Metadata-Driven Data Science for Data Quality Monitoring in R

ODOT Annual Transportation Safety Conference

Tuesday, Sept 17, 2024

Oregon Health Authority

OREGON EMERGENCY MEDICAL SERVICES PROGRAM

CENTER FOR HEALTH PROTECTION

Acknowledgements

- The Oregon Emergency Medical Services Program personnel and Data Team members past and present
- ODOT and the TRCC for supporting this work
- National EMS Quality Alliance (NEMSQA)

Data Quality Monitoring Plan

The data quality monitoring plan is our roadmap for developing and implementing a set of tools to improve data quality:

- Complete dataset documentation/metadata
- Robust monitoring tools
- Replicable processes for update
- Scale up integrated development environment pilot project to enterprise
- Deployment of dashboards at the local level
- Development of a data quality toolkit
- Support for state and local data quality improvement initiatives

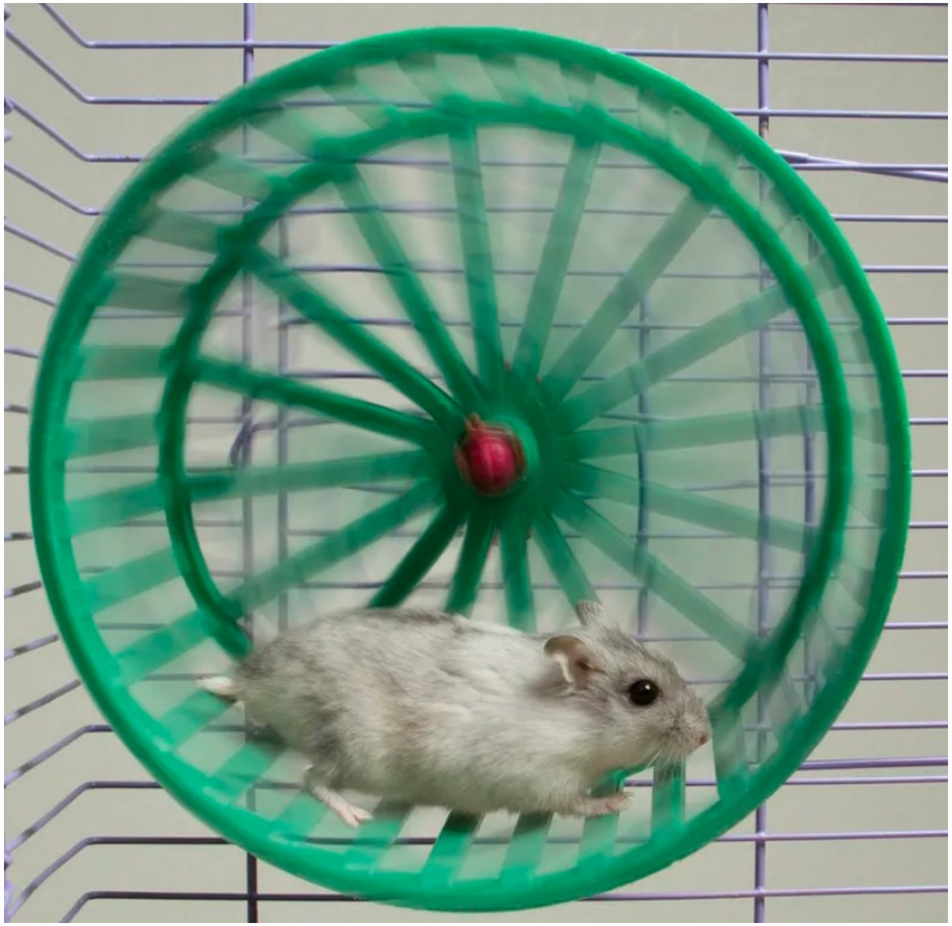
The Public Sector Quandary

Recent trends in public health, data science, EMS, and Trauma:

- Increasing opportunities for quality assurance and process improvement initiatives working with our committees, local, state, and national partners
- Public health modernization has brought in new expectations for public health programs to be data driven
- Greater visibility of data science as a discipline has brought new tools, increased data literacy, and greater demand for data
- Our data systems have expanded in coverage, making the data housed there more useful for statewide performance improvement initiatives, epidemiological work, and research

Potemkin Village Data Science

Potemkin Village Data Science



The Public Sector Quandary

- The only way to manage this situation is to get upstream from the project
- The only way to get upstream is to work with our metadata

Metadata-Driven Data Science

- The goal is to use the metadata to extract, process and visualize data to shine a light on data quality problems and share them with the end user
  - Process for updating metadata repository (data dictionary)
  - Scripts for pulling in metadata, and using metadata to build queries to extract data from our databases
  - Use metadata to standardize naming of columns in analytic dataset based on field names from the national data standard to support a national community of practice
  - Scripts to pull in updates to the analytic dataset, process, combine and deduplicate
  - All of this supports eventual automation of data product update

Metadata-Driven Data Science

Metadata-Driven Data Science

Why?

