

- Building RDF with LA
Public Safety Data

By Lauro Cabral, Christopher Vargas, and Sotheanith Sok



Datasets' Criteria

Why we chose these datasets

- Criteria 1: Related Datasets

- The relationship between arrest dataset and crime dataset is complex enough to derive many conclusions and facts about crime in LA

- Criteria 2: Structured Datasets

- Datasets have inherent structures such as person, location, and weapon and we can theorize hierarchical structures from such datasets

- Criteria 3: Large Datasets

- Datasets are large enough such that a complex RDF file can be deduced.



Dataset 1: Arrest Data from 2020 to Present

URL: <https://data.lacity.org/resource/amvf-fr72>

● Arrest Data from 2020 to Present: Labels

- Report ID
- Report Type
- Arrest Date
- Time
- Area ID
- Area Name
- Reporting District
- Age
- Sex Code
- Descent Code
- Charge Group Code
- Charge Group Description
- Arrest Type Code
- Charge
- Charge Description
- Disposition Description
- Address
- Cross Street
- LAT
- LON
- Location
- Booking Date
- Booking Time
- Booking Location
- Booking Location Code



Dataset 2: Crime Data from 2020 to Present

URL: <https://data.lacity.org/resource/2hrs-mtv8>

● Crime Data from 2020 to Present: Labels

- DR_NO
- Date Rptd
- DATE OCC
- TIME OCC
- AREA
- AREA NAME
- Rpt Dist No
- Part 1-2
- Crm Cd
- Crm Cd Desc
- Mocodes
- Vict Age
- Vict Sex
- Vict Descent
- Premis Cd
- Premis Desc
- Weapon Used Cd
- Weapon Desc
- Status
- Status Desc
- Crm Cd 1
- Crm Cd 2
- Crm Cd 3
- Crm Cd 4
- LOCATION
- Cross Street
- LAT
- LON



Queries

- Query 1

- Does age affect the likelihood that a person will be involved in a crime?

- Query 2

- What is the safest time to travel in LA?

- Query 3

- Based on your gender, how likely are you to be involved with a crime in a given neighborhood?



RDF Schema

Classes

There are 11 classes:

- Report
- Person
- Location
- ArrestReport
- Charge
- Booking
- CrimeReport
- Crime
- Premise
- Weapon
- Status

- Report Class

- - It is an instance of RDFS: Class
 - Properties:
 - “hasID” - XSD: integer
 - “hasPerson” - Person class
 - “hasID” - XSD: integer
 - “hasTime” - XSD: integer
 - “hasDate” - XSD: integer
 - “hasLocation” - Location class

- Person Class

- - It is an instance of RDFS: Class
 - Properties:
 - “hasAge” - XSD: integer
 - “hasSex” - XSD: string
 - “hasDescendent” - XSD: string

● Location Class

- It is an instance of RDFS: Class
- Properties:
 - “hasReportingDistrictNumber” - XSD: integer
 - “hasAreaID” - XSD: integer
 - “hasAreaName” - XSD: string
 - “hasAddress” - XSD: string
 - “hasCrossStreet” - XSD: string
 - “hasLatitude” - XSD: double
 - “hasLongitude” - XSD: double

- ArrestReport Class

- - It is a subclass of Report class
 - Properties:
 - “hasDispositionDescription” - XSD: string
 - “hasReportType” - XSD: string
 - “hasArrestType” - XSD: string
 - “hasCharge” - Charge class
 - “hasBooking” - Booking class

- Charge Class

- - It is an instance of RDFS: Class
 - Properties:
 - “hasChargeGroupCode” - XSD: integer
 - “hasChargeGroupDescription” - XSD: string
 - “hasChargeCode” - XSD: integer
 - “hasChargeDescription” - XSD: string

- Booking Class

- - It is an instance of RDFS: Class
 - Properties:
 - “hasBookingDate” - XSD: date
 - “hasBookingTime” - XSD: time
 - “hasBookingLocation” - XSD: string
 - “hasBookingCode” - XSD: integer

- CrimeReport Class

- - It is a subclass of Report class
 - Properties:
 - “hasDateReported” - XSD: date
 - “hasMocodes” - XSD: string
 - “hasCrime” - Crime class
 - “hasStatus” - Status class
 - “hasWeapon” - Weapon class
 - “hasPremise” - Premise class
 - “hasPart1-2 ” - XSD: integer

