Site_Grouping_Module

October 31, 2017

1 Phase II - Tool Creation

1.1 Document database

1.1.1 Bike-Ped metadata tables

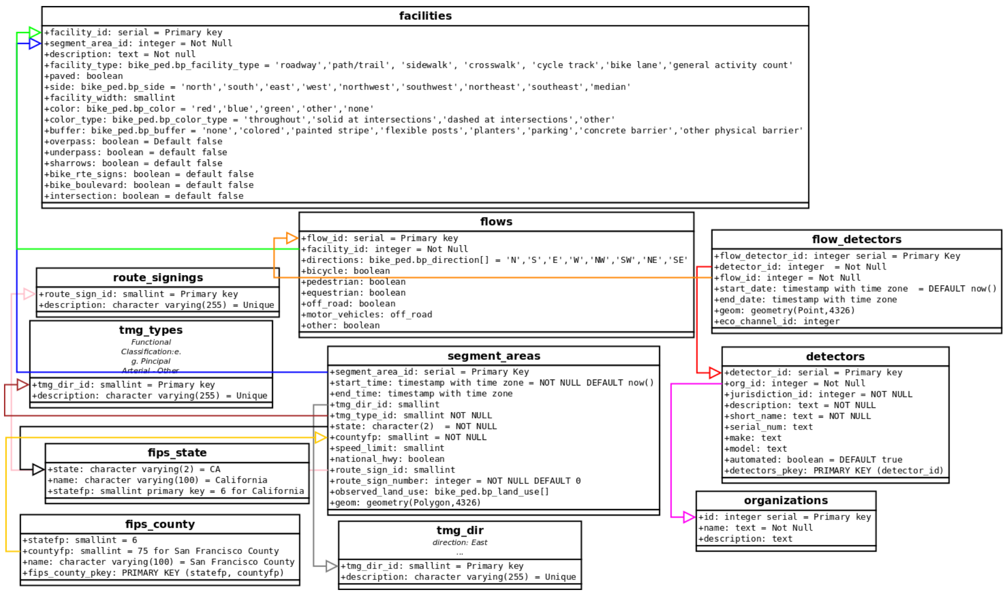

facilities
+facility_id: serial = Primary key
+segment_area_id: integer = Not Null
+description: text = Not null
+facility_type: bike_ped.bp_facility_type = 'roadway','path/trail', 'sidewalk', 'crosswalk', 'cycle track','bike lane','general activity count'
+paved: boolean
+side: bike_ped.bp_side = 'north','south','east','west','northwest','southwest','northeast','southeast','median'
+facility_width: smallint
+color: bike_ped.bp_color = 'red','blue','green','other','none'
+color_type: bike_ped.bp_color_type = 'throughout','solid at intersections','dashed at intersections','other'
+buffer: bike_ped.bp_buffer = 'none','colored','painted stripe','flexible posts','planters','parking','concrete barrier','other physical barrier'
+overpass: boolean = Default false
+underpass: boolean = default false
+sharrows: boolean = default false
+bike_rte_signs: boolean = default false
+bike_boulevard: boolean = default false
+intersection: boolean = default false
flows
+flow_id: serial = Primary key
+facility_id: integer = Not Null
+directions: bike_ped.bp_direction[] = 'N','S','E','W','NW','SW','NE','SE'
+bicycle: boolean
+pedestrian: boolean
+equestrian: boolean
+off_road: boolean
+motor_vehicles: off_road
+other: boolean
flow_detectors
+flow_detector_id: integer serial = Primary Key
+detector_id: integer = Not Null
+flow_id: integer = Not Null
+start_date: timestamp with time zone = DEFAULT now()
+end_date: timestamp with time zone
+geom: geometry(Point,4326)
+eco_channel_id: integer
route_signings
+route_sign_id: smallint = Primary key
+description: character varying(255) = Unique
tmg_types
Functional Classification:e.g. Pincipal Arterial - Other
+tmg_dir_id: smallint = Primary key
+description: character varying(255) = Unique
segment_areas
+segment_area_id: serial = Primary Key
+start_time: timestamp with time zone = NOT NULL DEFAULT now()
+end_time: timestamp with time zone
+tmg_dir_id: smallint
+tmg_type_id: smallint NOT NULL
+state: character(2) = NOT NULL
+countyfp: smallint = NOT NULL
+speed_limit: smallint
+national_hwy: boolean
+route_sign_id: smallint
+route_sign_number: integer = NOT NULL DEFAULT 0
+observed_land_use: bike_ped.bp_land_use[]
+geom: geometry(Polygon,4326)
detectors
+detector_id: serial = Primary key
+org_id: integer = Not Null
+jurisdiction_id: integer = NOT NULL
+description: text = NOT NULL
+short_name: text = NOT NULL
+serial_num: text
+make: text
+model: text
+automated: boolean = DEFAULT true
+detectors_pkey: PRIMARY KEY (detector_id)
fips_state
+state: character varying(2) = CA
+name: character varying(100) = California
+statefp: smallint primary key = 6 for California
organizations
+id: integer serial = Primary key
+name: text = Not Null
+description: text
fips_county
+statefp: smallint = 6
+countyfp: smallint = 75 for San Francisco County
+name: character varying(100) = San Francisco County
+fips_county_pkey: PRIMARY KEY (statefp, countyfp)
tmg_dir
direction: East
...
+tmg_dir_id: smallint = Primary key
+description: character varying(255) = Unique