- Our data sets are not static:
  - What data elements are collected may be determined by national data standards which can change
  - Data Systems are hosted by vendors which may change
  - Our state dataset changes when the need arises
- We design our workflows so that we can update the data dictionary, and these changes flow downstream to all projects
- Functions in this pipeline can be updated in one place and one place only when updates/improvements are needed
- This is a preparatory step to full automation of our reporting and dashboard work

Data Dictionary

A data dictionary is many things:

- Resource for data system users
- A key for data system admins
- An agreement on how data are collected and used
- A tool for automating basic functions related to pulling and analyzing data
  - Machine readable
  - A single "source of truth"

Data Dictionary: Data Elements

	A	AL	AM	AN	AQ
1	Element_Name	Schema	Table.Name	Column.Name	query_order
2	Dim_Scene_PK	DwEms	Dim_Scene	Dim_Scene_PK	1
3	eScene.18	DwEms	Dim_Scene	Scene_Incident_State_Name_Default	3
4	Dim_Patient_PK	DwEms	Dim_Patient	Dim_Patient_PK	1
5	ePatient.15	DwEms	Dim_Patient	Patient_Age	3
6	ePatient.16	DwEms	Dim_Patient	Patient_Age_Units	5
7	eExam.01	DwEms	Dim_Patient	Patient_Weight_In_Kilograms_With_Not_Values	7
8	eExam.02	DwEms	Dim_Patient	Patient_Length_Based_Color	11
9	Dim_Situation_PK	DwEms	Dim_Situation	Dim_Situation_PK	1
10	eSituation.11	DwEms	Dim_Situation	Situation_Provider_Primary_Impression	3
11	eSituation.12	DwEms	Dim_Situation	Situation_Provider_Secondary_Impression_List	5
12	Dim_Incident_PK	DwEms	Dim_Incident	Dim_Incident_PK	1
13	Incident_Date_Time	DwEms	Dim_Incident	Incident_Date_Time	3
14	eTimes.02	DwEms	Dim_Incident	Incident_Dispatch_Notified_Date_Time	5
15	eTimes.03	DwEms	Dim_Incident	Incident_Unit_Notified_By_Dispatch_Date_Time	7
16	eRecord.01	DwEms	Dim_Incident	Incident_Patient_Care_Report_Number	11
17	NEMSIS_UUID	DwEms	Dim_Incident	NemsisUUID	13
18	Dim_Response_PK	DwEms	Dim_Response	Dim_Response_PK	1
19	eResponse.01	DwEms	Dim_Response	Response_EMS_Agency_Number	3
20	eResponse.05	DwEms	Dim_Response	Response_Type_Of_Service_Requested_With_Code	5
21	eResponse.24	DwEms	Dim_Response	Response_Additional_Response_Mode_Descriptors_List	7
22	eResponse.23	DwEms	Dim_Response	Response_Mode_To_Scene	11

Data Dictionary: Crosswalk

	A	B	C	D	E
1	db_Table1_Name	Table1_Foreign_Key	Correspondence	db_Table2_Name	Table2_Foreign_Key
2	Fact_Incident	Dim_Scene_FK	`1:1	Dim_Scene	Dim_Scene_PK
3	Fact_Incident	Dim_Patient_FK	`1:1	Dim_Patient	Dim_Patient_PK
4	Fact_Incident	Dim_Situation_FK	`1:1	Dim_Situation	Dim_Situation_PK
5	Fact_Incident	Dim_Incident_FK	`1:1	Dim_Incident	Dim_Incident_PK
6	Fact_Incident	Dim_Response_FK	`1:1	Dim_Response	Dim_Response_PK
7	Fact_Incident	Dim_Disposition_FK	`1:1	Dim_Disposition	Dim_Disposition_PK
8	Dim_Vitals	Fact_Incident_FK	`1:M	Fact_Incident	Fact_Incident_PK
9	Dim_Medications	Fact_Incident_FK	`1:M	Fact_Incident	Fact_Incident_PK
10	Dim_Procedures	Fact_Incident_FK	`1:M	Fact_Incident	Fact_Incident_PK
11	Fact_Incident	Agency_ID_Internal	`1:1	Dim_Agency	Agency_ID_Internal
12	Fact_Incident	Dim_Incident_Date_FK	`1:1	DSV_Dim_Incident_Date	Dim_Incident_Date_FK