● Crime Class

- It is an instance of RDFS: Class
- Properties:
 - “hasCrimeCommitted” - XSD: string
 - “hasCrimeCommittedDescription” - XSD: string
 - “hasCrimeCommitted1” - XSD: string
 - “hasCrimeCommitted2” - XSD: string
 - “hasCrimeCommitted3” - XSD: string
 - “hasCrimeCommitted4” - XSD: string

- Status Class

- - It is an instance of RDFS: Class
 - Properties:
 - “hasStatusCode” - XSD: integer
 - “hasStatusDescription” - XSD: string

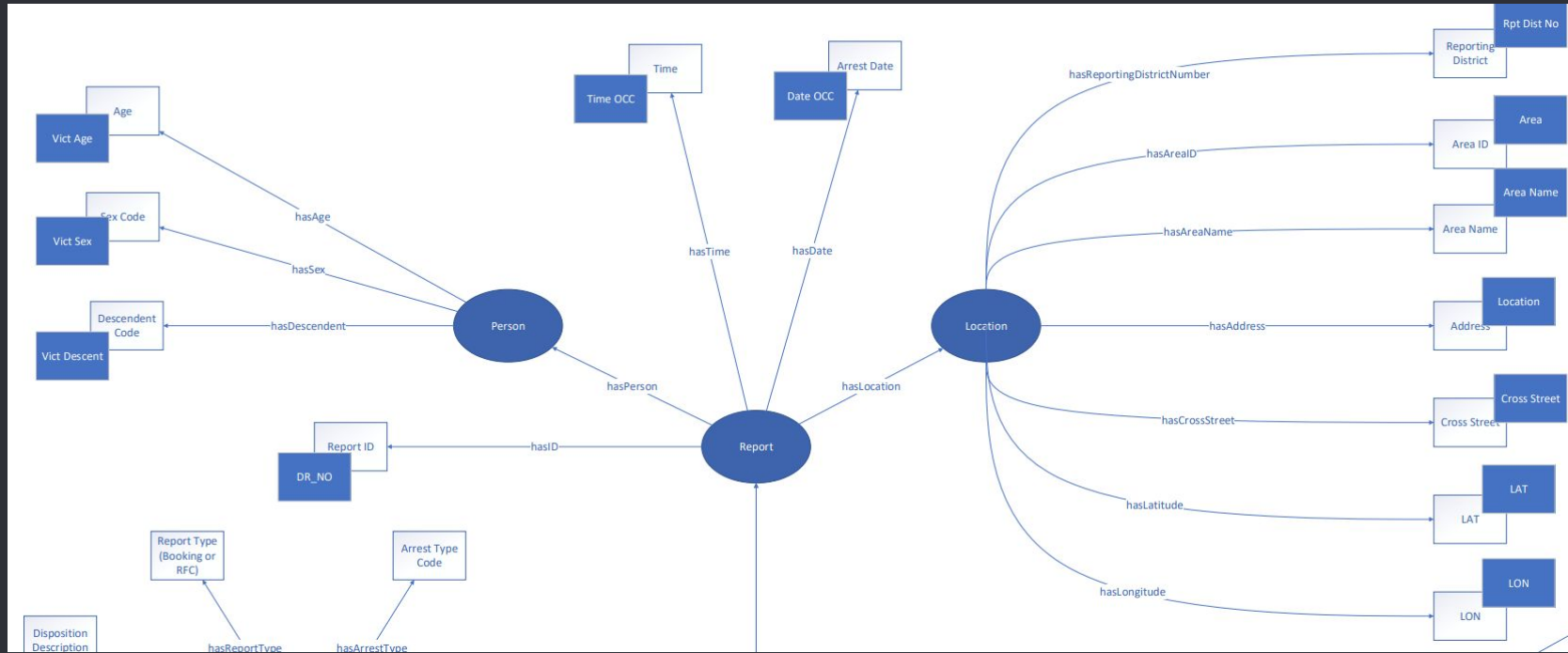
- Weapon Class

- - It is an instance of RDFS: Class
 - Properties:
 - “hasWeaponUsedCode” - XSD: integer
 - “hasWeaponDescription” - XSD: string