Figure 1. Bike-Ped tables

1.2 Site Grouping Module

1.2.1 Choose a region

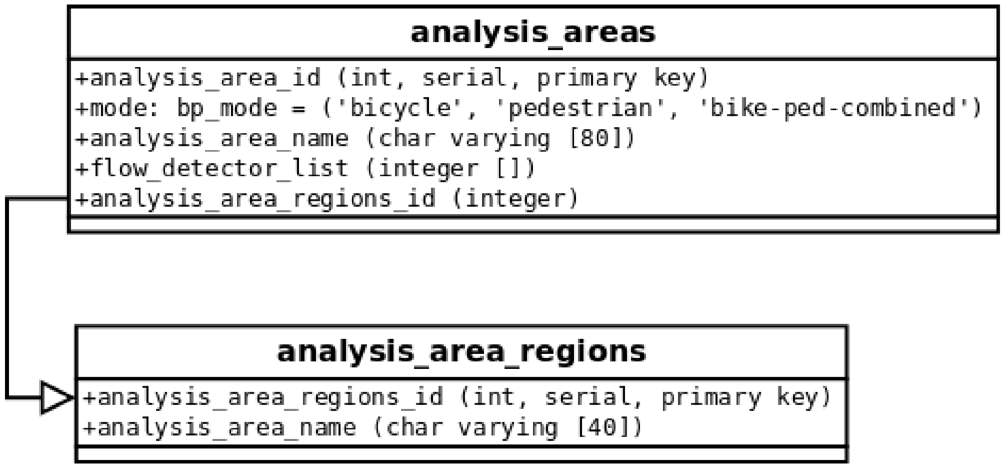
- listing existing regions in the database

```
In [4]: from utility import db_connect, query2csv
        from settings import  DBNAME, DBPASS, DBUSER, DBHOST

        qsql="""
        select analysis_area_regions_id, analysis_area_name
        from baa.analysis_area_regions
        """
        result_csv='analysis_area_regions.csv'
        query2csv(qsql,result_csv)

<IPython.core.display.HTML object>
```

1.2.2 Analysis tables



analysis_areas
+analysis_area_id (int, serial, primary key)
+mode: bp_mode = ('bicycle', 'pedestrian', 'bike-ped-combined')
+analysis_area_name (char varying [80])
+flow_detector_list (integer [])
+analysis_area_regions_id (integer)
analysis_area_regions
+analysis_area_regions_id (int, serial, primary key)
+analysis_area_name (char varying [40])

