Ordered by last name then first name. Json with the following:

1. c_id (integer) – contact_id.
2. first_name (character varying(64)) – first name of contact.
3. last_name (character varying(128)) – last name of contact.
4. email (character varying(100)) - email of contact.
5. o_id (integer) – organization_id in which the contact belongs to.
6. org (character varying(255)) - organization name of the organization in which the contact belongs to.

Example(s)

```
SELECT * FROM pmt_contacts();
```

```
...
{
    "c_id":4,
    "first_name":"John",
    "last_name":"Doe",
    "email":"john.doe@mymail.com",
    "o_id":13,
    "org":"FAO"
}
...
```

pmt_countries ==========

Description

Filter countries by classifications.

Parameter(s)

1. classification_ids (character varying) – Optional. Restrict data to classification(s).

Result

Json with the following:

1. c_id (integer) – classification_id.
2. name (character varying(255)) – name of country.
3. bounds (json object) – bounding box of country.

Example(s)

- Country for Afghanistan:

```
SELECT * FROM pmt_countries('24');
```


```
{
    "c_id":24,
    "name":"afghanistan",
    "bounds":"{
        \"type\":\"Polygon\",

\"coordinates\":[[[60.4758911132812,29.3773193359375],[60.4758911132812,38.49
072265625],[74.889892578125,38.49072265625],[74.889892578125,29.3773193359375
],[60.4758911132812,29.3773193359375]]]
    }
}
```


pmt_create_user ==========

Description

Create new user.

Parameter(s)

1. organization_id (integer) – **Required**. Organization id user will be assigned to.
2. data_group_id (integer) - **Required**. Data group id user will be assigned to.
3. role_id (integer) – **Required**. Role id user will be assigned to.
4. username (character varying(255)) - **Required**. User username.
5. password (character varying(255)) - **Required**. User password.
6. email (character varying(255)) - **Required**. User email address.
7. first_name (character varying(150)) - Optional. User first name.
8. last_name (character varying(150)) - Optional. User last name.

Result

Boolean. Successful (true). Unsuccessful (false).

Example(s)

- Create new user for Jane Doe as a Reader (role_id:1) for BMGF organization (organization_id:13) in data group BMGF(classification_id:768):

```
SELECT * FROM pmt_create_user(13, 768, 1, 'janedoe', 'secretpassword',
'jane.doe@email.com', 'Jane', 'Doe');
```

```
TRUE
```

pmt_data_groups

Description

Returns all active data groups. Data group is a taxonomy that all data into group by source or owner. Data group is use to determine user permissions to a record.

Parameter(s)

None.

Result

1. c_id (integer) – Classification_id of the Data Group classification.
2. name (character varying) – Name of the Data Group classification.

Example(s)

- Get data groups:

```
SELECT * FROM pmt_data_groups();
```

c_id	Name
768	"BMGF"
769	"AGRA"
1068	"AWG"
…	…

↩ Back to Function List

pmt_edit_activity ==========

Description

Edit an activity.

Parameter(s)

1. user_id (integer) – **Required**. User_id of user requesting edits.
2. activity_id (integer) – **Required for delete and update operations**. Activity_id of activity to update or delete.
3. project_id (integer) – **Required for create operation**. Project_id of activity to create.
4. json (json) - **Required for create and update operations**. Key/value pair as json of field/values to edit. Column names are case sensitive. Enclose all values in double quotes, including null. **If your text values include a single quote, use two adjacent single quotes, e.g., 'Dianne''s dog'. This is because PostgreSQL uses the single quote to encase string constants.** You can include any existing field that exists for activity, even custom fields. The following fields cannot be edited even if included:
   - activity_id
   - project_id

- active
- retired_by
- created_by
- created_date
- updated_by
- updated_date

Json key/value format:

```
{"column_name": "value"}
```

5. delete_record (boolean) - Optional, default is false. True to request the record be deleted.

Result

Json with the following:

1. id (integer) – activity_id of the activity created, updated or deleted.
2. message(character varying) - Message containing either "Success" for a successful transaction or containing an error description regarding the unsuccessful transaction. Some possible error messages:
   - The json parameter is required for a create/update operation
   - project_id is required for a create operation - When activity_id is null a create operation is assumed an project_id is required.
   - activity_id is required for a delete operation - When delete_record is true, an activity_id is required.
   - user_id is a required parameter for all operations
   - User does NOT have authority to create a new activity for project_id
   - User does NOT have authority to delete this activity
   - project_id is not valid - the provide project_id is either not active or invalid
   - User does NOT have authority to update this activity

Example(s)

- Update the title, description and start_date for activity_id 14863 and set it's opportunity_id to null.

```
select * from pmt_edit_activity(34,14863, null,'{"title": "Project Objective
1",
"description":"Market opportunities, Policies and Partnerships",
"start_date":"9-2-2012",
"opportunity_id": "null"}', false);

{"id":14863,"message":"Success"}
```

- Create a new activity for project_id 749.

```
select * from pmt_edit_activity(34, null, 749,'{"title": "A New Activity",
"description":"Doing some good work in Nepal", "start_date":"6-1-2014",
"end_date":"5-31-2016"}', false);
```

```
{"id":15821,"message":"Success"}
```

- Delete activity_id 15821.

```
select * from pmt_edit_activity(34, 15821, null, null, true);
```

```
{"id":15821,"message":"Success"}
```

pmt_edit_activity_contact ==========

Description

Edit the relationship between an activity and a contact.

Parameter(s)

1. user_id (integer) – **Required**. User_id of user requesting edits.
2. activity_id (integer) – **Required**. Activity_id of activity to edit.
3. contact_id (integer) – **Required**. Contact_id of contact to associate to activity_id.
4. edit_action (enum) - Optional. Options:
    1. add (default) - will add a relationship between provided activity_id and contact_id
    2. delete - will remove the relationship between the provided activity_id and contact_id
    3. replace - will replace all existing relationships to the provided activity_id with any contact, with the contact of the provided contact_id.

Result

Boolean. Sucessfull (true) or unsuccessful (false). Unsuccessful is usually due invalid parameters or user does not have authorization to edit. Use pmt_validate_user_authority to determine authorization.

Example(s)

- Add Don John (contact_id:169) as a contact for activity_id 14863 as user sparadee (user_id:34):

```
select * from pmt_edit_activity_contact(34,14863, 169, 'add');
```

```
TRUE
```

- Replace all contacts with Edward Jones (contact_id:145) for activity_id 14863 as user sparadee (user_id:34):

```
select * from pmt_edit_activity_contact(34,14863, 145, 'replace');
```

```
TRUE
```

- Delete Edward Jones (contact_id:145) as contact for activity_id 14863 as user sparadee (user_id:34):

```
select * from pmt_edit_activity_contact(34,14863, 145, 'delete');
```

```
TRUE
```

pmt_edit_activity_taxonomy ==========

Description

Edit the relationship between activity(ies) and a taxonomy classification(s). **Important Note: This function DOES NOT call the refresh_taxonomy_lookup() function, which updates the materialized views that support a majority of the functions in the database. Be sure to call this function when editing is complete.**

Parameter(s)

1. user_id (integer) – **Required**. User_id of user requesting edits.
2. activity_ids (character varying) – **Required**. Comma separated list of activity_ids to edit.
3. classification_ids (character varying) – **Required/Optional**. Comma separated list of classification_ids of taxonomy classifications in relationship to activity_id(s). Not required if using remove_taxonomy_ids parameter.
4. remove_taxonomy_ids (character varying) – **Required/Optional**. Comma separated list of taxonomy_ids to remove from relationships to activity_id(s). Not required if using classification_ids parameter.
5. edit_activity (enum) - Optional. Options:
    1. add (default) - will add a relationship between provided activity_id(s) and classification_id(s)
    2. delete - will remove the relationship between the provided activity_id(s) and classification_id(s)
    3. replace - will replace all relationships between the provided activity_ids(s) and taxonomy of the provided classification_id(s), with a relationship between the provided activity_id(s) and classification_id(s) per taxonomy.

Result

Boolean. Sucessfull (true) or unsuccessful (false). Unsuccessful is usually due invalid parameters or permissions.

Example(s)

- Add a relationship to Sub-Imitative 'Headquarters' (classification_id:779) and to Crops and Livestock 'Cocoa' (classification_id:1196) to activity_ids 13264,13355,13361:

```
select * from pmt_edit_activity_taxonomy(54,'13264,13355,13361','779,1196',
null, 'add');
```

```
TRUE
```

- Remove the relationship to Sub-Initiative 'Seed Systems' (classification_id:791) and to Crops and Livestock 'Oranges' (classification_id:1192) from activity_ids 13264,13355,13361:

```
select * from pmt_edit_activity_taxonomy(54,'13264,13355,13361','791,1192',
null, 'delete');
```

```
TRUE
```

- Replace all relationships to the Sub-Initiative taxonomy with the relationship to Sub-Initiative 'Crop Improvements' (classification_id:773) and replace all the relationships to the Crops and Livestock taxonomy with the relationships to the Crops and Livestock 'Tomatoes' (classification_id:1180) and 'Sweet Pepper' (classification_id:1190) for activity_ids 13264,13355,13361:

```
select * from pmt_edit_activity_taxonomy(54, '13264,13355,13361',
'773,1180,1190', null, 'replace');
```

```
TRUE
```

- Remove all the relationships within the Crops and Livestock taxonomy (taxonomy_id:51) from activity_ids 13264,13355,13361:

```
select * from pmt_edit_activity_taxonomy(54, '13264,13355,13361', null, '51',
null);
```

```
TRUE
```

pmt_edit_contact ==========

Description

Edit a contact.

Parameter(s)

1. user_id (integer) – **Required**. User_id of user requesting edits.
2. contact_id (integer) – **Required for update and delete operations**. Contact_id of existing contact to update or delete, if left null then a new contact record will be created.
3. json (json) - **Required**. Key/value pair as json of field/values to edit. Column names are case sensitive. Enclose all values in double quotes, including null. **If your text values include a single quote, use two adjacent single quotes, e.g., 'Dianne''s dog'. This is because PostgreSQL uses the single quote to encase string constants.** You can include any existing field that exists for contact, even custom fields. The following fields cannot be edited even if included:
   - contact_id
   - active
   - retired_by
   - created_by

- created_date
- updated_by
- updated_date

Json key/value format:

```
{"column_name": "value"}
```

4. delete_record (boolean) - Optional, default is false. True to request the record be deleted.

Result

Json with the following:

1. id (integer) – contact_id of the contact created or updated.
2. message(character varying) - Message containing either "Success" for a successful transaction or containing an error description regarding the unsuccessful transaction. Possible error messages:
   - Must include user_id and json data parameters - Received when both required parameters are not provided.
   - User does NOT have authority to create a new contact - Received when the user_id provided does not have authority to create under current role.
   - User does NOT have authority to update an existing contact - Received when the user_id provided does not have authority to update under current role.
   - Invalid contact_id - Received when the contact_id provided is invalid.