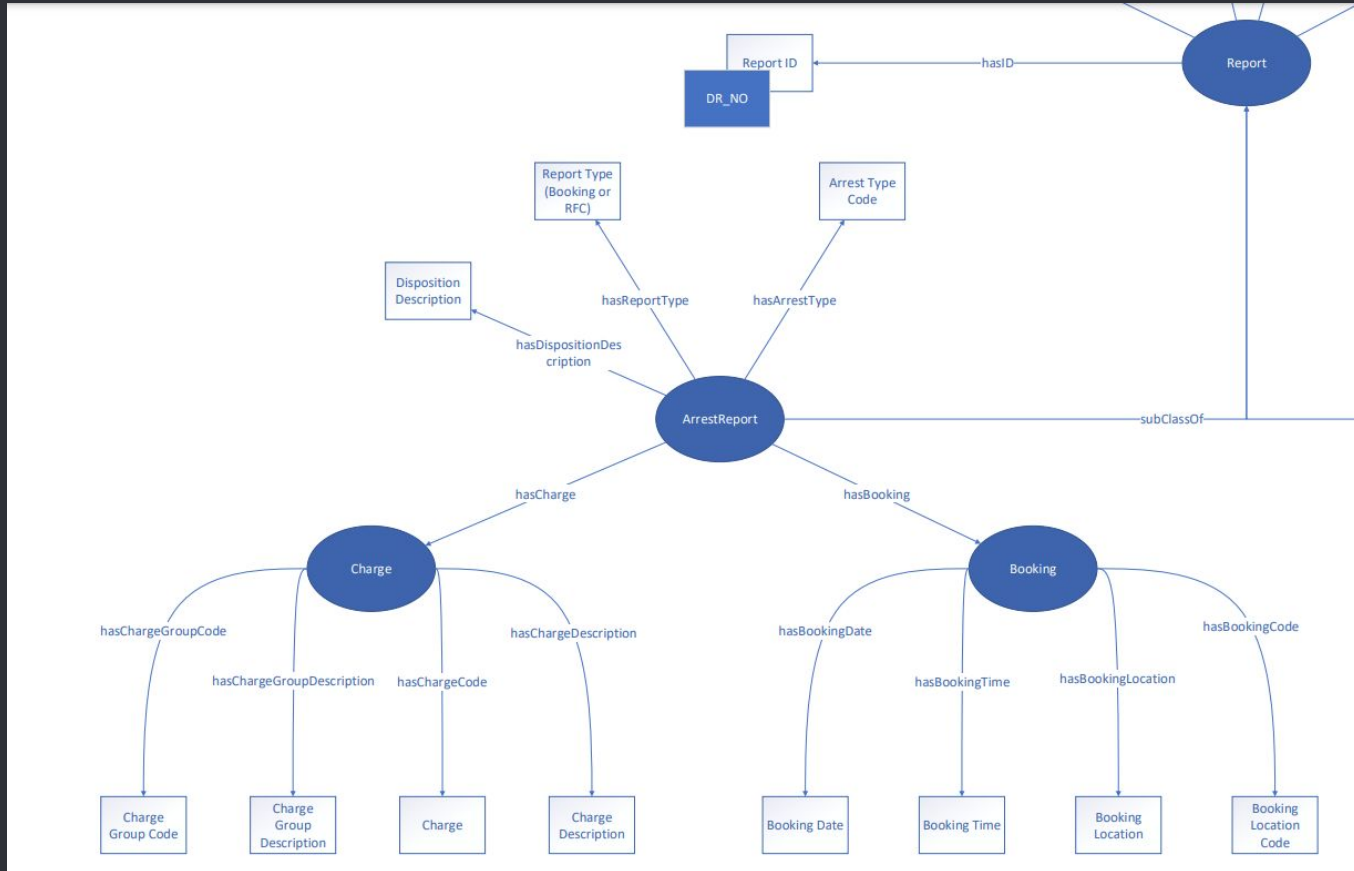
- Premise Class

- - It is an instance of RDFS: Class
 - Properties:
 - “hasPremiseCode” - XSD: integer
 - “hasPremiseDescription” - XSD: string