Creating Your Data Dictionary

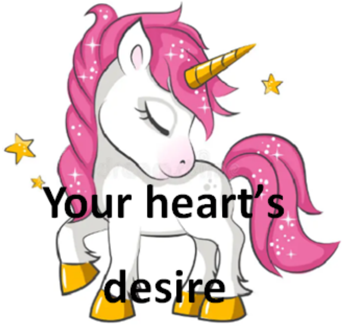
Initial set up:

- Bring together the data standard with database documentation
- Decide what to include:
  - Data elements
  - Variants of data elements (codes, descriptions, etc.)
  - Create custom columns that meet the needs of your use case (Sort Order, List?, etc.)
- Develop an automated method of updating your data dictionary

IMAGETREND®

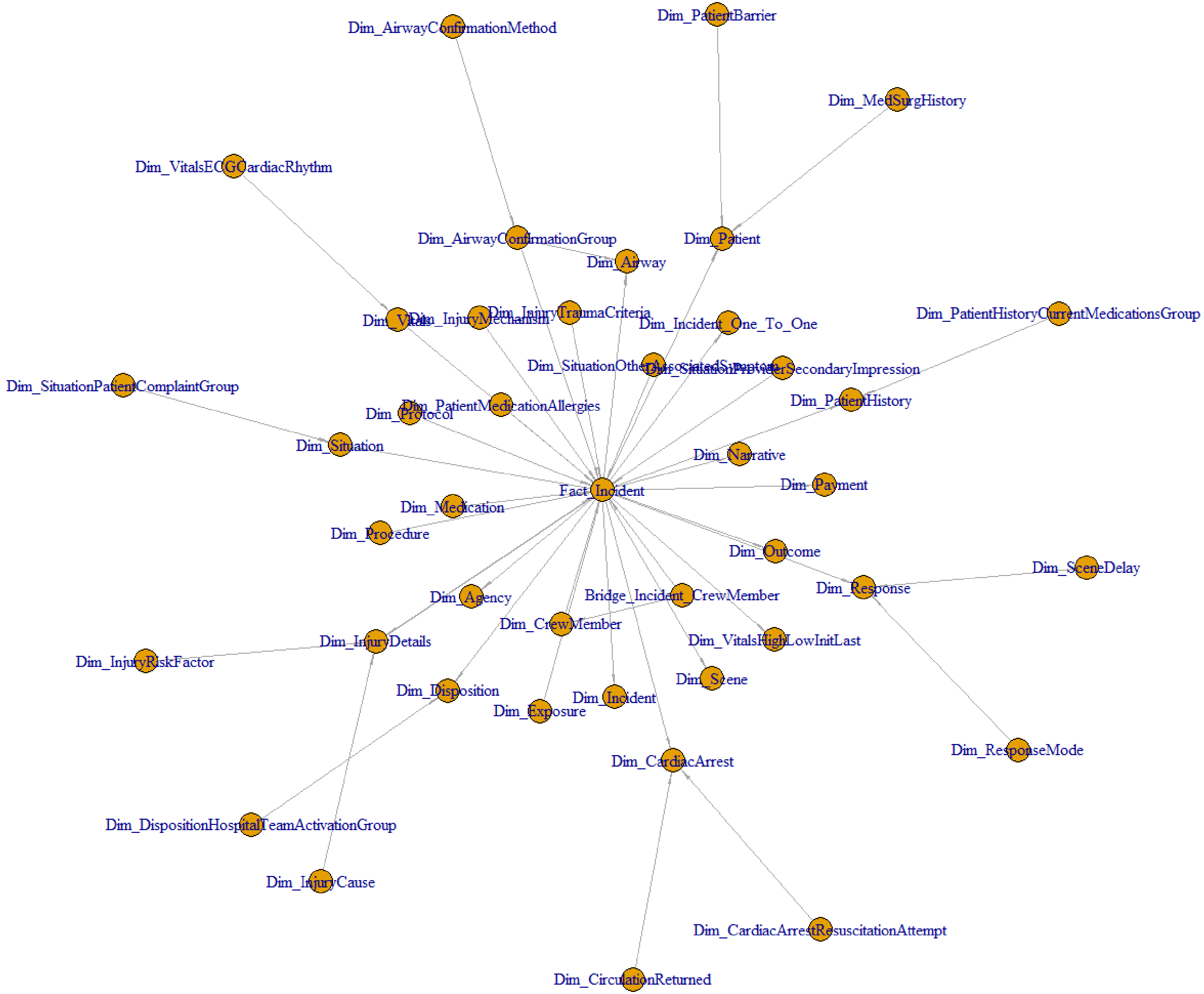
DataMart

Data Dictionary

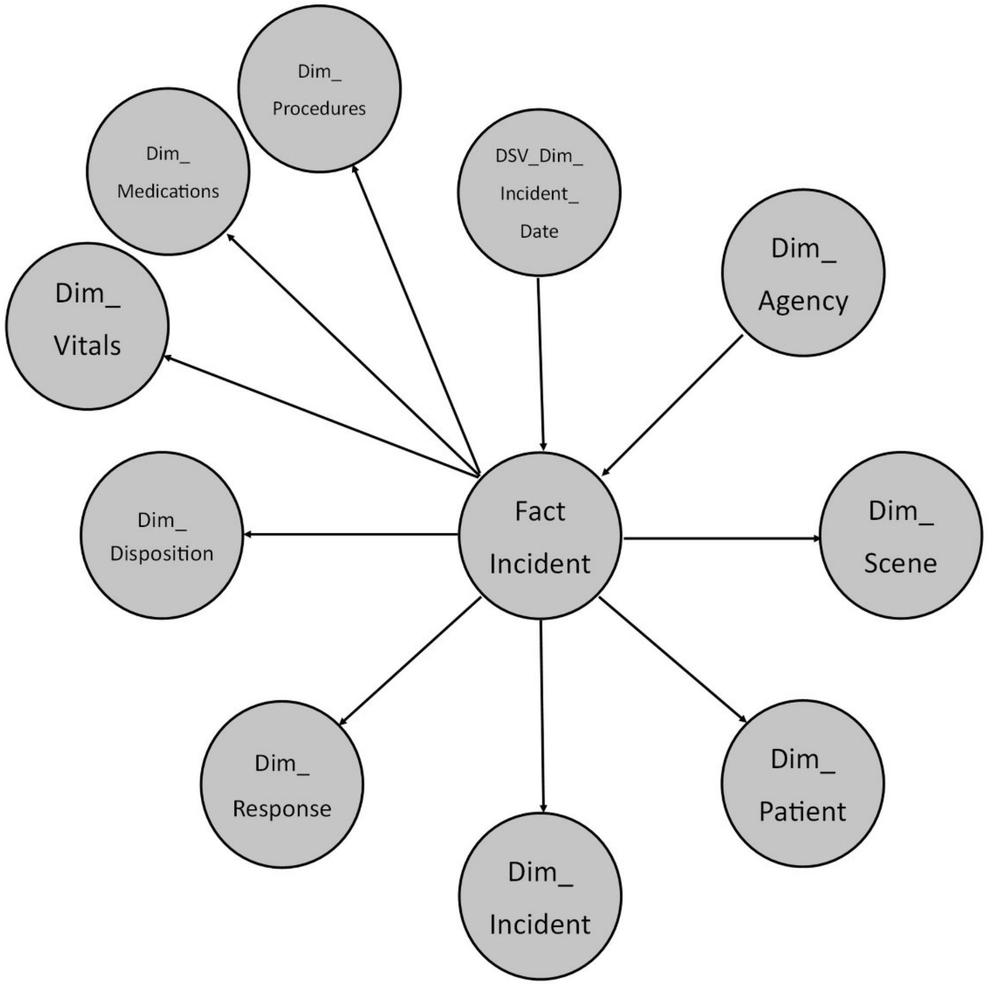


OR-EMSIS DataMart: The Star Schema

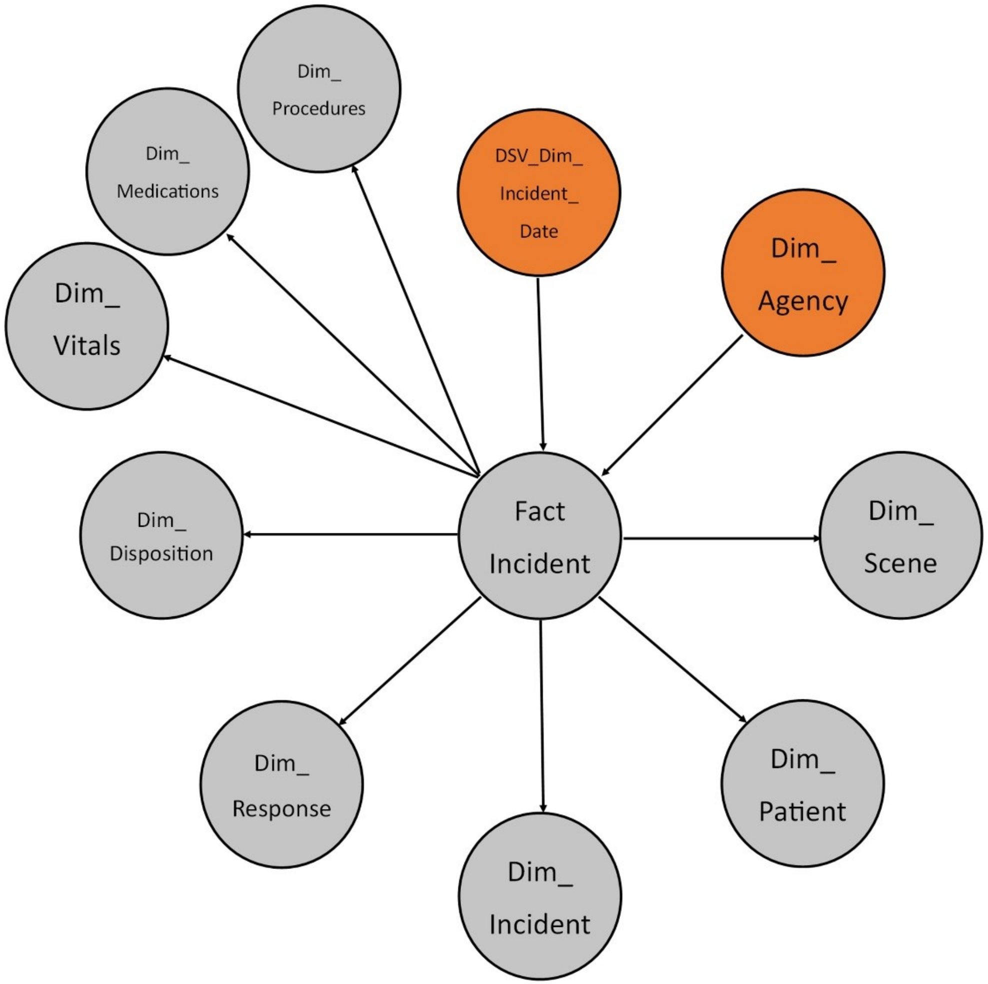