Example(s)

- Update the email and title for John Hancock (contact_id:148) as user sparadee (user_id:34)

```
select * from pmt_edit_contact(34,148,'{"email":"jhanhock@mail.com",
"title":"CEO"}', false);
```

```
{"id":148,"message":"Success"}
```

- Add new contact for BMGF (organization_id:13) as user sparadee (user_id:34)

```
select * from pmt_edit_contact(34,null,'{"first_name":"John",
"last_name":"Hanhock", "email":"jhanhock@mail.com",
"title":"CEO", "organization_id": 13}', false);
```

```
{"id":672,"message":"Success"}
```

- Delete contact_id 672 as user sparadee (user_id:34)

```
select * from pmt_edit_contact(34,672,null, true);
```

```
{"id":672,"message":"Success"}
```

pmt_edit_detail ==========

Description

Edit a detail.

Parameter(s)

1. user_id (integer) – **Required**. User_id of user requesting edits.
2. detail_id (integer) – **Required for update and delete operations**. detail_id of existing detail to update or delete, if left null then a new detail record will be created.
3. project_id (integer) – **Required for create operations on project details**. project_id of detail to create.
4. activity_id (integer) – **Required for create operations on activity details**. activity_id of detail to create.
5. json (json) - **Required for update and create operations**. Key/value pair as json of field/values to edit. Column names are case sensitive. Enclose all values in double quotes, including null. **If your text values include a single quote, use two adjacent single quotes, e.g., 'Dianne''s dog'. This is because PostgreSQL uses the single quote to encase string constants.** You can include any existing field that exists for detail, even custom fields. The following fields cannot be edited even if included:
   - detail_id
   - active
   - retired_by
   - created_by
   - created_date
   - updated_by
   - updated_date

   Json key/value format:

   ```
   {"column_name": "value"}
   ```

6. delete_record (boolean) - Optional, default is false. True to request the record be deleted.

Result

Json with the following:

1. id (integer) – detail_id of the detail created or updated.
2. message(character varying) - Message containing either "Success" for a successful transaction or containing an error description regarding the unsuccessful transaction. Possible error messages:
   - Must include json parameter when delete_record is false
   - Must include detail_id when delete_record is true
   - Must include project_id or activity_id parameter when detail_id parameter is null
   - Must include user_id parameter

- Invalid project_id

Example(s)

- Update title for detail_id 136 as user sparadee (user_id:34)

```
select * from pmt_edit_detail(34,136,null,null,'{"title": "Description of
Activities Related to Nutrition"}',
false);
```

```
{"id":136,"message":"Success"}
```

- Add new detail for activity_id 493 as user sparadee (user_id:34)

```
select * from pmt_edit_detail(34,null, null, 493,'{"title": "Test Title",
"description":"a description",
"amount":3}', false);
```

```
{"id":672,"message":"Success"}
```

- Add new detail for project_id 13 as user sparadee (user_id:34)

```
select * from pmt_edit_detail(34, null, 13, null, '{"title": "Test Title",
"description":"a description",
"amount":3}', false);
```

```
{"id":673,"message":"Success"}
```

- Delete detail_id 673 as user sparadee (user_id:34)

```
select * from pmt_edit_detail(34, 673, null, null, null, true);
```

```
{"id":673,"message":"Success"}
```

pmt_edit_financial ==========

Description

Edit a financial record.

Parameter(s)

1. user_id (integer) – **Required**. User_id of user requesting edits.
2. financial_id (integer) – **Required for update and delete operations**. financial_id of existing financial to update or delete, if left null then a new financial record will be created.
3. project_id (integer) – **Required for create operations on project financials**. project_id of financial record to create.
4. activity_id (integer) – **Required for create operations on activity financials**. activity_id of financial record to create.
5. json (json) - **Required for update and create operations**. Key/value pair as json of field/values to edit. Column names are case sensitive. Enclose all values in double quotes, including null. **If your text values include a single quote, use two adjacent single quotes, e.g., 'Dianne''s dog'. This is because PostgreSQL uses the single**

quote to encase string constants. You can include any existing field that exists for financial, even custom fields. The following fields cannot be edited even if included:

- financial_id
- active
- retired_by
- created_by
- created_date
- updated_by
- updated_date

Json key/value format:

```
{"column_name": "value"}
```

6. delete_record (boolean) - Optional, default is false. True to request the record be deleted.

Result

Json with the following:

1. id (integer) – financial_id of the financial created or updated.
2. message(character varying) - Message containing either "Success" for a successful transaction or containing an error description regarding the unsuccessful transaction. Possible error messages:
   - Must include json parameter when delete_record is false
   - Must include financial_id when delete_record is true
   - Must include project_id or activity_id parameter when financial_id parameter is null
   - Must include user_id parameter
   - Invalid project_id

Example(s)

- Update amount and start_date for financial_id 136 as user sparadee (user_id:34)

```
select * from pmt_edit_financial(34,136,null,null,'{"amount": 130900.00,
"start_date":"1-1-2014"}', false);
```

```
{"id":136,"message":"Success"}
```

- Add new financial record for activity_id 493 as user sparadee (user_id:34)

```
select * from pmt_edit_financial(34,null, null, 493,'{"amount": "100500.00",
"start_date":"1-1-2014",
"end_date":"12-31-2016"}', false);
```

```
{"id":672,"message":"Success"}
```

- Add new financial for project_id 13 as user sparadee (user_id:34)

```
select * from pmt_edit_financial(34, null, 13, null, '{"amount":
"1000000.00",
"start_date":"1-1-2014", "end_date":"12-31-2016"}', false);
```

```
{"id":673,"message":"Success"}
```

- Delete financial_id 673 as user sparadee (user_id:34)

```
select * from pmt_edit_financial(34, 673, null, null, null, true);
```

```
{"id":673,"message":"Success"}
```

pmt_edit_financial_taxonomy ==========

Description

Edit the relationship between financial(s) and a taxonomy classification(s). **Important Note: This function DOES NOT call the refresh_taxonomy_lookup() function, which updates the materialized views that support a majority of the functions in the database. Be sure to call this function when editing is complete.**

Parameter(s)

1. user_id (integer) – **Required**. User_id of user requesting edits.
2. financial_ids (character varying) – **Required**. Comma separated list of financial_ids to edit.
3. classification_ids (character varying) – **Required/Optional**. Comma separated list of classification_ids of taxonomy classifications in relationship to financial_id(s). Not required if using remove_taxonomy_ids parameter.
4. remove_taxonomy_ids (character varying) – **Required/Optional**. Comma separated list of taxonomy_ids to remove from relationships to financial_id(s). Not required if using classification_ids parameter.
5. edit_financial (enum) - Optional. Options:
    1. add (default) - will add a relationship between provided financial_id(s) and classification_id(s)
    2. delete - will remove the relationship between the provided financial_id(s) and classification_id(s)
    3. replace - will replace all relationships between the provided financial_ids(s) and taxonomy of the provided classification_id(s), with a relationship between the provided financial_id(s) and classification_id(s) per taxonomy.

Result

Boolean. Sucessfull (true) or unsuccessful (false). Unsuccessful is usually due invalid parameters or permissions.

Example(s)

- Add a relationship to Currency 'Tanzanian Shilling' (classification_id:419) to financial_ids 8061,8062,8063:

```
select * from
pmt_edit_financial_taxonomy(27,'8061,8062,8063','419',null,'add');
```

```
TRUE
```

- Remove the relationship to Currency 'Tanzanian Shilling' (classification_id:419) from financial_ids 8064,8065,8066,8067:

```
select * from
pmt_edit_financial_taxonomy(27,'8064,8065,8066,8067',null,'6','delete');
```

```
TRUE
```

- Replace all relationships to the Currency taxonomy 'Tanzanian Shilling' with the relationship to Currency 'Lebanese Pound' (classification_id:356) for financial_ids 8061,8062,8063:

```
select * from
pmt_edit_financial_taxonomy(27,'8061,8062,8063','356',null,'replace');
```

```
TRUE
```

- Remove all the relationships within the Currency (taxonomy_id:6) from financial_ids 8061,8062,8063:

```
select * from
pmt_edit_financial_taxonomy(27,'8061,8062,8063',null,'6','delete');
```

```
TRUE
```

pmt_edit_location ==========

Description

Edit a location.

Parameter(s)

1. user_id (integer) – **Required**. User_id of user requesting edits.
2. location_id (integer) – **Required for delete and update operations**. Location_id of location to update or delete.
3. activity_id (integer) – **Required for create operation**. Activity_id of the location to create.
4. json (json) - **Required for create and update operations**. Key/value pair as json of field/values to edit. Enclose all values in double quotes, including null. **If your text values include a single quote, use two adjacent single quotes, e.g., 'Dianne''s dog'. This is because PostgreSQL uses the single quote to encase string constants.** You can include any existing field that exists for location, even custom fields. The following fields cannot be edited even if included:
   - location_id
   - project_id

- activity_id
- x
- y
- lat_dd
- long_dd
- latlong
- georef
- active
- retired_by
- created_by
- created_date
- updated_by
- updated_date

Json key/value format:

```
{"column_name": "value"}
```

5. delete_record (boolean) - Optional, default is false. True to request the record be deleted.

Result

Json with the following:

1. id (integer) – location_id of the location created, updated or deleted.
2. message(character varying) - Message containing either "Success" for a successful transaction or containing an error description regarding the unsuccessful transaction. Some possible error messages:
   - The json parameter is required for a create/update operation
   - activity_id is required for a create operation - (When location_id is null a create operation is assumed an activity_id is required.)
   - location_id is required for a delete operation - (When delete_record is true, an location_id is required.)
   - user_id is a required parameter for all operations
   - User does NOT have authority to create a new location for activity_id
   - User does NOT have authority to delete this location
   - activity_id is not valid - (the provided activity_id is either not active or invalid)
   - User does NOT have authority to update this location

Example(s)

- Update the point geometry for location_id 81118.

```
select * from pmt_edit_location(3, 81118, null, '{"point":"POINT(4.921875
27.308333052145453)"}', false);
```

```
{"id":81118,"message":"Success"}
```

- Update an existing location to a boundary feature polygon (**Note: Since this is an existing point location, this will replace the current point with the centroid of the given boundary feature. If a location record has a boundary_id and feature_id the point is ALWAYS the centroid of the associated feature**).

```
select * from pmt_edit_location(3, 79564, null,
'{"boundary_id":3,"feature_id":25675}', false);
```

```
{"id":79564,"message":"Success"}
```

- Create a new location with title for activity_id 6.

```
select * from pmt_edit_location(3, null, 6, '{"title": "A village in Nepal",
"point":"POINT(39.0234375 6.9427857850946015)"}', false);
```

```
{"id":15821,"message":"Success"}
```

- Create a new location as a boundary feature polygon for activity_id 6 (**Note: You do not have to provide a point value as it is automatically created as the centroid of the provided polygon**).

```
select * from pmt_edit_location(3, null, 6,
'{"boundary_id":3,"feature_id":25675}', false);
```

```
{"id":15821,"message":"Success"}
```

- Delete location_id 81116.

```
select * from pmt_edit_location(3, 81116, null, null, true);
```

```
{"id":81116,"message":"Success"}
```

pmt_edit_location_taxonomy ==========

Description

Edit the relationship between a location(s) and a taxonomy classification(s). **Important Note: This function DOES NOT call the refresh_taxonomy_lookup() function, which updates the materialized views that support a majority of the functions in the database. Be sure to call this function when editing is complete.**

Parameter(s)

1. user_id (integer) – **Required**. User_id of user requesting edits.
2. location_ids (character varying) – **Required**. Comma seperated list of location_ids to edit.
3. classification_ids (character varying) – **Required/Optional**. Comma seperated list of classification_ids of taxonomy classifications in relationship to location_id(s). Not required if using remove_taxonomy_ids parameter.
4. remove_taxonomy_ids (charcter varying) - **Required/Optional**. Comma seperated list of taxonomy_ids to remove from relationships to location_id(s). Not required if using classification_ids parameter.
5. edit_location (enum) - Optional. Options:

1. add (default) - will add a relationship between provided location_id(s) and classification_id(s)
2. delete - will remove the relationship between the provided location_id(s) and classification_id(s)
3. replace - will replace all relationships between the provided location_ids(s) and taxonomy of the provided classification_id(s), with a relationship between the provided location_id(s) and classification_id(s) per taxonomy.