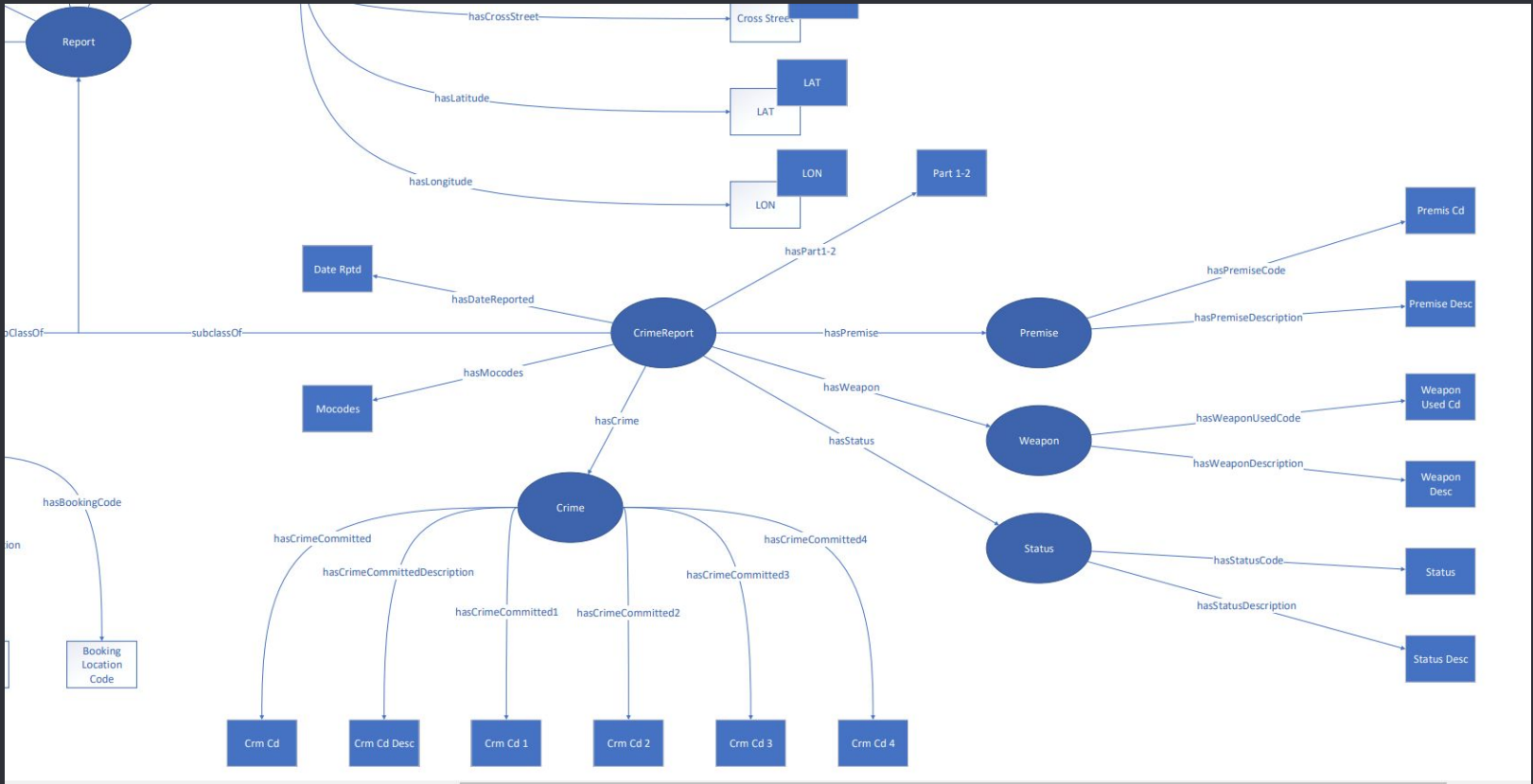
What Does It Look Like - Part 1



What Does It Look Like - Part 2



What Does It Look Like - Part 3





Source Code

- Requirements

- - Python
 - Request
 - RdfLib
 - Pandas
 - Auto-Py-to-Exe

Code Snippet 2.0: Adding Arrest Reports data to the RDF graph

```
def _add_arrest_reports_to_graph(self, arrest_reports, graph, namespace):
    """Add arrest reports dataset to the RDF graph

    Args:
        arrest_reports_dataset (string): a CSV contains arrest reports with well-formatted data
        graph (Graph): an RDF graph
        namespace (string): base namespace for all resources

    Returns:
        [Graph]: an RDF graph contains data from the arrest report dataset
    """

    #Print out the process
    print("INFO: Add arrest reports dataset to graph...")

    #Looping through everyone row of arrest reports
    for i in range(1, len(arrest_reports)):

        #Determine how many instances of Report already exist in the graph
        number_report = len(list(graph.subject_objects(predicate=namespace["hasID"])))

        #Add a new instances of ArrestReport class and fill its properties that it inherent from Report class
        graph.add((namespace["Report#" + str(number_report)], RDF.type, namespace["ArrestReport"]))
        graph.add((namespace["Report#" + str(number_report)], namespace["hasID"], Literal(arrest_reports[i][0], datatype=XSD.integer)))
        graph.add((namespace["Report#" + str(number_report)], namespace["hasDate"], Literal(arrest_reports[i][2], datatype=XSD.date)))
        graph.add((namespace["Report#" + str(number_report)], namespace["hasTime"], Literal(arrest_reports[i][3], datatype=XSD.time)))
        graph.add((namespace["Report#" + str(number_report)], namespace["hasReporType"], Literal(arrest_reports[i][1], datatype=XSD.string)))
        graph.add((namespace["Report#" + str(number_report)], namespace["hasArrestType"], Literal(arrest_reports[i][12], datatype=XSD.string)))
        graph.add((namespace["Report#" + str(number_report)], namespace["hasDispositionDescription"], Literal(arrest_reports[i][15], datatype=XSD.string)))
```

Code Snippet 2.1: Adding Arrest Reports data to the RDF graph

```
#Add an instance of Person class
#Check if an instance of Person class already exist in the graph
people_age = set(graph.subjects(predicate = namespace["hasAge"], object=Literal(arrest_reports[i][7], datatype=XSD.integer)))