Figure 2. Analysis area tables

- If user is choosing a flow detector in a region not yet in the analysis_area_regions table, we need to create that region first.
- Assume the new region is San Francisco, we will either create a new one or retrieve its ID if it is in the database already.

```
In [4]: from utility import db_connect, query2csv
        from settings import  DBNAME, DBPASS, DBUSER, DBHOST

        def get_or_create_analysis_region(region_name):
            get_query = """
                select
                  analysis_area_regions_id
                from
                  baa.analysis_area_regions
                where
                  analysis_area_name = '{0}'
            """.format(region_name)

            cols = 'analysis_area_name'
            place_holder = '%s'
            query_str = """insert into baa.analysis_area_regions({0}) values ({1})
            RETURNING analysis_area_regions_id""".format(cols, place_holder)
            conn = db_connect()
            with conn:
                with conn.cursor() as curs:
                    curs.execute(get_query)
                    rows = curs.fetchall()
                    if(len(rows) == 1):
                        return rows[0][0]

                    curs.execute(query_str,(
                         region_name,
                    ))
                    res = curs.fetchone()
                    last_inserted_region_id = res[0]
            return last_inserted_region_id

        region_name='San Francisco'
        region_id = get_or_create_analysis_region(region_name)
        print('region id for {0} is {1}'.format(region_name, region_id))
```

```
region id for San Francisco is 7
```

1.2.3 Select a set of flow detectors as an analysis area

Inputs for the selection of an analysis area

- a region
- choose a mode which would be one of bicycle, pedestrian, or bike-ped-combined
- function **get_flow_detector_by_mode** to get list of flow detectors based on mode

```
In [25]: def get_flow_detector_by_mode(is_bike, is_ped):
             get_query = """
                 select flow_detector_id from
                 bike_ped.flow_detectors where flow_id in (
                 select
                   flow_id
                 from
                   bike_ped.flows
                 where
                   bicycle = %s
                   and pedestrian = %s
                 )
             """

             conn = db_connect()
             with conn:
                 with conn.cursor() as curs:
                     curs.execute(get_query, (is_bike, is_ped))
```

```
                    rows = curs.fetchall()
                    return [row[0] for row in rows]
```

- function usuage

```
In [26]: # return all flow detectors of bicycle mode
         flow_detector_list_bicycle = get_flow_detector_by_mode(True, False)
         # return all flow detectors of pedestrian mode
         flow_detector_list_pedestrian = get_flow_detector_by_mode(False, True)
         # return all flow detectors of bike_ped_combined mode
         flow_detector_list_bike_ped_combined=get_flow_detector_by_mode(True, True)
```

- a set of flow detectors that satisfy above two requirements
- if such analysis area exist return its ID other wise create one and return the ID

1.2.4 Get or create an analysis area

- Assume our region is **Portland**
- Assume the mode for the analysis area is **bicycle**
- List all flow detectors satisfy above assumptions

```
In [28]: from utility import db_connect, query2csv
         from settings import  DBNAME, DBPASS, DBUSER, DBHOST

         qsql="""
         select
           analysis_area_id,
           analysis_area_name,
           mode,
           flow_detector_list,
           analysis_area_regions_id
         from
           baa.analysis_areas
         where
           analysis_area_regions_id in
                (select analysis_area_regions_id
                 from baa.analysis_area_regions
                 where analysis_area_name = 'Portland')
           and mode = 'bicycle'
         """
         csvfile='analysis_areas_bicycle_Portland.csv'
         query2csv(qsql,csvfile)
```

1.2.5 Function to group a site: get_or_create_analysis_area

- User selected flow detector **1293** and **1295** at **Woodland Trail E of Fones Rd SE**
- We will call the function **get_or_create_analysis_area** with following inputs
- region: **San Diago**