Result

Boolean. Successful (true) or unsuccessful (false). Unsuccessful is usually due invalid parameters or insufficient permissions.

Example(s)

- Add a relationship to Location Reach 'Intended Beneficiaries' (classification_id:975), Location Class 'Populated Place' (classification_id:971) and Location Class 'Ward (level-3)' (classification_id:1196) to location_ids 1470,1471,1472,1473:

```
select * from
pmt_edit_location_taxonomy(55,'1470,1471,1472,1473','975,971,1196',null,'add'
)
```

```
TRUE
```

- Remove the relationship to Location Reach 'Partners' (classification_id:976), Location Class 'River Basin' (classification_id:973) and Location Class 'District (level-2)' (classification_id:970) to location_ids 1470,1471,1472,1473:

```
select * from
pmt_edit_location_taxonomy(55,'1470,1471,1472,1473','976,970,973',null,'delet
e');
```

```
TRUE
```

- Replace all relationships to the Location Reach taxonomy with the relationship to Location Reach 'Action/intervention' (classification_id:974) and all the relationships to Location Class with the relationship to Location Class 'District (level-2)' (classification_id:970) for location_ids 1470,1471,1472,1473:

```
select * from
pmt_edit_location_taxonomy(55,'1470,1471,1472,1473','974,970','replace');
```

```
TRUE
```

- Remove the realtionships within the 'Country' taxonomy (taxonomy_id:5) for location_ids 1472,1473,1474,1475,1476,1477:

```
select * from
pmt_edit_location_taxonomy(55,'1472,1473,1474,1475,1476,1477',null,'5',null);
```

```
TRUE
```

pmt_edit_organization ==========

Description

Edit a organization.

Parameter(s)

1. user_id (integer) – **Required**. User_id of user requesting edits.
2. organization_id (integer) – **Required for update and delete operations**. organization_id of existing organization to update or delete, if left null then a new organization record will be created.
3. json (json) - **Required**. Key/value pair as json of field/values to edit. Column names are case sensitive. Enclose all values in double quotes, including null. **If your text values include a single quote, use two adjacent single quotes, e.g., 'Dianne''s dog'. This is because PostgreSQL uses the single quote to encase string constants.** You can include any existing field that exists for organization, even custom fields. The following fields cannot be edited even if included:
   - organization_id
   - active
   - retired_by
   - created_by
   - created_date
   - updated_by
   - updated_date

   Json key/value format:

   ```
   {"column_name": "value"}
   ```

4. delete_record (boolean) - Optional, default is false. True to request the record be deleted.

Result

Json with the following:

1. id (integer) – organization_id of the organization created or updated.
2. message(character varying) - Message containing either "Success" for a successful transaction or containing an error description regarding the unsuccessful transaction. Possible error messages:
   - Must include user_id and json data parameters - Received when both required parameters are not provided.
   - User does NOT have authority to create a new organization - Received when the user_id provided does not have authority to create under current role.
   - User does NOT have authority to update an existing organization - Received when the user_id provided does not have authority to update under current role.

- Invalid organization_id - Received when the organization_id provided is invalid.

Example(s)

- Update the email and url for CIAT (organization_id:25) as user sparadee (user_id:34)

```
select * from pmt_edit_organization(34,25,'{"email":"ciatk@mail.com",
"url":"www.ciat.org"}', false);

{"id":25,"message":"Success"}
```

- Add new organization as user sparadee (user_id:34)

```
select * from pmt_edit_organization(34,null,'{"name":"SpatialDev",
"url":"www.spatialdev.com",
"email":"info@spatialdev.com"}', false);

{"id":672,"message":"Success"}
```

- Delete organization_id 672 as user sparadee (user_id:34)

```
select * from pmt_edit_organization(34,672,null, true);

{"id":672,"message":"Success"}
```

pmt_edit_participation ==========

Description

Edit the relationship between projects, activities and organizations.

Parameter(s)

1. user_id (integer) – **Required**. User_id of user requesting edits.
2. participation_id (integer) – **Required when edit_activity option is delete**. participation_id of participation record to edit.
3. project_id (integer) – **Required for project participation when edit_activity option is add/replace**. Project_id of project that organization has participation in.
4. activity_id (integer) – **Required for activity participation when edit_activity option is add/replace**. Activity_id of activity that organization has participation in.
5. organiation_id (integer) – **Required when edit_activity option is add/replace***. Organiation_id of organization that is participating in the project/activity.
6. classification_ids (character varying) – **Required when edit_action option is add/replace***. Classification_ids from Organisation Role taxonomy that represents the organizations participation role in the project/activity.
7. edit_action (enum) - Optional. Options:
    1. add (default) - will add a participation record to project/activity.
    2. delete - will remove a participation record.
    3. replace - will replace all existing participation records with the new participation record for the project/activity **make sure you understand what this is doing before requesting this edit action***

Result

Json with the following:

1. id (integer) – organization_id of the organization created or updated.
2. message(character varying) - Message containing either "Success" for a successful transaction or containing an error description regarding the unsuccessful transaction.

Example(s)

- Add IFPRI (organization_id:519) as a Funding & Implementing (classification_id:496 & 497) organization for 'STAPLE CROPS PROGRAMME' project (project_id:463) activity 'P008-09-P1-08-008-Integrated Striga Management For Improved Sorghum Productivity In East And Central Africa' (activity_id:1653) as user sparadee (user_id:34):

```
select * from pmt_edit_participation(34, null, 463, 1653, 519, '496,497',
'add');
```

```
{"id":618,"message":"Success"}
```

- Delete IFPRI as an Implementing organization for 'STAPLE CROPS PROGRAMME' project activity 'P008-09-P1-08-008-Integrated Striga Management For Improved Sorghum Productivity In East And Central Africa'(participation_id:10708) as user sparadee (user_id:34):

```
select * from pmt_edit_participation(34, 10708, null, null, null, null,
'delete');
```

```
{"id":10708,"message":"Success"}
```

- Replace **ALL** participation records for 'STAPLE CROPS PROGRAMME' project (project_id:463) activity 'P008-09-P1-08-008-Integrated Striga Management For Improved Sorghum Productivity In East And Central Africa' (activity_id:1653) with IFPRI as an Implementing organization as user sparadee (user_id:34).

```
select * from pmt_edit_participation(34, null, 463, 1653, 519, '497',
'replace');
```

```
{"id":1058,"message":"Success"}
```

pmt_edit_participation_taxonomy ==========

Description

Edit the relationship between a participation and a taxonomy classification. **Important Note: This function DOES NOT call the refresh_taxonomy_lookup() function, which updates the materialized views that support a majority of the functions in the database. Be sure to call this function when editing is complete.**

Parameter(s)

1. user_id (integer) – **Required**. User_id of user requesting edits.

2. participation_id (integer) – **Required**. participation_id to edit.
3. classification_id (integer) – **Required**. Classification_id of taxonomy classification in relationship to the participation_id.
4. edit_action (enum) - Optional. Options:
    1. add (default) - will add a relationship between provided participation_id and classification_id
    2. delete - will remove the relationship between the provided participation_id and classification_id
    3. replace - will replace all relationships between the provided participation_ids and taxonomy of the provided classification_id, with a single relationship between the provided participation_id and classification_id.

Result

Boolean. Sucessfull (true) or unsuccessful (false). Unsuccessful is usually due invalid parameters or insufficient permissions.

Example(s)

- Add a relationship to Organisation Role 'Accountable' (classification_id:494) and 'Funding' (classification_id:496) to participation_ids 6003,6004:

```
select * from
pmt_edit_participation_taxonomy(55,'6003,6004','494,496','add');
```

```
TRUE
```

- Remove a relationship to Organisation Role 'Accountable' (classification_id:494) and 'Funding' (classification_id:496) for participation_ids 6003,6004:

```
select * from pmt_edit_participation_taxonomy(55,'6003,6004','494,496',
'delete');
```

```
TRUE
```

- Replace all relationships to the Organisation Role taxonomy with the relationship to Organisation Role 'Implementing' (classification_id:497) to participation_id 6003,6004:

```
select * from pmt_edit_participation_taxonomy(55,'6003,6004', '497',
'replace');
```

```
TRUE
```

pmt_edit_user

Description

Edit a user.

Parameter(s)

1. request_user_id (integer) – **Required**. User_id of user requesting edits.
2. target_user_id (integer) – **Required for delete and update operations**. User_id of user record to be updated or deleted.
3. json (json) - **Required for create and update operations**. Key/value pair as json of field/values to edit. Column names are case sensitive. Enclose all values in double quotes, including null. **If your text values include a single quote, use two adjacent single quotes, e.g., 'Dianne''s dog'. This is because PostgreSQL uses the single quote to encase string constants.** You can include any existing field that exists for user even custom fields. The following fields cannot be edited even if included:
   - id
   - _retired_by
   - _created_by
   - _created_date
   - _updated_by
   - _updated_date

   Json key/value format:

   ```
   {"column_name": "value"}
   ```

4. delete_record (boolean) - Optional, default is false. True to request the record be deleted.

Result

Json with the following:

1. id (integer) – project_id of the project created, updated or deleted.
2. message(character varying) - Message containing either "Success" for a successful transaction or containing an error description regarding the unsuccessful transaction. Some possible error messages:
   - Must include json parameter when delete_record is false.
   - User does NOT have authority to create a new user
   - User does NOT have authority to delete this user
   - User does NOT have authority to update this user
   - Invalid user_id.
   - Invalid organization_id;

Example(s)

- User updates their own email and password. Users can change any element of their user record with the exception of their role_id which determines thier database level permissions.

```
select * from pmt_edit_user(4,4, '{"email": "mymail@email.com",
"password":"Summ3r!"}', false);
```

```
{"id":4,"message":"Success"}
```

- Create a new user as user (user_id:3) with with either the Administrator or Super role. The following fields are required for a new user: email, username and password.

```
select * from pmt_edit_user(3,null,'{"email":"jhanhock@mail.com",
"username":"jhanhock1",
"password":"p@ssw0rd"}', false);
```

```
{"id":45,"message":"Success"}
```

- Delete user_id 45 as user (user_id:3) with with either the Administrator or Super role

```
select * from pmt_edit_user(3, 45, null, true);
```

```
{"id":45,"message":"Success"}
```

pmt_export

Description

Create an export dataset based on filters. Intended to be exported at csv using application or json to csv conversion. This function is intended to be the generic export function, but there may be custom export functions for specific data groups: - pmt_export_bmgf - pmt_export_tanaim - pmt_export_ethaim

Parameter(s)

1. data_group_ids (character varying) - comma seperated list of classification id(s) from the Data Group taxonomy to restrict data. If no data group id is provided, all data groups are included.
2. classification_ids (character varying) - comma seperated list of classification id(s) for any taxonomy (filter).
3. org_ids (character varying) - comma seperated list of organization id(s) for organizations for all participation types (filter).
4. imp_org_ids (character varying) - comma seperated list of organization id(s) for implementing organizations (filter).
5. fund_org_ids (character varying) - comma seperated list of organization id(s) for funding organizations (filter).
6. start_date (date) - start date for activities (filter).
7. end_date (date) - end date for activities (filter).
8. unassigned_taxonomy_ids (character varying) - comma seperated list of taxonomy id(s) for any taxonomy, will return activities that *DO NOT* have that taxonomy assigned (filter).