people_sex = set(graph.subjects(predicate = namespace["hasSex"], object=Literal(arrest_reports[i][8], datatype=XSD.string)))
people_descendent = set(graph.subjects(predicate = namespace["hasDescendent"], object=Literal(arrest_reports[i][9], datatype=XSD.string)))
person = list(people_age & people_sex & people_descendent)

# Return the number of instances of Person class
number_person = len(list(graph.subject_objects(predicate=namespace["hasAge"])))

#Create a new instance of Person class
if(len(person) == 0):
    # add to Person
    graph.add((namespace["Person#" + str(number_person)], RDF.type, namespace["Person"]))
    graph.add((namespace["Person#" + str(number_person)], namespace["hasAge"], Literal(arrest_reports[i][7], datatype=XSD.integer)))
    graph.add((namespace["Person#" + str(number_person)], namespace["hasSex"], Literal(arrest_reports[i][8], datatype=XSD.string)))
    graph.add((namespace["Person#" + str(number_person)], namespace["hasDescendent"], Literal(arrest_reports[i][9], datatype=XSD.string)))
    person = namespace["Person#" + str(number_person)]

#Reuse an existing instance of Person class
else:
    person = person[0]

#Add to report
graph.add((namespace["Report#" + str(number_report)], namespace["hasPerson"], person))
graph.add((namespace["Report#" + str(number_report)], namespace["hasLocation"], location))
graph.add((namespace["Report#" + str(number_report)], namespace["hasBooking"], booking))
graph.add((namespace["Report#" + str(number_report)], namespace["hasCharge"], charge))
```




Technical Challenges

- Challenge 1: Synonym Labels of Data

- Problem:

Even though both datasets come from a single source

(<https://data.lacity.org/>), there are multiple labels used to define similar data.

Ex: Report ID (Arrest Reports)
== DR_NO (Crime Reports)

Solution:

We examines all data and unify similar data under a single label.

Ex: Report ID (Arrest Reports)
== DR_NO (Crime Reports) ==
ID (RDF Graph)

- Challenge 2: Inconsistent Data Format

- Problem:

In both datasets, some data do not match the format of XML Schema.

Ex:

XSD: time \Rightarrow hh:mm:ss

Data from dataset \Rightarrow "0935"

Solution:

We format all data to match XML Schema format

Ex: "0935" \Rightarrow 09:35:00

- Challenge 3: Missing Data

- Problem:

Since datasets are generated by real life events, there are a lot of missing data values.

Solution:

For completeness' sake, we will represent those missing values with black nodes