- mode: **bicycle**
- flow_detector_list array: **(1293, 1295)**, should be among return list of flow detectors from function **get_flow_detector_by_mode(is_bike=True, is_ped=False)**
- analysis area name: **SD-Woodland-Trail-E-Fones-Rd-SE**
- The function will return an analysis area ID either existed in the database or newly created

```
In [30]: from utility import db_connect, query2csv
         from settings import  DBNAME, DBPASS, DBUSER, DBHOST

         def get_or_create_analysis_area(region_name, region_id, mode,
                                         flow_detector_list, analysis_area_name):
             get_query = """
                 select
                   analysis_area_id
                 from
                   baa.analysis_areas
                 where
                   flow_detector_list = %s
                   and mode = %s
                   and analysis_area_regions_id =
                   ( select
                       analysis_area_regions_id
                     from
                       baa.analysis_area_regions
                     where analysis_area_name = %s
                   )
             """

             cols='mode,analysis_area_name,flow_detector_list,analysis_area_regions
             place_holder = '%s::baa.bp_mode, %s, %s::integer[],%s'
             query_str = """insert into baa.analysis_areas ({0}) values ({1})
             RETURNING analysis_area_id""".format(cols, place_holder)
             conn = db_connect()
             with conn:
                 with conn.cursor() as curs:
                     curs.execute(get_query, (flow_detector_list,mode,region_name))
                     rows = curs.fetchall()
                     if(len(rows) == 1):
                         return rows[0][0]
                     curs.execute(query_str,(
                           mode,
                           analysis_area_name,
                           flow_detector_list,
                           region_id,
                     ))
                     res = curs.fetchone()
                     last_inserted_region_id = res[0]
             return last_inserted_region_id
```

```
        region_name='San Diago'
        region_id = 3
        mode = 'bicycle'
        flow_detector_list = [1293,1295]
        analysis_area_name = 'SD-Woodland-Trail-E-Fones-Rd-SE'
        analysis_area_id =get_or_create_analysis_area(region_name,
                                                       region_id, mode,
                                                       flow_detector_list,
                                                       analysis_area_name)
        print('The analysis_area_id for "%s", mode:%s and \nflow detectors:%s is %
              % (region_name, mode, ','.join(str(x) for x in flow_detector_list),
                analysis_area_id))
```

```
The analysis_area_id for "San Diago", mode:bicycle and
flow detectors:1293,1295 is 233
```

1.2.6 get aggregated daily volume of analysis area 233

```
In [31]: from utility import db_connect, query2csv
         from settings import  DBNAME, DBPASS, DBUSER, DBHOST

         qsql="""
         with hrly_233 as (
         select
           baaa.analysis_area_id,
           date_trunc('day', bpd.start_time) as date,
           to_char(bpd.start_time, 'HH24') as hour,
           sum(bpd.volume) as volume
         from
           baa.analysis_areas as baaa
             inner join baa_ex_sus.data as bpd
               on bpd.flow_detector_id = Any(baaa.flow_detector_list::int[])
         where
           baaa.analysis_area_id = 233
           group by analysis_area_id, bpd.start_time
         ),
         daily_233 as (
         select
           analysis_area_id,
           date_trunc('day', date) as date,
           sum(volume) as volume
         from
         hrly_233
         group by analysis_area_id, date_trunc('day', date)
         )
         select * from daily_233
```

```
        order by date
        """
        csvfile='daily_volume_analysis_areas_233.csv'
        query2csv(qsql,csvfile)
```