Result

A json object containing columns of data by row:

1. c1 (text) – first column of data

2. c2 (text) – second column of data
3. c3 (text) - third column of data ...
4. c18 (text) - the eighteenth column of data

Example(s)

- Data export for all data for World Bank (data_group_id:2210):

```
SELECT * FROM pmt_export('2210',null,null,null,null,null,null,null);

...
{
    "c1":"Activity Data",
    "c2":"PMT ActivityID",
    "c3":"Activity Title",
    "c4":"Activity Description",
    "c5":"Sector - Name",
    "c6":"Sector - Code",
    "c7":"Latitude Longitude",
    "c8":"Country",
    "c9":"Funding Organization(s)",
    "c10":"Implementing Organization(s)",
    "c11":"Start Date",
    "c12":"End Date",
    "c13":"Total Budget",
    "c14":"Activity Status",
    "c15":null,
    "c16":null,
    "c17":null,
    "c18":null
}
{
    "c1":"",
    "c2":"24586",
    "c3":"BF-Compet &amp;amp; Enterprise Dev (FY03)",
    "c4":"The Competitiveness and Enterprise Development Project for Burkina
Faso will assist Burkina Faso to improve the competitiveness of its economy
through privatization and utility reform (...)"
    "c5":"Agricultural development",
    "c6":"31120",
    "c7":"32.01524 -5.2584",
    "c8":"Tanzania",
    "c9":"World Bank",
    "c10":"Local NGOs",
    "c11":"2005-1-15",
    "c12":"2008-5-30",
    "c13":"1530250",
    "c14":"Complete",
    "c15":null,
    "c16":null,
    "c17":null,
```

```
    "c18":null
}
...
```

↩ Back to Function List

pmt_filter

Description

Accepts a number of filter parameters and returns a list of activity ids that have requested attributes.

Parameter(s)

1. data_group_ids (character varying) - comma seperated list of classification id(s) from the Data Group taxonomy to restrict data. If no data group id is provided, all data groups are included.
2. classification_ids (character varying) - comma seperated list of classification id(s) for any taxonomy (filter).
3. org_ids (character varying) - comma seperated list of organization id(s) for organizations for all participation types (filter).
4. imp_org_ids (character varying) - comma seperated list of organization id(s) for implementing organizations (filter).
5. fund_org_ids (character varying) - comma seperated list of organization id(s) for funding organizations (filter).
6. start_date (date) - start date for activities (filter).
7. end_date (date) - end date for activities (filter).
8. unassigned_taxonomy_ids (character varying) - comma seperated list of taxonomy id(s) for any taxonomy, will return activities that *DO NOT* have that taxonomy assigned (filter).

Result

Integer array of filtered activity ids. Only active activity ids will be returned.

Example(s)

- Filter activities to include only BMGF data (data_group_id: 768) where the Initative is Research & Development (classification_id: 831) or there is **NO** Initiative (taxonomy_id: 23) and Non-Governmental organizations (NGOs) (organization_id:1681) are participating in activities occuring between 1/1/2012 and 12/31/2018:

```
SELECT * FROM
pmt_filter('768','831','1681','','','1/1/2012','12/31/2018','23');
```

```
[2070,2071,2072,2073,2074,2077,2078,2094,2095,2108,12039]
```

↩ Back to Function List

pmt_filter_iati ==========

Description

Create and email a IATI formatted Activity xml file of data filtered by classification, organization and date range, reporting associated organization(s).

Parameter(s)

1. classification_ids (character varying) – Optional. Restrict data to classification(s).
2. organization_ids (character varying) – Optional. Restrict data to organization(s)
3. unassigned_tax_ids (character varying) – Optional. Include data without assignments to specified taxonomy(ies).
4. start_date (date) – Optional. Restrict data to a data range. Used with end_date parameter.
5. end_date (date) – Optional. Restrict data to a data range. Used with start_date parameter.
6. email (text) - **Required**. Email address to send the created csv to.

Result

A xml document of in the IATI Activity Schema.

Example(s)

- Data export for AGRA data group (classification_id:769):

```
SELECT * FROM pmt_filter_iati('769','','',null,null,
'sparadee@spatialdev.com');
```

```
TRUE
```

pmt_filter_string

Description

Accepts the same parameters as pmt_filter and returns a "pretty string" of selected filters.

Parameter(s)

1. data_group_ids (character varying) - comma seperated list of classification id(s) from the Data Group taxonomy to restrict data. If no data group id is provided, all data groups are included.
2. classification_ids (character varying) - comma seperated list of classification id(s) for any taxonomy (filter).
3. org_ids (character varying) - comma seperated list of organization id(s) for organizations for all participation types (filter).
4. imp_org_ids (character varying) - comma seperated list of organization id(s) for implementing organizations (filter).

5. fund_org_ids (character varying) - comma seperated list of organization id(s) for funding organizations (filter).
6. start_date (date) - start date for activities (filter).
7. end_date (date) - end date for activities (filter).
8. unassigned_taxonomy_ids (character varying) - comma seperated list of taxonomy id(s) for any taxonomy, will return activities that *DO NOT* have that taxonomy assigned (filter).

Result

A string.

Example(s)

- Filter activities to include only BMGF data (data_group_id: 768) where the Initative is Research & Development (classification_id: 831) or there is **NO** Initiative (taxonomy_id: 23) and Non-Governmental organizations (NGOs) (organization_id:1681) are participating in activities occuring between 1/1/2012 and 12/31/2018:

```
SELECT * FROM
pmt_filter_string('768','831','1681','','','1/1/2012','12/31/2018','23');
```

```
"PMT 3.0, Database Version 3.0.10.34, Retrieval Date: 2016-06-22, Filters:
Data Group: BMGF | Initiative: Research & Development | Unassigned
taxonomies: Initiative | Organization: Non-Governmental organizations (NGOs)
| DateRange: 2012-01-01 to 2018-12-31"
```

↩ Back to Function List

pmt_global_search

Description

Searches all text and character varying data type fields for the activity table for the search text.

Parameter(s)

1. search_text (text) – **Required**. Text string to search activity table.
2. data_group_ids (character varying) - comma delimited string of data group ids it restrict search to.

Result

Json with the following:

1. ids (integer[]) – array of activity ids.

Example(s)

- Search for the term 'wheat' in the RED&FS data (data_group_id: 2237):

```
SELECT * FROM pmt_global_search('wheat', '2237');
```

```
{
    "ids":[26255,26257,26259]
}
```

↩ Back to Function List

pmt_iati_import

Description

Imports an IATI Activities formatted xml document.

Parameter(s)

1. user_id (integer) **Required**. The user id of the user loading the xml document.
2. file_path (text) – **Required**. Path to IATI Activities formatted xml document.
3. file_encoding (text) – **Required**. The encoding of the xml document. The encoding should be an attribute on the xml element. Must use the Postgres equivalent for the encoding.
4. data_group_name (character varying) - **Required**. Name of the data group. If data group does not exist it will be created.

Result

True (success) or false (unsuccessful).

Example(s)

```
SELECT * FROM pmt_iati_import(34, '/usr/local/pmt_iati/BoliviaIATI.xml',
'utf-8', 'Bolivia');
```

TRUE

↩ Back to Function List

pmt_is_data_group

Description

Validates a data group name or data group classification_id. Data group is a taxonomy that is used to determine data source and provide user authentication.

Parameter(s)

1. name (character varying) – any Data Group name to be tested as valid. ** OR **
2. classification_id (integer) - any integer to be tested as a valid classification_id in the Data Group taxonomy.

Result

True or false.

Example(s)

```
SELECT * FROM pmt_is_datagroup('Charlie Chocolate');
```

FALSE

```
SELECT SELECT * FROM pmt_is_datagroup(768);
```

TRUE

↩ Back to Function List

pmt_isdate

Description

Validates a text value for date data type

Parameter(s)

1. (text) – any text value to be tested.

Result

True or false.

Example(s)

```
SELECT pmt_isdate('14-1-2012');
```

FALSE

```
SELECT pmt_isdate('2012-1-13');
```

TRUE

↩ Back to Function List

pmt_isnumeric

Description

Validates a text value for numeric data type

Parameter(s)

1. (text) – any text value to be tested.

Result

True or false.

Example(s)

```
SELECT pmt_isnumeric('');
```

FALSE

```
SELECT pmt_isnumeric(null);
```

TRUE

↩ Back to Function List

pmt_locations

Description

All information for one or more locations.

Parameter(s)

1. location_ids (character varying) - **Required.** Comma delimited list of location_ids.

Result

Json with the following:

1. id (integer) – location id.
2. activity_id (integer) – activity id of location.
3. boundary_id (integer) – boundary id of location's associated boundary layer.
4. feature_id (integer) – feature id of location's associated boundary feature.
5. _title (character varying) – title of location.
6. _description (character varying) – description of location.
7. _x (numeric) – x coordinate.
8. _y (numeric) – y coordinate.
9. _lat_dd (numeric) – latitude decimal degrees.
10. _long_dd (numeric) – longitude decimal degrees.
11. _latlong (character varying) – latitude and longitude.
12. _georef (character varying) – geo-reference format.
13. _updated_by (character varying(50)) - last user to update activity information.
14. _updated_date (timestamp) - last date and time activity information was updated.
15. _custom_fields (various) - any custom fields in the activity table that are not in the Core PMT will be returned as well.
16. taxonomy(object) - An object containing all associated taxonomy for the activity
    1. taxonomy_id (integer) - taxonomy id.
    2. taxonomy (character varying) - taxonomy name.
    3. classification_id (integer) - classification id.
    4. classification (character varying) - classification name.
    5. _code (character varying) - classification code.

17. point (object) - An object containing geoJson representation of the point feature
18. polygon (object) - An object containing geoJson representation of the associated polygon feature

Example(s)

```
select * from pmt_locations('79564,39489');
```


```
{
    "id":79564
    ,"activity_id":80
    ,"_title":null
    ,"_description":null
    ,"_x":1496805
    ,"_y":1833818
    ,"_lat_dd":16.251080
    ,"_long_dd":13.44603
    ,"_latlong":"16°15'4\"N 13°26'46\"E
    ,"_georef":"NHPB26461504"
    ,"_updated_by":"super"
    ,"_updated_date":"2014-05-05 22:40:42.741879"
    ,"boundary_id":3,
    ,"feature_id":25675,
    ,"taxonomy":[
        {
        "taxonomy_id":5
        ,"taxonomy":"Country"
        ,"classification_id":185
        ,"classification":"NIGER"
        ,"_code":"NE"
        },{
        "taxonomy_id":25
        ,"taxonomy":"Location Class"
        ,"classification_id":970
        ,"classification":"Administrative Region"
        ,"_code":"1"
        },{
        "taxonomy_id":26
        ,"taxonomy":"Location Reach"
        ,"classification_id":974
        ,"classification":"Action/intervention"
        ,"_code":"101"
        }]
    ,"point":{
        "type":"Point"
        ,"coordinates":[13.4460309287303,16.2510804045115]
        }
    ,"polygon":{
        "type":"MultiPolygon"
        ,"coordinates":[[[
```

```
[15.5595092773437,18.0062866210938],[15.556884765625,17.9564819335938],

[15.5551147460937,17.9263305664062],[15.5543212890625,17.9204711914062],

[15.5535278320312,17.9146728515625],[15.5535278320312,17.8839111328125],
            [ (...)]]]"
        }
```

↩ Back to Function List

pmt_locations_for_boundaries

Description

Calculates and returns the counts for activities and locations for all features in a requested boundary.