- Examples are based on our OR-EMSIS DataMart, but these methods are agnostic to database structure
- DataMart is arranged in a Star Schema
  - An efficient data model for query of relational data
  - A central Fact_Incident table is the center of the star
  - Radiating out to dimensional tables
  - Connected by primary and foreign keys


Dim_AirwayConfirmationMethod
Dim_PatientBarrier
Dim_MedSurgHistory
Dim_VitalsECGCardiacRhythm
Dim_AirwayConfirmationGroup
Dim_Patient
Dim_Airway
Dim_Vitals
Dim_InjuryMechanism
Dim_InjuryTraumaCriteria
Dim_Incident_One_To_One
Dim_PatientHistoryCurrentMedicationsGroup
Dim_SituationOtherAssociatedSymptom
Dim_SituationProviderSecondaryImpression
Dim_SituationPatientComplaintGroup
Dim_PatientMedicationAllergies
Dim_Protocol
Dim_PatientHistory
Dim_Situation
Dim_Narrative
Dim_Payment
Fact_Incident
Dim_Medication
Dim_Procedure
Dim_Outcome
Dim_SceneDelay
Dim_Response
Dim_Agency
Bridge_Incident_CrewMember
Dim_CrewMember
Dim_InjuryDetails
Dim_VitalsHighLowInitLast
Dim_InjuryRiskFactor
Dim_Scene
Dim_Disposition
Dim_Incident
Dim_Exposure
Dim_CardiacArrest
Dim_ResponseMode
Dim_DispositionHospitalTeamActivationGroup
Dim_InjuryCause
Dim_CardiacArrestResuscitationAttempt
Dim_CirculationReturned