1.2.7 WWI module

- calculate the WWI of an analysis area

```
In [15]: from utility import db_connect, query2csv
         from settings import  DBNAME, DBPASS, DBUSER, DBHOST

         def calculate_wwi(analysis_area_id):
             query = """
                 with d as (
                   select generate_series(0,6) as dayofweek
                 ),
                 m as (
                   select generate_series(1,12) as month
                 ),
                 hrly as (
                 select
                   baaa.analysis_area_id,
                   date_trunc('day', bpd.start_time) as date,
                   to_char(bpd.start_time, 'HH24') as hour,
                   sum(bpd.volume) as volume
                 from
                   baa.analysis_areas as baaa
                     inner join baa_ex_sus.data as bpd
                     on bpd.flow_detector_id = Any(baaa.flow_detector_list::int[])
                 where
                   baaa.analysis_area_id = {0}
                   group by analysis_area_id, bpd.start_time
                 ),
                 daily as (
                 select
                   analysis_area_id,
                   date_trunc('day', date) as date,
                   sum(volume) as volume
                 from
                 hrly
                 group by analysis_area_id, date_trunc('day', date)
                 ) ,
                 -- v_ijmy:Compute an average by day of week for each month.
                 v_ijmy as (
                   select
                       baadv.analysis_area_id,
                       to_char(baadv.date, 'YYYY') as year,
                       avg(baadv.volume)::bigint as volume_i,
                       avg(baadv.volume) as volume,
                       d.dayofweek,
                       m.month
                   from
                       daily as baadv,
                       d,
                       m
                   where
                       extract(dow from baadv.date) in (d.dayofweek)
                       AND date_part('month', baadv.date) = m.month
                       group by baadv.analysis_area_id, year, d.dayofweek, m.month
                 ),
                 -- madt: average volume each month, each year for sites
                 madt as (
                   select
                       analysis_area_id,
                       month,
                       year,
                       avg(volume)::bigint as volume_i,
                       avg(volume) as volume
                   from
                       v_ijmy
                       group by analysis_area_id, year, month
                       having count(dayofweek)=7 --having 7 days of data each week
                 ),
                 AADT as (
                 select
                   analysis_area_id,
                   year,
                   avg(volume)::bigint as AADT_i,
                   round(avg(volume), 2) as AADT
                 from madt
                   group by analysis_area_id, year
                   having count(month) = 12 -- having 12 months of data
                 ),
                 -- daily_exclude_holiday: daily counts for sites excluding holiday
                 daily_exclude_holiday as (
                 select
                  baaad.analysis_area_id,
                  baaad.date,
                  baaad.volume,
                  date_part('month', baaad.date) as month,
                  date_part('dow', baaad.date) as dow
                 from
                   daily as baaad
                   left join baa.holidays as baahd
                     on baaad.date::date = baahd.holiday_date
                 where
                   baahd.holiday_id is null
                   group by 1,2,3
                 ),
                 V_jmyl_exclude_holiday as (
                   select
                       baadv.analysis_area_id,
                       to_char(baadv.date, 'YYYY') as year,
                       avg(baadv.volume) as volume,
                       d.dayofweek,
                       m.month
                   from
                       daily_exclude_holiday as baadv,
                       d,
                       m
                   where
                     extract(dow from baadv.date) in (d.dayofweek)
                     AND date_part('month', baadv.date) = m.month
                     group by baadv.analysis_area_id, year, d.dayofweek, m.month
                 ),
                 -- 84 factors volume count should exclude holiday weeks
                 factor84 as (
                 select
                   v_jmyl_nh.analysis_area_id,
                   v_jmyl_nh.volume as v_jmyl,
                   AADT.aadt as aadt,
                   round(v_jmyl_nh.volume/aadt::numeric, 2) as f_jmys,
                   v_jmyl_nh.dayofweek,
                   v_jmyl_nh.month,
                   v_jmyl_nh.year
                 from
                   V_jmyl_exclude_holiday as v_jmyl_nh inner join AADT
                     using(analysis_area_id, year)
                 where
                   AADT.AADT <> 0
                 ),
                 -
                 select wwi,weekly_group from wwi
             """.format(analysis_area_id)
             conn = db_connect()
             with conn:
                 with conn.cursor() as curs:
                     curs.execute(query)
                     rows = curs.fetchall()
                     if rows[0]:
                         return (rows[0][0], rows[0][1])
```