Parameter(s)

1. boundary_id (integer) - **Required**. boundary_id of the boundary layer to aggregate location and activity counts to.
2. data_group_ids (character varying) - comma seperated list of classification_id(s) from the Data Group taxonomy to restrict data aggregation to. If no data group id is provided, all data groups are included.
3. classification_ids (character varying) - comma seperated list of classification_id(s) for any taxonomy (filter).
4. org_ids (character varying) - comma seperated list of organization_id(s) for any organization regardless of role (filter).
5. imp_org_ids (character varying) - comma seperated list of organization_id(s) for implementing organizations (filter).
6. fund_org_ids (character varying) - comma seperated list of organization_id(s) for funding organizations (filter).
7. start_date (date) - start date for activities (filter).
8. end_date (date) - end date for activities (filter).
9. unassigned_taxonomy_ids (character varying) - comma seperated list of taxonomy id(s) for any taxonomy, will return activities that *DO NOT* have that taxonomy assigned (filter).
10. activity_ids (character varying) - comma seperated list of activity id(s) to restrict data aggregation to.
11. boundary_filter (json) - a json array of objects. Each object must contain "b" with a boundary id and "ids" with an array of feature ids (i.e. `[{"b":12,"ids":[2,3]},{"b":13,"ids":[73,85]}]`)

Result

Json with the following:

1. id (integer) – feature_id of the feature within the requested boundary.
2. p (integer) - total number of parent activities within feature.
3. a (integer) – total number of activities within feature.
4. l (integer) – total number of locations within feature.
5. b (integer) – boundary_id of the boundary feature is associated to.

Example(s)

- Aggregate location/activity counts to the continent boundary (boundary_id: 8) for BMGF data (data_group_id: 768) where Activity Status is Complete (classification_id: 797) and activities occur between 1/1/2012 and 12/31/2018:

```
SELECT * FROM
pmt_locations_for_boundaries(8,'768','797',null,null,'1/1/2012','12/31/2018',
null,null);
```


```
{
    "id":1,
    "p":1,
    "a":1,
    "l":1,
    "b":8
},{
    "id":4,
    "p":3,
    "a":66,
    "l":66,
    "b":8
}
```


- Aggregate location/activity counts to the UNOCHA administrative level 1 boundary (boundary_id: 12) for RED&FS data (data_group_id: 2237) where Program is "Stand Alone Project" (classification_id: 2239) and activities occur between 1/1/2002 and 12/31/2020 within the Oromia region (boundary_id: 12 feature_id: 8) or in the Majang zone (boundary_id: 13 feature_id: 38) within the Gambela region :

```
SELECT * FROM
pmt_locations_for_boundaries(12,'2237','2239',null,null,'1/1/2002','12/31/202
0',null,'[{"b":12,"ids":[8]},{"b":13,"ids":[38]}]');
```


```
{
    "id":1,
    "a":3,
    "l":3,
    "b":12
},
...
```

```
{
    "id":10,
    "a":15,
    "l":80,
    "b":12
},
{
    "id":11,
    "a":20,
    "l":49,
    "b":12
}
```


↩ Back to Function List

pmt_org_inuse

Description

Organizations participating in activities.

Parameter(s)

1. data_group_ids (character varying) – Optional. Classification_id(s) from the Data Group taxonomy. Restrict data to data groups(s).
2. org_role_ids (character varying) – Optional. Classification_id(s) from the Organisation Role taxonomy. Restrict data to Organisation Role(s).

Result

Ordered by most used. Json with the following:

1. id (integer) – organization id.
2. n (character varying(255)) – name of organization.
3. ct (integer) - number of activities organization participates in (can order by ct for most active organizations).
4. o (character varying(1)) - the first letter of the organization name (can use o for an alphabetical lookup).

Example(s)

- Implementing organizations (classification_id: 497) participating in activities in the BMGF data group (classification_id:768):

```
SELECT * FROM pmt_org_inuse('768','497');
```


```
{   "id":270,
    "n":"International Institute of Tropical Agriculture (IITA)",
    "ct":695,
    "o":"i"
}
```

```
{
    "id":5,
    "n":"TechnoServe",
    "ct":525,
    "o":"t"
}
{
    "id":22,
    "n":"Agricultural Cooperative Development International and Volunteers in
Overseas Cooperative Assistance (ACDI/VOCA)",
    "ct":508,
    "o":"a"
}
...
```


↩ Back to Function List

pmt_orgs

Description

Get all organizations.

Parameter(s)

No parameters.

Result

Json with the following:

1. id (integer) – id of organization.
2. _name (character varying(255)) – name of organization.

Example(s)

```
SELECT * FROM pmt_orgs();
```


```
...
{
    "id":32,
    "_name":"CARE International"
}
...
```


↩ Back to Function List

pmt_overview_stats

Description

Function to calculate overview statistics.

Parameter(s)

1. data_group_ids (character varying) - comma seperated list of classification_id(s) from the Data Group taxonomy to restrict data. If no data group id is provided, all data groups are included (filter).
2. classification_ids (character varying) - comma seperated list of classification_id(s) for any taxonomy (filter).
3. start_date (date) - start date for activities (filter).
4. end_date (date) - end date for activities (filter).
5. boundary_id (integer) - the boundary id referenced by the feature_id (filter).
6. feature_ids (character varying) - comma seperated list of featuer ids for the boundary to restrict activities to (filter).

Result

Json with the following:

1. activity_count (integer) - total number of activities
2. implmenting_count (integer) - total number of implementing organizations
3. total_investment (integer) - total amount of invested money for activities
4. country_count (integer) - total number of countries where activities have locations

Example(s)

- Overview statistics for BMGF data (data_group_id: 768) in Ethiopia (feature_id: 74) and Tanzania (feature_id: 227) using the GADM boundary (boundary_id: 15):

```
SELECT * FROM pmt_overview_stats('768',null,null,null,15,'74,227');
```

```
{
    "activity_count":117,
    "implmenting_count":252,
    "total_investment":1576921373.00,
    "country_count":2
}
```

↩ Back to Function List

pmt_partner_pivot

Description

Function to create a filterable pivot table showing organization represation within two taxonomies (row and column) based on a single organization role.

Parameter(s)

1. row_taxonomy_id (integer) - **Required** taxonomy id for taxonomy that will represent rows (y axis of pivot).
2. column_taxonomy_id (integer) - **Required** taxonomy id for taxonomy that will represent columns (x axis of pivot).
3. org_role_id (integer) - **Required** classification id for the Organisation Role to be used to select participating organizations (table data).
4. data_group_ids (character varying) - comma seperated list of classification_id(s) from the Data Group taxonomy to restrict data. If no data group id is provided, all data groups are included (filter).
5. classification_ids (character varying) - comma seperated list of classification_id(s) for any taxonomy (filter).
6. start_date (date) - start date for activities (filter).
7. end_date (date) - end date for activities (filter).
8. boundary_id (integer) - the boundary id referenced by the feature_id (filter).
9. feature_id (integer) - the featuer id to restrict activities to (filter).

Result

Json with the following:

1. c# (character varying) - each object will contain a value for each column in a row numbered 1 - x
2. f1 (character varying) - organization label
3. f2 (integer) - organization id
4. f3 (character varying) organization name

Example(s)

- Partner pivot table data to include only EthATA data (data_group_id: 2237) where Program is the row/y-axis (taxonomy id: 68) and Crops and Livestock is the column/x-axis (taxonomy id: 69) and the organizations represented are Funding (classification id: 496) for activities in the Oromia region (feature_id: 869) of Ethiopia using GADM boundaries (boundary_id: 16):

```
SELECT * FROM pmt_partner_pivot(68,69,496,'2237',null,null,null,16,896);
```


```
{
    "c1":"",
    "c2":"Barley",
    "c3":"Cactus",
    "c4":"Coffee",
    "c5":"Fruit",
    "c6":"Horticulture",
    "c7":"Livestock",
    "c8":"Potato",
    "c9":"Rice",
    "c10":"Sericulture",
```

```
    "c11":"Wheat",
    "c12":"Unspecified"
}
{
    "c1":"Agricultural Growth Program",
    "c2":[null],
    "c3":[null],
    "c4":[null],
    "c5":[null],
    "c6":[null],
    "c7":[null],
    "c8":[null],
    "c9":[null],
    "c10":[null],
    "c11":[null],
    "c12":[
        {
            "f1":"CIDA",
            "f2":3135,
            "f3":"Canadian International Development Agency (CIDA)"
        },
        {
            "f1":"Global Agriculture and Food Security Program",
            "f2":2588,
            "f3":"Global Agriculture and Food Security Program"
        },

        {
            "f1":"GoE",
            "f2":3148,
            "f3":"Government of Ethiopia (GoE)"
        }
    ]
},
{
    "c1":"Concern Livlihoods Program",
    "c2":[null],
    "c3":[null],
    "c4":[null],
    "c5":[null],
    "c6":[null],
    "c7":[null],
    "c8":[null],
    "c9":[null],
    "c10":[null],
    "c11":[null],
    "c12":[
        {
            "f1":"Concern Ethiopia",
            "f2":3166,
```

```
            "f3":"Concern Ethiopia"
        },

        {
            "f1":"EU",
            "f2":1027,
            "f3":"European Union (EU)"
        },

        {
            "f1":"WRDA",
            "f2":3222,
            "f3":"Wonnta Rural Development Association (WRDA)"
        }
    ]
},
{
    "c1":"Food Security Program",
    "c2":[null],
    "c3":[null],
    "c4":[null],
    "c5":[null],
    "c6":[null],
    "c7":[null],
    "c8":[null],
    "c9":[null],
    "c10":[null],
    "c11":[null],
    "c12":[
        {
            "f1":"CIDA",
            "f2":3135,
            "f3":"Canadian International Development Agency (CIDA)"
        },

        {
            "f1":"DEFID",
            "f2":3170,
            "f3":"DEFID"
        },

        {
            "f1":"Department of Foreign and International Affairs",
            "f2":3171,
            "f3":"Department of Foreign and International Affairs"
        }
    ]
},
{
```

```
"c1":"Stand Alone Project",
"c2":[
    {
        "f1":"GDC",
        "f2":3149,
        "f3":"German Development Cooperation (GDC)"
    }
],
"c3":[
    {
        "f1":"ITAL",
        "f2":3100,
        "f3":"Italian Development Cooperation (ITAL)"
    }
],
"c4":[
    {
        "f1":"EU",
        "f2":1027,
        "f3":"European Union (EU)"
    },
    {
        "f1":"ITAL",
        "f2":3100,
        "f3":"Italian Development Cooperation (ITAL)"
    },

    {
        "f1":"UNDP",
        "f2":3093,
        "f3":"United Nations Development Program (UNDP)"
    }
],
"c5":[null],
"c6":[null],
"c7":[null],
"c8":[null],
"c9":[null],
"c10":[null],
"c11":[null],
"c12":[
    {
        "f1":"ADA",
        "f2":3163,
        "f3":"Austrian Development Agency (ADA)"
    }
]
```


