PRJ 1 LA Crime Code Documentation:

Table of Contents

Click below to navigate to section:

1. Creating Fact Table and Loading Data
2. Cleaning the Data
3. Creating Reference Tables and Applying Constraints and Foreign Keys
4. Adding custom columns to the Data
5. Creating a custom Descendent and Date Table
6. Creating Calculated fields and rolling totals.

Data Details:

Data Source:
https://catalog.data.gov/dataset/crime-data-from-2020-to-present

Data Dictionary:
https://data.lacity.org/Public-Safety/Crime-Data-from-2020-to-Present/2nrs-mtv8/about_data

Creating Fact Table and Loading Data

```
CREATE TABLE la_crimes (
division_records_num varchar(10) PRIMARY KEY,
date_rptd timestamptz,
date_occ timestamptz,
time_occ time,
geo_area_num varchar(5),
area_name varchar(20),
rpt_dist_no varchar(5),
part_1_2 integer,
crime_code varchar(5),
crime_code_description text,
modus_operandi_code varchar(50),
victim_age smallint,
victim_sex varchar(1),
victim_descent varchar(1),
premises_code integer,
premises_description text,
weapon_used_code varchar(3),
weapon_description varchar(50),
```

```
case_status text,
status_description text,
crime_code_1 varchar(5),
crime_code_2 varchar(5),
crime_code_3 varchar(5),
crime_code_4 varchar(5),
crime_location varchar(50),
cross_street varchar(50),
latitude double precision,
longtitude double precision
);
```

- Before loading the Data the table is created. The names of the original data set were changed when creating the table to make the data user friendly.
- The primary key was assigned as a division record number because this uniquely identifies each crime case.
- Data types were selected to match the original data, enable accurate and efficient queries and optimize storage.

```
-- Import la_crimes data into table.
COPY la_crimes
FROM 'C:\yourpathhere ' --<<--- Change to 'your path'
WITH (FORMAT CSV, HEADER);
```

- Using a COPYand WITH statement a CSV with headers version of the data was loaded into the postgres database.

Cleaning the Data

```
---Remove Duplicates
BEGIN TRANSACTION;

-- Create a backup of the original table, just in case
CREATE TABLE la_crimes_backup AS
SELECT * FROM la_crimes;

-- Remove all existing data from the original table
TRUNCATE TABLE la_crimes;
```

```
-- Insert distinct rows back into the original table
INSERT INTO la_crimes
SELECT DISTINCT * FROM la_crimes_backup;

-- Drop the backup table if everything is successful
DROP TABLE la_crimes_backup;

COMMIT;
```

- The data set is being cleared of duplicates using Transaction and Commit to allow for reversal to give room to reverse transactions.
- The data in **la_crimes** is backed up in a copy of itself, then all rows are removed from the original using Truncate.
- The table is then filled with the **la_crimes** data set to all as distinct. This removes duplicate rows.
- The data, after review, is normal without errors. This allows for this deletion of the backup table.

```
--- Clean Address fields of abnormal spaces (run until spaces are uniform and proper.)
UPDATE la_crimes
SET crime_location = TRIM(REPLACE(crime_location, '  ', ' ')),
cross_street = TRIM(REPLACE(cross_street, '  ', ' '));
SELECT
   crime_location
   ,cross_street
FROM la_crimes
WHERE cross_street IS NOT NULL
```

- The number spaces of streets and roads in the location and street data are excessive and not uniform. Using the above code repeatedly will eventually lead to only a single space between words in all field values.

Creating Reference Tables and Applying Constraints and Foreign Keys

```
-- Crimes
CREATE TABLE crimes AS
SELECT DISTINCT
   crime_code,
   crime_code_description
```

```
FROM la_crimes
WHERE crime_code_description IS NOT NULL;

ALTER TABLE crimes
ADD CONSTRAINT pk_crimes PRIMARY KEY (crime_code);
```

- The crimes table is a reference table that holds all possible crimes and crime codes that can be found in the data. Crime code is made to be the foreign key. All reference tables are constructed by selecting distinct combinations of fields, removing NULLs and assigning the primary key constraint to a code field.

```
-- Weapons
CREATE TABLE crime_weapons AS
SELECT DISTINCT
   weapon_used_code,
   weapon_description
FROM la_crimes
WHERE weapon_used_code IS NOT NULL AND weapon_description IS NOT NULL;

ALTER TABLE crime_weapons
ADD CONSTRAINT pk_crime_weapons PRIMARY KEY (weapon_used_code);
```

- Crime_weapons is a reference table created to store the type of weapons used in crime cases

```
-- Case Details
CREATE TABLE case_details AS
SELECT DISTINCT
   case_status,
   status_description
FROM la_crimes
WHERE case_status IS NOT NULL AND status_description IS NOT NULL;

ALTER TABLE case_details
ADD CONSTRAINT pk_case_details PRIMARY KEY (case_status);
```