1.2.8 Calculate WWI for the just created analysis area 233

```
In [21]: analysis_area_id = 233
         wwi = calculate_wwi(analysis_area_id)
         print ('WWI for analysis area:{0} is {1} with "{2}" type'.format(
                 analysis_area_id,
                 wwi[0],
                 wwi[1]))
```

```
WWI for analysis area:233 is 0.87 with "Weekly Multipurpose" type
```

1.2.9 Compute AMI:

- AMI = Average Morning/Midday Index
- vh = Average weekday hourly count for hour (h) where hours are given as starting time of the hour

Grouping via AMI

- Hourly groups by Average AMI metric
- Hourly Noon Activity: Average AMI <= 0.7
- Hourly Multipurpose: 7< (Average AMI) < =1.4
- Hourly Commute: Average AMI > 1.4

```
In [22]: from utility import db_connect, query2csv
         from settings import  DBNAME, DBPASS, DBUSER, DBHOST

         def calculate_ami(analysis_area_id):
             query = """
         with hrly as (
          select
             baaa.analysis_area_id,
             date_trunc('day', bpd.start_time) as date,
             to_char(bpd.start_time, 'HH24') as hour,
             sum(bpd.volume) as volume
          from
             baa.analysis_areas as baaa
             inner join baa_ex_sus.data as bpd
               on bpd.flow_detector_id = Any(baaa.flow_detector_list::int[])
          where
              baaa.analysis_area_id = {0}
              group by analysis_area_id, bpd.start_time
         ),
         v_h_7_8 as (
         select
           baadv.analysis_area_id,
           avg(baadv.volume) volume
         from
```

```
  hrly as baadv
where
  baadv.hour in ('07', '08')
  and extract(dow from baadv.date) in (1,2,3,4,5)
  group by baadv.analysis_area_id
  ),
  v_h_11_12 as (
select
  baadv.analysis_area_id,
  avg(baadv.volume) volume
from
  hrly as baadv
where
  baadv.hour in ('11', '12')
  and extract(dow from baadv.date) in (1,2,3,4,5)
  group by baadv.analysis_area_id
  ),
  ami as (
  select
    vh78.analysis_area_id,
    round(vh78.volume, 2) as vh_78,
    round(vh1112.volume,2) as vh_11_12,
    round(vh78.volume/vh1112.volume, 2) as ami,
    case
     when (round(vh78.volume/vh1112.volume, 2) <= 0.7)
       then 'Hourly Noon Activity'
     when (round(vh78.volume/vh1112.volume, 2) >  1.4)
       then 'Hourly Commute'
     ELSE 'Hourly Multipurpose'
  END as hour_group
  from
    v_h_7_8 as vh78 inner join v_h_11_12 as vh1112
      using(analysis_area_id)
  )
  select
    ami.analysis_area_id,
    baaa.mode,
    baaa.analysis_area_name,
    ami.vh_78,
    ami.vh_11_12,
    ami.ami,
    ami.hour_group
  from
   ami inner join baa.analysis_areas as baaa using(analysis_area_id)
   order by 1
    """.format(analysis_area_id)
    conn = db_connect()
    with conn:
```

```
                    with conn.cursor() as curs:
                        curs.execute(query)
                        rows = curs.fetchall()
                        if rows[0]:
                            return (rows[0][5], rows[0][6])
```

1.2.10 Calculate AMI for the just created analysis area 233

```
In [34]: analysis_area_id = 233
         ami = calculate_ami(analysis_area_id)
         print ('AMI for analysis area:{0} is {1} with "{2}" type'.format(
                 analysis_area_id,
                 ami[0],
                 ami[1]))
```

```
AMI for analysis area:233 is 1.47 with "Hourly Commute" type
```