Results can then be rendered as a table:

	Barley	Cactus	Coffee	Fruit	Horticulture	Livestock	Potato	Rice	Sericulture	Wheat	Unspecified
Agricultural Growth Program											CIDA, Global Agriculture and Food Security Program, GoE, Local, NETH, SPAN, undetermined, USAID, WB
Food Security Program											CIDA, DEFID, Department of Foreign and International Affairs, EC, GoE, IREAID, NETH, SWED, USAID, WB, WFP
Sustainable Land Management Program											CIDA, DED, EU, FIN, GIZ, IFAD, KFW, MASHAV, WB
Stand Alone Project	GDC	ITAL	EU, ITAL, UNDP	ITAL	ITAL	EU, GoE, OCHA, USAI	USAID	CIDA	GoE	GDC, ITAL	ADA, African Development Bank,

	D	Belgium Cooperation, Christian Aid, CIDA, DANIDA, Department of Foreign and International Affairs, EU, FIN, GIZ, GoE, IFAD, IIRR, International Rescue Committee, IREAID, ITAL, JICA, Korea, Local, MEDA, NETH
Concern Livlihoods Program		Concern Ethiopia, EU, WRDA

↩ Back to Function List

pmt_partner_sankey

Description

Function specifically for reporting data in the D3 Sankey data format using nodes and links for the partnerlink tool. Function accepts filtering parameters.

Parameter(s)

1. data_group_ids (character varying) - comma seperated list of classification_id(s) from the Data Group taxonomy to restrict data. If no data group id is provided, all data groups are included.
2. classification_ids (character varying) - comma seperated list of classification_id(s) for any taxonomy (filter).
3. org_ids (character varying) - comma seperated list of organization_id(s) for organizations for all participation types (filter).
4. start_date (date) - start date for activities (filter).
5. end_date (date) - end date for activities (filter).
6. unassigned_taxonomy_ids (character varying) - comma seperated list of taxonomy id(s) for any taxonomy, will return activities that *DO NOT* have that taxonomy assigned (filter).

Result

Json with the following:

1. nodes (object) – Json object.
    1. name (string) - name of organization (*node levels 0-2*) or activity title (*node level 3*).
    2. node (numeric) - unique identifier for node using a numeric pattern. Whole number represents the organization_id and the decimal represents the node level (i.e. 13.1 is organization_id 13 at node level 1 (*Grantee*)).
    3. level (integer) - node level (0-3). Each node level represents a participation role or the activity:
        1. 0-Funder (*Organisation Role Accountable*)
        2. 1-Grantee (*Organisation Role Funding*)
        3. 2-Partner (*Organisation Role Implementing*)
        4. 3-Activity
2. links (object) – Json object.
    1. source (numeric) - the source node id.
    2. source_level (integer) - the source node level.
    3. target (numeric) - the target node id.
    4. target_level (integer) - the target node level.
    5. link (string) - a concatenated text representation of the source/target relationship (source node id + _ + target node id).
    6. value (integer) - count of activities.

Example(s)

- Partner Sankey data to include only BMGF data (data_group_id: 768) where the Initative is Research & Development (classification_id: 831) and activities occure between 1/1/2012 and 12/31/2018:

```
SELECT * FROM
pmt_partner_sankey('768','831',null,'1/1/2012','12/31/2018',null);
```


```
{
    "nodes": [{
        "name":"Partner Not Reported"
        ,"node":3.2
        ,"level":2
        },{
        "name":"Kickstart International"
        ,"node":10.0
        ,"level":0
        },{
        "name":"Kickstart International"
        ,"node":10.2
        ,"level":2
        },{
        "name":"Bill & Melinda Gates Foundation (BMGF)"
        ,"node":13.1
        ,"level":1
        },{
        ...
        },{
        "name":"GAAP-Kickstart-Impact of Kickstart Treadle Pumps in East
Africa"
        ,"node":14968.3
        ,"level":3
        }],
    "links":[{
        "source":495
        ,"source_level":0
        ,"target":13.1
        ,"target_level":1
        ,"link":"495_13.1"
        ,"value":584
        },{
        "source":367
        ,"source_level":0
        ,"target":13.1
        ,"target_level":1
        ,"link":"367_13.1"
        ,"value":207
        },{
        "source":1025
        ,"source_level":0
        ,"target":13.1
        ,"target_level":1
        ,"link":"1025_13.1"
        ,"value":167
        },{
```

```
        ...
        },{
        "source":1674.2
        ,"source_level":2
        ,"target":3173.3
        ,"target_level":3
        ,"link":"1674.2_3173.3"
        ,"value":1
        }]
}
```


↩ Back to Function List

pmt_partner_sankey_activities

Description

Function specifically for requesting applicable activities for a given organization in the partnerlink.

Parameter(s)

1. data_group_ids (character varying) - **Required** comma seperated list of classification_id(s) from the Data Group taxonomy to restrict data.
2. organization (character varying) - **Required** the organization name that is participating in the activities.
3. partnerlink_level (integer) - **Required** the partnerlink node level that the organization is in (options: 0,1,2).

Result

Json with the following:

1. activity_id (integer) – the activity id.
2. title (character varying) – the activity title

Example(s)

- Activities in the African Development Bank data (data_group_id: 2209) where the African Development Fund is in the first node (Funder):

```
SELECT * FROM pmt_partner_sankey_activities('2209','African Development
Fund',0);
```


```
{
    "activity_id":23951,
    "title":"Projet d'appui Ñ la Bonne Gouvernance et Ñ la
Decentralisation"
},
{
    "activity_id":23963,
    "title":"Appui Institutionnel  Ñ Quatres MinistÑres"
```

```
},
{
    "activity_id":24110,
    "title":"Anyianam-Kumasi Road Rehabilitation Project"
},
...
{
    "activity_id":24520,
    "title":"Strengthening of science and technical teacher of the school of
education"
}
```


↩ Back to Function List

pmt_purge_activities

Description

Deletes all records assoicated to an activity, including all **children** activities. **Warning!! This function permanently deletes ALL data associated to the given activity ids.** Use pmt_activate_activity to deactivate if deletion is not desired.

Parameter(s)

1. a_id (integer[]) – **Required**. Array of activity ids to be deleted.

Result

Boolean. True/False successful.

Example(s)

- Remove activity id 101, 102, 103 & 104:

```
SELECT * FROM pmt_purge_activities([101,102,103,104]);
```

TRUE

pmt_purge_activity

Description

Deletes all records assoicated to an activity, including all **children** activities. **Warning!! This function permanently deletes ALL data associated to the given activity id.** Use pmt_activate_activity to deactivate if deletion is not desired.

Parameter(s)

1. a_id (integer) – **Required**. Id of the activity to be deleted.

Result

Boolean. True/False successful.

Example(s)

- Remove activity id 101:

```
SELECT * FROM pmt_purge_activity(101);
```

TRUE

↩ Back to Function List

pmt_stat_activity_by_tax

Description

Statistics function providing filterable investments and activity counts by a taxonomy.

Parameter(s)

1. taxonomy_id (integer) – **Required**. the taxonomy id to classify returned activity investments and counts.
2. data_group_ids (character varying) - comma seperated list of classification id(s) from the Data Group taxonomy to restrict data. If no data group id is provided, all data groups are included.
3. classification_ids (character varying) - comma seperated list of classification id(s) for any taxonomy (filter).
4. start_date (date) - start date for activities (filter).
5. end_date (date) - end date for activities (filter).
6. boundary_id (integer) - the boundary id referenced by the feature_id (filter).
7. feature_id (integer) - the feature id to restrict activities to (filter).
8. record_limit (integer) - the number of records to limit return to, remaining classifications will be aggregated into an "other" classification.

Result

Json with the following:

1. id (integer) – the classification id for the given taxonomy.
2. classification (character varying) - the classification name for the given taxonomy.
3. count (integer) - the number of activities assigned to the classification.
4. sum (numeric) - the investment for the assigned classification.

Example(s)

- Activity investments and counts for all of BMGF (data_group_id:768) for the Initiative taxonomy (taxonomy_id:23) in Ethiopia (feature_id: 74) using GADM boundary (boundary_id: 15):

```
SELECT * FROM pmt_stat_activity_by_tax(23,'768',null,null,null,15,74,null);
```


```
{
    "classification_id":831,
```

```
    "classification":"Research & Development",
    "count":57,
    "sum":100000.00
},
{
    "classification_id":null,
    "classification":"Unspecified",
    "count":2146,
    "sum":500000.00
},
{
    "classification_id":2213,
    "classification":"Country & Policy",
    "count":2,
    "sum":null
},
{
    "classification_id":2212,
    "classification":"Farmer Systems & Services",
    "count":389,
    "sum":null
},
{
    "classification_id":829,
    "classification":"Other",
    "count":1,
    "sum":null
}
```


↩ Back to Function List

pmt_stat_by_org

Description

Statistics function providing filterable activity counts by organization and role, ordered by greatest activity count.

Parameter(s)

1. data_group_ids (character varying) - comma seperated list of classification id(s) from the Data Group taxonomy to restrict data. If no data group id is provided, all data groups are included.
2. classification_ids (character varying) - comma seperated list of classification id(s) for any taxonomy (filter).
3. start_date (date) - start date for activities (filter).
4. end_date (date) - end date for activities (filter).
5. org_role_id (integer) - the organization role to filter data to (filter).
6. boundary_id (integer) - the boundary id referenced by the feature_id (filter).

7. feature_id (integer) - the featuer id to restrict activities to (filter).
8. limit_records (integer) - number of records to return.

Result

Json with the following:

1. id (integer) – the organization id.
2. name (character varying) – the organization name.
3. label (character varying) – the organization name's abbreviation or shortened name.
4. role (character varying) – the organization's role.
5. activity_count (integer) – the number of activities the organization is involved in.

Example(s)

- Top 5 implmenting (org_role_id: 497) organizations for BMGF & AGRA data groups (data_group_id: 769,768) in Ethiopia (feature_id: 74) using GADM boundary (boundary_id: 15):

```
SELECT * FROM pmt_stat_by_org('769,768',null,null,null,497,15,74,5);
```


```
{
    "id":9,
    "name":"United Nations Food and Agriculture Organization (FAO)",
    "label":"FAO",
    "role":"Implementing",
    "activity_count":16
},
{
    "id":12,
    "name":"U.S. Agency for International Development (USAID)",
    "label":"USAID",
    "role":"Implementing",
    "activity_count":1
},
{
    "id":22,
    "name":"Agricultural Cooperative Development International and Volunteers
in Overseas Cooperative Assistance (ACDI/VOCA)",
    "label":"ACDI/VOCA",
    "role":"Implementing",
    "activity_count":6
},
{
    "id":221,
    "name":"Kenya Agricultural Research Institute (KARI)",
    "label":"KARI",
    "role":"Implementing",
    "activity_count":1
},
```

```
{
    "id":270,
    "name":"International Institute of Tropical Agriculture (IITA)",
    "label":"IITA",
    "role":"Implementing",
    "activity_count":1
}
```


↩ Back to Function List

pmt_stat_invest_by_funder

Description

Statistics function providing filterable investments by funding organization, ordered by greatest investment amount.