- Case_details gives the type of case event to occur, including details on the perpetrator.

```
-- Premises
CREATE TABLE premises AS
SELECT DISTINCT
```

```
    premises_code,
    premises_description
FROM la_crimes
WHERE premises_code IS NOT NULL AND premises_description IS NOT NULL;

ALTER TABLE premises
ADD CONSTRAINT pk_premises PRIMARY KEY (premises_code);
```

- Premises is a reference table that is meant to describe the crime locations type. Ex. Hospital.

```
-- Area
CREATE TABLE crime_geos AS
SELECT DISTINCT
    geo_area_num,
    area_name
FROM la_crimes
WHERE geo_area_num IS NOT NULL AND area_name IS NOT NULL;

ALTER TABLE crime_geos
ADD CONSTRAINT pk_crime_geos PRIMARY KEY (geo_area_num);
```

- Crime geos is a reference table meant to give details on the area of LA the crime was committed.

```
-- Victim Descent
CREATE TABLE crime_victim_descents AS
SELECT DISTINCT
    victim_descent
FROM la_crimes
WHERE victim_descent IS NOT NULL;

ALTER TABLE crime_victim_descents
ADD CONSTRAINT pk_crime_victim_descents PRIMARY KEY (victim_descent);
```

- Crime_vivtim_descent is a reference table capturing the victims descent. Ex. Cambodian

```
-- All Crime Codes
CREATE TABLE crime_codes AS
SELECT DISTINCT
    division_records_num,
    date_rptd::date AS date_rptd,
```

```
   crime_code_1,
   crime_code_2,
   crime_code_3,
   crime_code_4
FROM la_crimes;

ALTER TABLE crime_codes
ADD CONSTRAINT pk_crime_codes PRIMARY KEY (division_records_num);
```

- Crime_codes is a reference table used to track the number of crimes committed and the number of crimes committed in a single case. Later on in the documentations it is used to create calculated fields to track case crime density and to create rolling totals of specific types of crimes on different date intervals.

```
--- Assigning Foreign Keys
ALTER TABLE la_crimes
  ADD CONSTRAINT fk_crime_code FOREIGN KEY (crime_code) REFERENCES
crimes(crime_code),
  ADD CONSTRAINT fk_weapon_used_code FOREIGN KEY (weapon_used_code) REFERENCES
crime_weapons(weapon_used_code),
  ADD CONSTRAINT fk_case_status FOREIGN KEY (case_status) REFERENCES
case_details(case_status),
  ADD CONSTRAINT fk_geo_area_num FOREIGN KEY (geo_area_num) REFERENCES
crime_geos(geo_area_num),
  ADD CONSTRAINT fk_victim_descent FOREIGN KEY (victim_descent) REFERENCES
crime_victim_descents(victim_descent);
```

- Foreign Keys are created between all tables to allow for proper table relationships.

Adding custom columns to the Data

```

--- Crime case count
ALTER TABLE crime_codes
ADD COLUMN crime_case_density INTEGER,
ADD COLUMN crime_tally INTEGER DEFAULT 1;

UPDATE crime_codes
```

```
SET crime_case_density =
 CASE
  WHEN crime_code_1 IS NOT NULL AND crime_code_2 IS NULL AND crime_code_3 IS NULL
AND crime_code_4 IS NULL THEN 1
  WHEN crime_code_1 IS NOT NULL AND crime_code_2 IS NOT NULL AND crime_code_3 IS
NULL AND crime_code_4 IS NULL THEN 2
  WHEN crime_code_1 IS NOT NULL AND crime_code_2 IS NOT NULL AND crime_code_3 IS
NOT NULL AND crime_code_4 IS NULL THEN 3
  WHEN crime_code_1 IS NOT NULL AND crime_code_2 IS NOT NULL AND crime_code_3 IS
NOT NULL AND crime_code_4 IS NOT NULL THEN 4
  ELSE NULL
 END;
```

- Crime_tally is a way of turning each unique crime into an integer of one to allow for calculations of crime totals and crime density.
- Crime_case_density used the crime code fields to identify how many crimes were committed in a given case. Using a case statement to check each crime code field in sequential order. The number in the crime code field indicates the number of crimes committed in a case and the level of severity of the crime.

```
ALTER TABLE crime_weapons
ADD COLUMN weapon_type  varchar(10);
UPDATE crime_weapons
SET weapon_type =
CASE
  WHEN weapon_used_code = '307'   THEN 'MISC'
  WHEN weapon_used_code = '514'   THEN 'Bludgeon'
  WHEN weapon_used_code IN('513','502') THEN 'GUN'
  WHEN weapon_used_code LIKE '1%' THEN 'GUN'
  WHEN weapon_used_code LIKE '2%' THEN 'Sharp Obj.'
  WHEN weapon_used_code LIKE '3%' THEN 'Bludgeon'
  WHEN weapon_used_code LIKE '4%' THEN 'Body'
  WHEN weapon_used_code LIKE '5%' THEN 'MISC'
  ELSE weapon_type
END;
```

- Weapon_type creates a higher level category for weapons to allow better data visualizations and reporting.