Parameter(s)

1. data_group_ids (character varying) - comma seperated list of classification id(s) from the Data Group taxonomy to restrict data. If no data group id is provided, all data groups are included.
2. classification_ids (character varying) - comma seperated list of classification id(s) for any taxonomy (filter).
3. start_date (date) - start date for activities (filter).
4. end_date (date) - end date for activities (filter).
5. boundary_id (integer) - the boundary id referenced by the feature_id (filter).
6. feature_id (integer) - the featuer id to restrict activities to (filter).
7. limit_records (integer) - number of records to return.

Result

Json with the following:

1. id (integer) – the funding organization id.
2. name (character varying) – the funding organization name.
3. label (character varying) – the funding organization name's abbreviation or shortened name.
4. count (integer) – the number of activities funded by the organization.
5. sum (numeric) – the total amount invested by the funding organization.
6. a_ids (integer[]) - a list of activity ids funded by organization.

Example(s)

- Top 5 investments by funding organization for EthATA data group (data_group_id:2237) in Ethiopia (feature_id: 74) using GADM boundary (boundary_id: 15):

```
SELECT * FROM pmt_stat_invest_by_funder('2237',null,null,null,15,74,5);
```


```
{
    "id":3112,
    "name":"World Bank (WB)",
    "label":"WB",
    "count":10,
    "sum":1030620000.00,
    "a_ids":[26271,26283,26284,26286,26287,26288,26322,26323,26324,26325]
},
{
    "id":3073,
    "name":"United States Agency for International Development (USAID)",
    "label":"USAID",
    "count":22,
    "sum":751774975.00,

"a_ids":[26284,26285,26287,26303,26304,26305,26306,26307,26308,26309,26310,26
311,26312,26313,26314,26315,26316,26317,26318,26319,26320,26321]
},
{
    "id":3175,
    "name":"European Commission (EC)",
    "label":"EC",
    "count":1,
    "sum":360000000.00,
    "a_ids":[26284]
},
{
    "id":3135,
    "name":"Canadian International Development Agency (CIDA)",
    "label":"CIDA",
    "count":20,
    "sum":309649285.00,

"a_ids":[26200,26201,26202,26203,26204,26205,26206,26207,26208,26209,26210,26
211,26212,26280,26283,26284,26285,26287,26326]
},
{
    "id":3170,
    "name":"DEFID",
    "label":"DEFID",
    "count":1,"sum":282300000.00,
    "a_ids":[26284]
}
```


↩ Back to Function List

pmt_statistic_data

Description

Provides statistics for a given indicator.

Parameter(s)

1. indicator_id (integer) – **Required**. Id for the target indicator found in stats_metadata.
2. code (character varying) – Optional. Restrict data to a specific country (ISO 3 digit codes).

Result

Json with the following:

1. indicator (character varying) – the name of the requested indicator.
2. boundary (character varying) – the name of the boundary for which the statistics apply.
3. _2000 (numeric) - the statistic value for the year 2000.
4. _2001 (numeric) - the statistic value for the year 2001.
5. _2002 (numeric) - the statistic value for the year 2002.
6. _2003 (numeric) - the statistic value for the year 2003.
7. _2004 (numeric) - the statistic value for the year 2004.
8. _2005 (numeric) - the statistic value for the year 2005.
9. _2006 (numeric) - the statistic value for the year 2006.
10. _2007 (numeric) - the statistic value for the year 2007.
11. _2008 (numeric) - the statistic value for the year 2008.
12. _2009 (numeric) - the statistic value for the year 2009.
13. _2010 (numeric) - the statistic value for the year 2010.
14. _2011 (numeric) - the statistic value for the year 2011.
15. _2012 (numeric) - the statistic value for the year 2012.
16. _2013 (numeric) - the statistic value for the year 2013.
17. _2014 (numeric) - the statistic value for the year 2014.
18. _2015 (numeric) - the statistic value for the year 2015.
19. _2016 (numeric) - the statistic value for the year 2016.

Example(s)

- Poverty gap at $1.90 a day (2011 PPP) (%) (indicator_id:1733) for Tanzania (code: TZA):

```
SELECT * FROM pmt_statistic_data(1733,'TZA');
```


```
{
    "indicator":"Poverty gap at $1.90 a day (2011 PPP) (%)",
    "boundary":"Tanzania",
    "_2000":44.54,
```

```
    "_2001":null,
    "_2002":null,
    "_2003":null,
    "_2004":null,
    "_2005":null,
    "_2006":null,
    "_2007":18.95,
    "_2008":null,
    "_2009":null,
    "_2010":null,
    "_2011":14.35,
    "_2012":null,
    "_2013":null,
    "_2014":null,
    "_2015":null,
    "_2016":null
}
```


↩ Back to Function List

pmt_statistic_indicators

Description

Lists available statistic indicators.

Parameter(s)

1. code (character varying) – Optional. Restrict data to a specific country (ISO 3 digit codes).

Result

Json with the following:

1. _category (character varying) – the category for the indicator.
2. _sub_categories (json[]) – object array containing sub categories
    1. _sub_category (character varying) – name of the sub category
    2. indicators (json[]) – object array of indicators
        1. id (integer) - the indicator id
        2. _name (character varying) - the name of the indicator

Example(s)

- Get all statistic indicators for Kenya (code: KEN):

```
SELECT * pmt_statistic_indicators('KEN');
```


```
{
    "_category":"Private Sector & Trade",
    "sub_categories":[
        {
```

```
            "_sub_category":"Tariffs",
            "indicators":[
                {
                    "id":3028,
                    "_name":"Share of tariff lines with international peaks,
all products (%)"
                },
                {
                    "id":3027,
                    "_name":"Bound rate, simple mean, all products (%)"
                },
                ...
                {
                    "id":1641,
                    "_name":"Share of tariff lines with international peaks,
all products (%)"
                }
            ]
        },
        ...
        {
            "_sub_category":"Exports",
            "indicators":[
                {
                    "id":3081,
                    "_name":"Merchandise exports to developing economies in
Europe & Central Asia (% of total merchandise exports)"
                },
                {
                    "id":3071,
                    "_name":"Fuel exports (% of merchandise exports)"
                }
                ...
            ]
        }
    ]
}
...
{
    "_category":"Poverty",
    "sub_categories":[
        {
            "_sub_category":"Conflict & fragility"
            ,"indicators":[
                {
                    "id":1717,
                    "_name":"Combined polity score"
                }
                ...
            ]
```

```
        }
    ]
}
```

↩ Back to Function List

pmt_tax_inuse ==========

Description

Taxonomy and associated classifications that are in use by any project, activity or location.

Parameter(s)

1. data_group_id (integer) – Optional. Restrict data to data group.
2. taxonomy_ids (character varying) – Optional. Restrict data to taxonomy(ies).
3. country_ids (character varying) – Optional. Restrict data to country(ies).

Result

Ordered by most used. Json with the following:

1. t_id (integer) – taxonomy_id.
2. name (character varying(255)) – name of taxonomy.
3. Is_cat (boolean) – is/not a taxonomy category.
4. cat_id (integer) – taxonomy_id of the taxonomy category for this taxonomy.
5. classifications (object) – classifications in use for this taxonomy.
    1. c_id (integer) – classification_id.
    2. cat_id (integer) – classification_id for the category classification.
    3. name (character varying(255)) – the name of the classification.

Example(s)

- Taxonomy/classifications for the World Bank data group (classification_id:772) in Bolivia (classification_id:50):

```
SELECT * FROM pmt_tax_inuse(772, '', '50');

...
{
    "t_id":15,
    "name":"Sector",
    "is_cat":false,
    "cat_id":14,
    "classifications":
        [
            {
                "c_id":731,
                "cat_id":552,
                "name":"Desarrollo rural"
```

```
            },
            {
                "c_id":636,
                "cat_id":540,
                "name":"Power generation/renewable sources"
            },
            ...
        ]
},
{
    "t_id":14,
    "name":"Sector Category",
    "is_cat":true,
    "cat_id":16,
    "classifications":
        [
            {
                "c_id":552,
                "cat_id":765,
                "name":"Other multisector"
            },
            ...
            {
                "c_id":540,
                "cat_id":764,
                "name":"ENERGY GENERATION AND SUPPLY"},
            ...
        ]
}
```


pmt_taxonomies ==========

Description

Taxonomy and associated classifications.

Parameter(s)

1. taxonomy_ids (character varying) – Optional. Restrict data to taxonomy(ies).

Result

Ordered by most used. Json with the following:

1. t_id (integer) – taxonomy_id.
2. name (character varying(255)) – name of taxonomy.
3. Is_cat (boolean) – is/not a taxonomy category.
4. cat_id (integer) – taxonomy_id of the taxonomy category for this taxonomy.
5. classifications (object) – classifications in use for this taxonomy.
    1. c_id (integer) – classification_id.

2. cat_id (integer) – classification_id for the category classification.
3. name (character varying(255)) – the name of the classification.

Example(s)

- Taxonomy/classifications for the Sector taxonomy (taxonomy_id:15):

```
select * from pmt_taxonomies('15');

...
{
    "t_id":15,
    "name":"Sector",
    "is_cat":false,
    "cat_id":14,
    "classifications":
        [
        {
        "c_id":636,
        "cat_id":540,
        "name":"Power generation/renewable sources",
        "code":23030
        },
        {
        "c_id":657,
        "cat_id":542,
        "name":"Privatisation",
        "code": 53030
        }
            ...
        ]
}
```

pmt_update_crosswalks

Description

Remove and recreate activity taxonomy crosswalk data for a data group.

Parameter(s)

1. data_group_id (integer) - **Required**. Classification id from the Data Group taxonomy.

Result

Success boolean (True/False).

Example(s)

- Update the crosswalk data for the BMGF data group (data_group_id: 768).

```
SELECT * FROM pmt_update_crosswalks(768);

TRUE
```

↩ Back to Function List

pmt_user_auth

Description

Authenticate user.

Parameter(s)

1. username (character varying) - **Required**. User's username.
2. password (character varying) - **Required**. User's password.

Result

Json with the following:

1. user_id (integer) – user id.
2. first_name (character varying) – first name of user.
3. last_name (character varying) – last name of user.
4. username (character varying) – username of user.
5. email (character varying) – email address of user.
6. organization_id (integer) – organization id for organization of user.
7. organization (character varying) – organization name for organization of user.
8. role_id (integer) – database role id user is assigned to.
9. role (character varying) – name of database role user is assigned to.
10. authorizations (json object) – list of project_ids by user's project role.
    1. role_id (integer) – user's role id on the projects in project_ids
    2. role (character varying) – user's role name on the projects in project_ids
    3. project_ids (integer array) - array of project_ids users is authorized to access with permissions provided by the role.