Creating a custom Descendent and Date Table

```
---- FOR DESCENDENT FIELD TO CREATE DESCENDENT REFERENCE TABLE---

/*
Conversion found in Data Dictionary link
Code:
 B - Black
C - Chinese
D - Cambodian
F - Filipino
G - Guamanian
H - Hispanic /Latin /Mexican
I - American Indian/Alaskan Native
J - Japanese
K - Korean
L - Laotian
O - Other
P - Pacific Islander
S - Samoan
U - Hawaiian
 V - Vietnamese
W - White
X - Unknown
Z - Asian Indian
 A - Other Asian
*/



ALTER TABLE crime_victim_descents
ADD COLUMN victim_descent_full VARCHAR(50);

UPDATE crime_victim_descents
SET victim_descent_full =
CASE
    WHEN victim_descent = 'B' THEN 'Black'
    WHEN victim_descent = 'C' THEN 'Chinese'
    WHEN victim_descent = 'D' THEN 'Cambodian'
```

```
    WHEN victim_descent = 'F' THEN 'Filipino'
    WHEN victim_descent = 'G' THEN 'Guamanian'
    WHEN victim_descent = 'H' THEN 'Hispanic/Latin/Mexican'
    WHEN victim_descent = 'I' THEN 'American Indian/Alaskan Native'
    WHEN victim_descent = 'J' THEN 'Japanese'
    WHEN victim_descent = 'K' THEN 'Korean'
    WHEN victim_descent = 'L' THEN 'Laotian'
    WHEN victim_descent = 'A' THEN 'Other Asian'
    WHEN victim_descent = 'O' THEN 'Other'
    WHEN victim_descent = 'P' THEN 'Pacific Islander'
    WHEN victim_descent = 'S' THEN 'Samoan'
    WHEN victim_descent = 'U' THEN 'Hawaiian'
    WHEN victim_descent = 'V' THEN 'Vietnamese'
    WHEN victim_descent = 'W' THEN 'White'
    WHEN victim_descent = 'X' THEN 'Unknown'
    WHEN victim_descent = 'Z' THEN 'Asian Indian'
END;
```

- Crime_victum_descent_full is a field that includes the full name for the victims descent to help clarify the background of the victim. The conversion is found in the data dictionary link.

```
-- Accommodate data set date range by creating the date table from 2019 to the present.
CREATE TABLE date_table AS
SELECT generate_series('2019-01-01'::date, CURRENT_DATE, '1 day')::date AS date;

ALTER TABLE date_table
ADD COLUMN month varchar(10),
ADD COLUMN month_year varchar(25),
ADD COLUMN year varchar(4),
ADD COLUMN day varchar(10),
ADD COLUMN holiday_season varchar(15),
ADD COLUMN season varchar(6);

UPDATE date_table
SET month = TRIM(TO_CHAR(date, 'Month')),
    year = TRIM(TO_CHAR(date, 'YYYY')),
    month_year = TRIM(TO_CHAR(date, 'Month YYYY')),
    day = TRIM(TO_CHAR(date, 'Day')),
```

```
  holiday_season = CASE
    WHEN month = 'December' THEN 'Christmas'
    WHEN month = 'November' THEN 'Thanksgiving'
    WHEN month = 'October' THEN 'Halloween'
    WHEN month = 'July' THEN '4th of July'
    WHEN month = 'February' THEN 'Valentines'
    WHEN month = 'March' THEN 'St. Patricks'
    ELSE NULL
  END,
  season = CASE
    WHEN month IN ('December', 'January', 'February') THEN 'Winter'
    WHEN month IN ('March', 'April', 'May') THEN 'Spring'
    WHEN month IN ('June', 'July', 'August') THEN 'Summer'
    WHEN month IN ('September', 'October', 'November') THEN 'Fall'
    ELSE NULL
  END;
```

- A date table is created from 2019 to present to allow for complete confidence that all dates in the data will be present in the data set. The data is updated periodically so it is important to have this included in the date field.
- Other date intervals and seasons both holiday and weather are included using case statements.

Creating Calculated fields and rolling totals.

```
-- Crime Counts Rolling

ALTER TABLE crime_codes
ADD COLUMN rptd_month_rolling_crime_cnt integer,
ADD COLUMN rptd_month_yr_rolling_crime_cnt integer,
ADD COLUMN rptd_yr_rolling_crime_cnt integer,
ADD COLUMN rptd_dy_rolling_crime_cnt integer,
ADD COLUMN rptd_holiday_szn_rolling_crime_cnt integer,
ADD COLUMN rptd_szn_rolling_crime_cnt integer;

WITH rolling_crime_counts AS (
  SELECT
    c.division_records_num,
     SUM(c.crime_tally) OVER(PARTITION BY month ORDER BY d.date ASC) AS
rptd_month_rolling_crime_cnt,
```

```
        SUM(c.crime_tally) OVER(PARTITION BY month_year ORDER BY d.date ASC) AS
rptd_month_yr_rolling_crime_cnt,
        SUM(c.crime_tally) OVER(PARTITION BY year ORDER BY d.date ASC) AS
rptd_yr_rolling_crime_cnt,
        SUM(c.crime_tally) OVER(PARTITION BY day ORDER BY d.date ASC) AS
rptd_dy_rolling_crime_cnt,
        SUM(c.crime_tally) OVER(PARTITION BY season ORDER BY d.date ASC) AS
rptd_holiday_szn_rolling_crime_cnt,
        SUM(c.crime_tally) OVER(PARTITION BY holiday_season ORDER BY d.date ASC) AS
rptd_szn_rolling_crime_cnt
    FROM crime_codes c
    LEFT JOIN date_table d
    ON c.date_rptd = d.date
)
UPDATE crime_codes c
SET
    rptd_month_rolling_crime_cnt = r.rptd_month_rolling_crime_cnt,
    rptd_month_yr_rolling_crime_cnt = r.rptd_month_yr_rolling_crime_cnt,
    rptd_yr_rolling_crime_cnt = r.rptd_yr_rolling_crime_cnt,
    rptd_dy_rolling_crime_cnt = r.rptd_dy_rolling_crime_cnt,
    rptd_holiday_szn_rolling_crime_cnt = r.rptd_holiday_szn_rolling_crime_cnt,
    rptd_szn_rolling_crime_cnt = r.rptd_szn_rolling_crime_cnt
FROM rolling_crime_counts r
WHERE c.division_records_num = r.division_records_num;
```

- Several fields are created to get rolling counts of crimes on several date interval levels. This allows for further analysis of crimes at specific time intervals.
- A CTE named rolling_crime_counts is created that includes window functions to generate the rolling totals and and the date_table is joined on the date to allow for the inclusion of these intervals.
- The CTE allows for the updating of the newly created fields in the crime code table with the data generated from these calculated fields via an update statement.

```
-- Crime Density

ALTER TABLE crime_codes
ADD COLUMN rptd_month_rolling_cd integer,
ADD COLUMN rptd_month_yr_rolling_cd integer,
ADD COLUMN rptd_yr_rolling_cd integer,
ADD COLUMN rptd_dy_rolling_cd integer,
ADD COLUMN rptd_holiday_szn_rolling_cd integer,
ADD COLUMN rptd_szn_rolling_cd integer;
```

```
WITH crime_codez AS (
   SELECT
      c.division_records_num,
       SUM(c.crime_case_density) OVER(PARTITION BY month ORDER BY d.date ASC) AS
rptd_month_rolling_cd,
       SUM(c.crime_case_density) OVER(PARTITION BY month_year ORDER BY d.date ASC) AS
rptd_month_yr_rolling_cd,
       SUM(c.crime_case_density) OVER(PARTITION BY year ORDER BY d.date ASC) AS
rptd_yr_rolling_cd,
       SUM(c.crime_case_density) OVER(PARTITION BY day ORDER BY d.date ASC) AS
rptd_dy_rolling_cd,
       SUM(c.crime_case_density) OVER(PARTITION BY season ORDER BY d.date ASC) AS
rptd_holiday_szn_rolling_cd,
       SUM(c.crime_case_density) OVER(PARTITION BY holiday_season ORDER BY d.date ASC)
AS rptd_szn_rolling_cd
     FROM crime_codes c
   LEFT JOIN date_table d
   ON c.date_rptd = d.date
)
UPDATE crime_codes
SET
   rptd_month_rolling_cd = r.rptd_month_rolling_cd,
   rptd_month_yr_rolling_cd = r.rptd_month_yr_rolling_cd,
   rptd_yr_rolling_cd = r.rptd_yr_rolling_cd,
   rptd_dy_rolling_cd = r.rptd_dy_rolling_cd,
   rptd_holiday_szn_rolling_cd = r.rptd_holiday_szn_rolling_cd,
   rptd_szn_rolling_cd = r.rptd_szn_rolling_cd
FROM crime_codez r
WHERE crime_codes.division_records_num = r.division_records_num;
```

- Above is a repeat of the other group of calculated fields except it is done using crime density.