Example(s)

- Authenticate Reader test user passing its username and password.

```
SELECT * FROM pmt_user_auth('johndoe', 'password');
```


```
{
    "id":1,
    "_first_name":"John",
    "_last_name":"Doe",
    "_username":"johndoe",
    "_email":"test@email.com",
    "organization_id":3,
    "organization":"spatial dev",
    "role_id":4,
    "role":"Administrator",
    "role_auth":
```

```
    {
        "_read":true,
        "_create":true,
        "_update":true,
        "_delete":true,
        "_super":true,
        "_security":true
    },
    "authorizations":
    [{
        "role_id":3,
        "role":"Super",
        "activity_ids":[37]
    },{
        "role_id":4,
        "role":"Administrator",

"activity_ids":[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,2
4,25,26,27,28,29,30,31,32,33,34,35,36]
    }]
}
```


- Authenticate Reader test user passing its username and an invalid password.

```
SELECT * FROM pmt_user_auth('reader', 'bad password');
```


```
{
    "message":"Invalid username or password."
}
```


↩ Back to Function List

pmt_user_salt

Description

Get the salt for a specific user.

Parameter(s)

1. username (text)– **Required**. Username for requested salt value.

Result

Salt value as text.

Example(s)

- Salt value for test user Reader(johndoe):

```
select * from pmt_user_salt('johndoe');
```


```
$2a$10$.V0ETMIAW6O9z2wekwMG1.
```

↩ Back to Function List

pmt_user ==========

Description

Get all user information for a single user.

Parameter(s)

1. user_id (integer) – **Required**. User id for user.

Result

Json with the following:

1. user_id (integer) – user id.
2. first_name (character varying) – first name of user.
3. last_name (character varying) – last name of user.
4. username (character varying) – username of user.
5. email (character varying) – email address of user.
6. organization_id (integer) – organization id for organization of user.
7. organization (character varying) – organization name for organization of user.
8. role_id (integer) – database role id user is assigned to.
9. role (character varying) – name of database role user is assigned to.
10. created_by (character varying) - first and last name OR username (if first name is null) of the user who created the user record.
11. projects (json object):
    1. project_id (integer) – project id.
    2. title (character varying) – title of project.
    3. role_id (integer) – project role id user is assigned to.
    4. role (character varying) – name of users role on project.

Example(s)

```
SELECT * FROM pmt_user(5);

...
{
    "user_id":315,
    "first_name":"Jane",
    "last_name":"Doe",
    "username":"myusername",
    "email":"jane.doe@email.com",
    "organization_id":13,
    "organization": "BMGF",
    "role_id":1,
    "role":"Reader",
    "created_by":"adminbob",
```

```
    "projects":[{
            "project_id":15
            ,"title":"Policies, Institutions, and Markets"
            ,"role_id":2
            ,"role":"Editor"
        },{
            "project_id":16
            ,"title":"Agriculture for Nutrition and Health"
            ,"role_id":5
            ,"role":"Publisher"
        }],
    "active":true
}
...
```

pmt_users

Description

Get all user and role information.

Parameter(s)

No parameters.

Result

Json with the following:

1. id (integer) – user id.
2. _first_name (character varying) – first name of user.
3. _last_name (character varying) – last name of user.
4. _username (character varying) – username of user.
5. _email (character varying) – email address of user.
6. organization_id (integer) – organization id for organization of user.
7. organization (character varying) – organization name for organization of user.
8. role_id (integer) – database role id user is assigned to.
9. role (character varying) – name of database role user is assigned to.
10. _access_date (timestamp without time zone) - most recent user login timestamp.
11. _active (boolean) - t/f user account is active.

Example(s)

```
SELECT * FROM pmt_users();

...
{
    "id":315,
    "_first_name":"Jane",
```

```
    "_last_name":"Doe",
    "_username":"myusername",
    "_email":"jane.doe@email.com",
    "organization_id":13,
    "organization": "BMGF",
    "role_id":1,
    "role":"Reader",
    "_access_date": "2014-05-21T23:29:27.497825",
    "_active":true
}
...
```

↩ Back to Function List

pmt_validate_activities

Description

Validate list of activity ids.

Parameter(s)

1. activity_ids (character varying) - **Required**. comma separated list of activity ids to validate.

Result

Integer array of valid ACTIVE activity ids.

Example(s)

```
SELECT * FROM pmt_validate_activities('11879,15432,15725,122');
```

integer[]

{11879,15432,15725}

↩ Back to Function List

pmt_validate_activity ==========

Description

Validate an activity_id.

Parameter(s)

1. activity_id (integer) - **Required**. activity_id to validate.

Result

Boolean. True/False valid.

Example(s)

```
SELECT * FROM pmt_validate_activity(11879);
```

TRUE

pmt_validate_boundary_feature

Description

Validate an boundary_id and feature_id combination.

Parameter(s)

1. boundary_id (integer) - **Required**. boundary_id to validate.
2. feature_id (integer) - **Required**. feature_id to validate.

Result

Boolean. True/False valid.

Example(s)

```
SELECT * FROM pmt_validate_boundary_feature(1, 23);
```

TRUE

↩ Back to Function List

pmt_validate_classification

Description

Validate a classification id.

Parameter(s)

1. classification_id (integer) - **Required**. classification id to validate.

Result

Boolean. True/False valid.

Example(s)

```
SELECT * FROM pmt_validate_classification(768);
```

TRUE

↩ Back to Function List

pmt_validate_classifications

Description

Validate list of classification ids.

Parameter(s)

1. classification_ids (character varying) - **Required**. comma separated list of classification ids to validate.

Result

Integer array of valid ACTIVE classification ids in ascending order.

Example(s)

```
SELECT * FROM pmt_validate_classifications('50,9999,720');
```

integer[]
{50,720}

↩ Back to Function List

pmt_validate_contact ==========

Description

Validate a contact_id.

Parameter(s)

1. contact_id (integer) - **Required**. contact_id to validate.

Result

Boolean. True/False valid.

Example(s)

```
SELECT * FROM pmt_validate_contact(169);
```

TRUE

pmt_validate_contacts ==========

Description

Validate list of contact_ids.

Parameter(s)

1. contact_ids (character varying) - **Required**. comma separated list of contact_ids to validate.

Result

Integer array of valid ACTIVE contact_ids.

Example(s)

```
SELECT * FROM pmt_validate_contacts('169,145,9999');
```

integer[]
{145,169}

pmt_validate_detail ==========

Description

Validate a detail_id.

Parameter(s)

1. detail_id (integer) - **Required**. detail_id to validate.

Result

Boolean. True/False valid.

Example(s)

```
SELECT * FROM pmt_validate_detail(169);
```

TRUE

pmt_validate_financial ==========

Description

Validate a financial_id.

Parameter(s)

1. financial_id (integer) - **Required**. financial_id to validate.

Result

Boolean. True/False valid.

Example(s)

```
SELECT * FROM pmt_validate_financial(19);
```

TRUE

pmt_validate_location

Description

Validate a location_id.

Parameter(s)

1. location_id (integer) - **Required**. location_id to validate.

Result

Boolean. True/False valid.

Example(s)

```
SELECT * FROM pmt_validate_location(19);
```

TRUE

↩ Back to Function List

pmt_validate_locations

Description

Validate list of location_ids.

Parameter(s)

1. location_ids (character varying) - **Required** Comma separated list of location_ids to validate.

Result

Integer array of valid ACTIVE location_ids.

Example(s)

```
SELECT * FROM pmt_validate_locations('9,12,15');
```

integer[]
{9,12,15}

↩ Back to Function List

pmt_validate_organization

Description

Validate a organization id.

Parameter(s)

1. organization_id (integer) - **Required**. organization id to validate.

Result

Boolean. True/False valid.

Example(s)

```
SELECT * FROM pmt_validate_organization(13);
```

TRUE

↩ Back to Function List

pmt_validate_organizations ==========

Description

Validate list of organization_ids.

Parameter(s)

1. organization_ids (character varying) - **Required** Comma separated list of organization_ids to validate.

Result

Integer array of valid ACTIVE organization_ids.

Example(s)

```
SELECT * FROM pmt_validate_organizations('13,27,2');
```

integer[]
{13,27}

pmt_validate_participation ==========

Description

Validate a participation_id.

Parameter(s)

1. participation_id (integer) - **Required**. participation_id to validate.

Result

Boolean. True/False valid.

Example(s)

```
SELECT * FROM pmt_validate_participation(57789);
```

TRUE

pmt_validate_participations ==========

Description

Validate list of participation_ids.

Parameter(s)

1. participation_ids (character varying) - **Required**. comma separated list of participation_ids to validate.

Result

Integer array of valid ACTIVE participation_ids.

Example(s)

```
SELECT * FROM pmt_validate_participations('18416,57789,34331,2');
```

integer[]

{2,57789}

pmt_validate_role ==========

Description

Validate an role_id.

Parameter(s)

1. role_id (integer) - **Required**. role_id to validate.

Result

Boolean. True/False valid.

Example(s)

```
SELECT * FROM pmt_validate_role(1);
```

TRUE

pmt_validate_taxonomies ==========

Description

Validate list of taxonomy_ids.

Parameter(s)

1. taxonomy_ids (character varying) - **Required**. comma separated list of taxonomy_ids to validate.

Result

Integer array of valid ACTIVE taxonomy_ids.

Example(s)

```
SELECT * FROM pmt_validate_taxonomies('5,10,99');
```

integer[]
{5,10}

pmt_validate_taxonomy

Description

Validate a taxonomy id.

Parameter(s)

1. id (integer) - **Required**. taxonomy id to validate.

Result

Boolean. True/False valid.

Example(s)

```
SELECT * FROM pmt_validate_taxonomy(1);
```

TRUE

↩ Back to Function List

pmt_validate_user ==========

Description

Validate an user_id.

Parameter(s)

1. user_id (integer) - **Required**. user_id to validate.

Result

Boolean. True/False valid.

Example(s)

```
SELECT * FROM pmt_validate_user(1);
```

TRUE

pmt_validate_user_authority

Description

Validate a user's authority on an activity or the database and the level of authority (CRUD).

Parameter(s)

1. user_id (integer) - **Required**. user_id of user to validate authority for.
2. activity_id (integer) - activity_id to validate authority for user to edit. If left blank then the function assumes you are validating CRUD action on database (adding new activities).
3. auth_type (enum) - **Required**. authority type. Options:
    1. create - user has ability to create new records.
    2. read - user has ability to read records.
    3. update - user has ability to update existing records.
    4. delete - user has ability to delete records.

Result

Boolean. True/False user has authority on database or activity with authority type.

Example(s)

```
select * from pmt_validate_user_authority(34, 422, 'update');
```

TRUE

↩ Back to Function List

pmt_version

Description

Provides the current version, iteration, changeset, instance creation date and last changeset update date.

Parameter(s)

No parameters.

Result

Json with the following:

1. pmt_version (text) – the version, iteration, changeset.
2. last_update (date) – the date of last changeset.
3. created (date) – the date instance was created.

Example(s)

```
SELECT * FROM pmt_version();
```

```
{
    "pmt_version":"3.0.10.16",
    "last_update":"2016-03-31",
    "created":"2014-02-05"
}
```

test_execute_unit_tests

Description

Execute all unit tests.

Parameter(s)

No parameters.

Result

Text. Results message.

Example(s)

- Execute tests:

```
SELECT * FROM test_execute_unit_tests();
```

Unit testing complete: pass (32) fail (0) execution_failures(0) total(32)

- View testing results:

```
SELECT * FROM unit_test;
```

↩ Back to Function List

---

[1] Douglas, Jack. "SQL to read XML from file into PostgreSQL database." StackExchange Database Administrators Nov 2011. Web. 02 Aug 2013 http://dba.stackexchange.com/questions/8172/sql-to-read-xml-from-file-into-postgresql-database