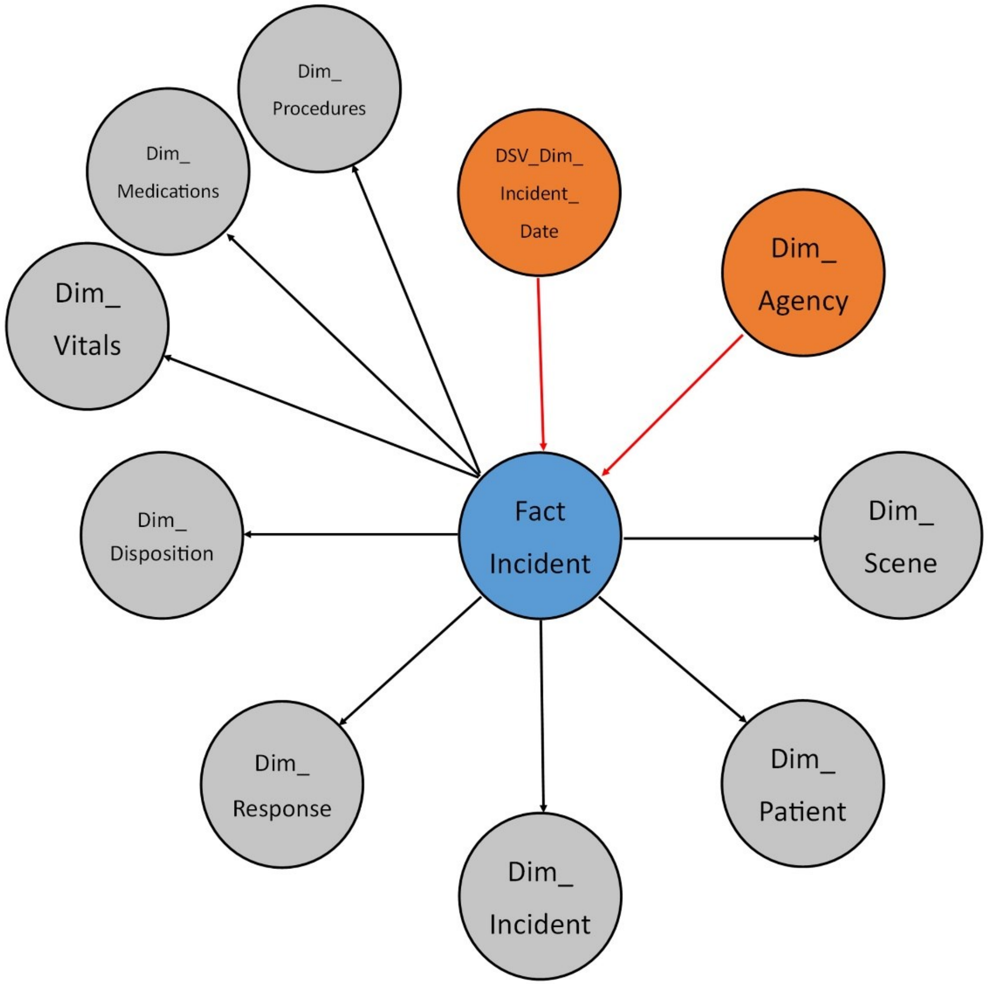
Star Schema: Order of Operations



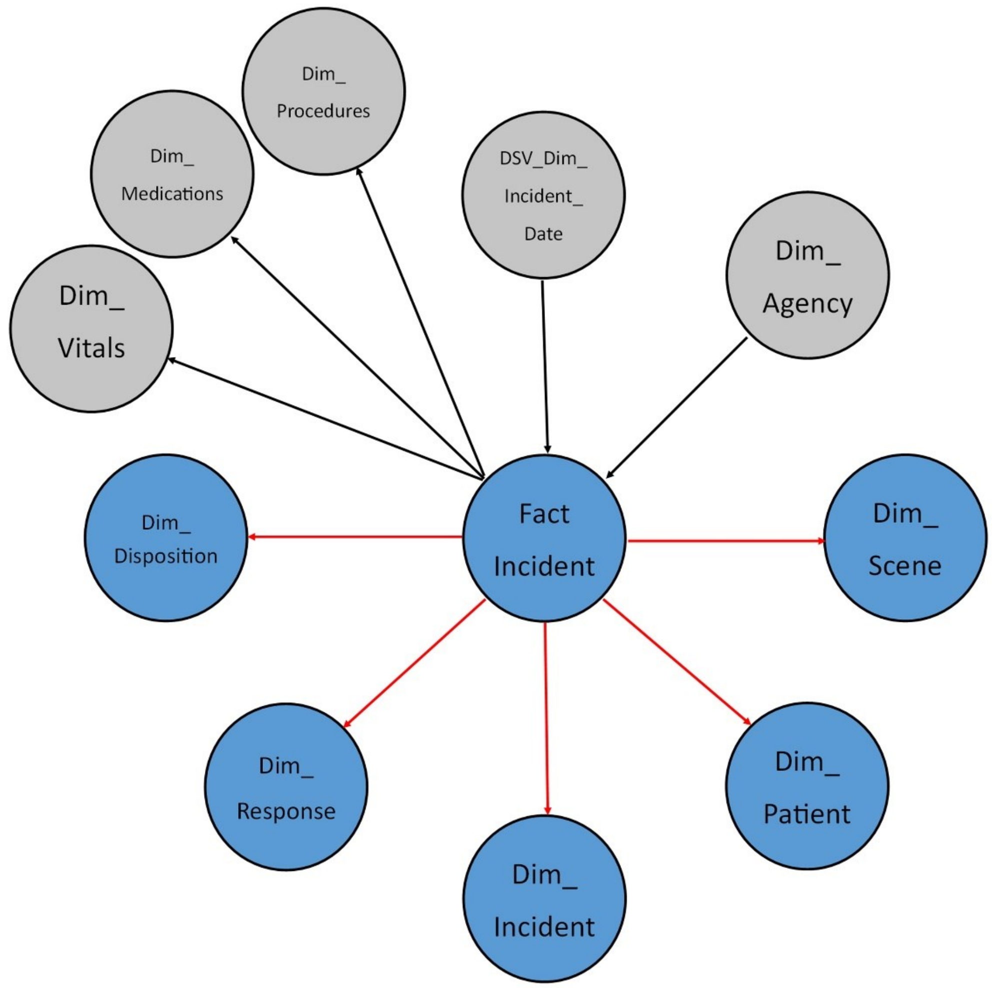
Star Schema: Order of Operations



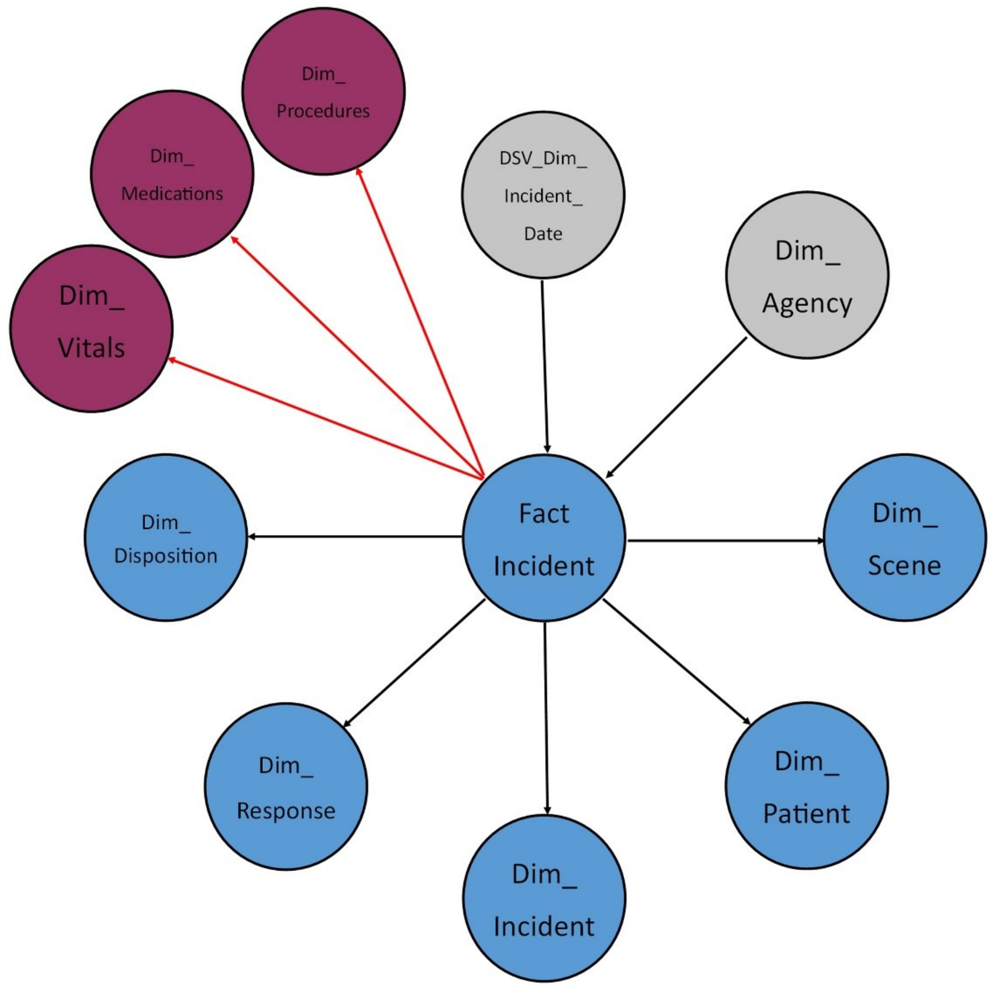
Star Schema: Order of Operations



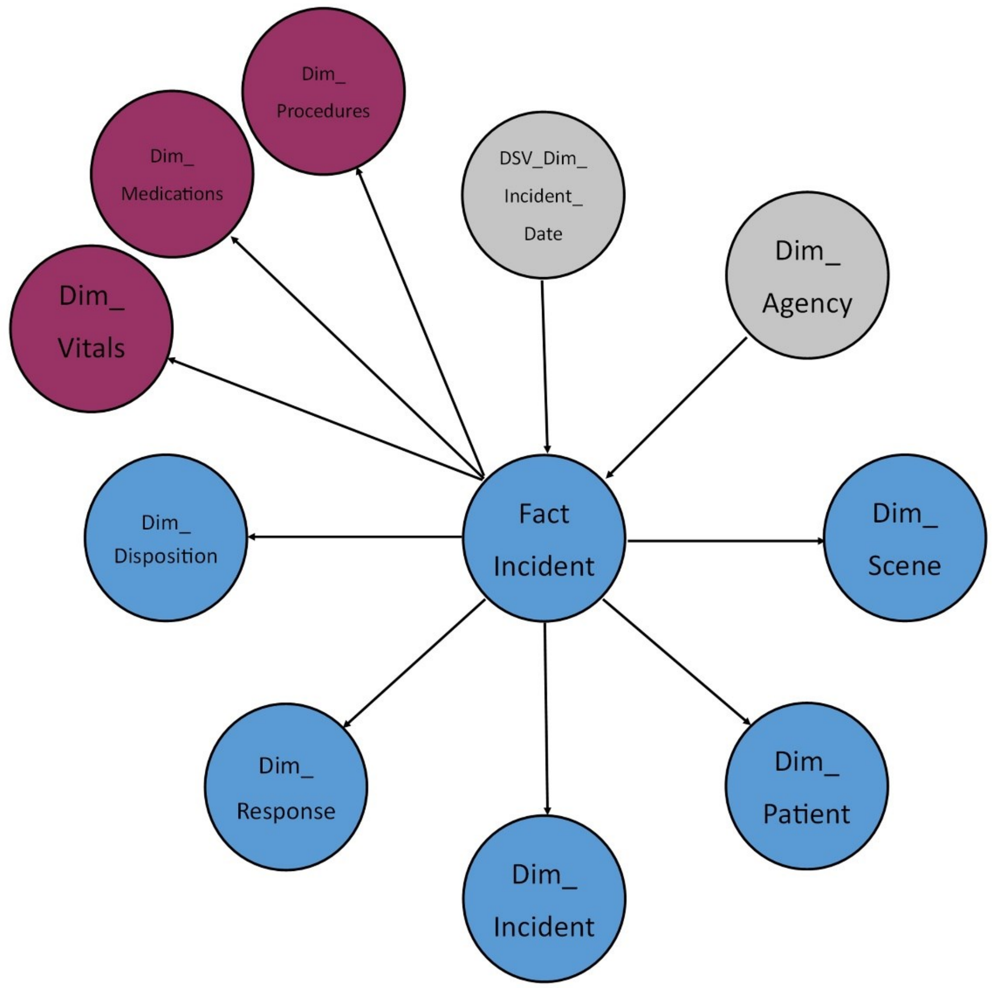
Star Schema: Order of Operations



Star Schema: Order of Operations



Star Schema: Order of Operations



Querying Your Database

We start many queries by hardcoding table and element names something like this:

```
fact_incident <- EMS_dbObject %>%
  tbl(in_schema("DwEms", "Fact_Incident")) %>%
  select(., Fact_Incident_PK,
         Agency_ID_Internal,
         Dim_Agency_FK,
         Dim_Incident_Date_FK,
         Dim_Incident_FK,
         Dim_Patient_FK,
         Dim_Disposition_FK,
         Patient_Age_In_Years,
         Patient_Weight_In_Kilograms) %>%
  filter(., Dim_Incident_Date_FK %in% local(dates) &
           Agency_ID_Internal %in% local(agency_id_int2)) %>%
  collect()
```

But if you are going to do this for dozens of tables, many projects, and update continually over time…

Querying Your Database

…it is a great opportunity to build a function:

```r
extract_db_table <- function(dbobject,
                             source_schema = NULL,
                             table_name,
                             element_list = everything(),
                             is_date = FALSE,
                             key_name = NULL,
                             value_list = NULL){

      ## Construct the tbl statement
      if (is.null(source_schema) == F) {
        source_table <- function(input, ...) {
          tbl(input, in_schema(source_schema, table_name))}
      } else {
        source_table <- function(input, ...) {
          tbl(input, table_name)}
      }

      ## Construct the filter statement
      if (is.null(key_name) == F) {
        criteria <- function(input, ...) {
          filter(input, !!rlang::sym(key_name) %in% local(value_list))}
      } else {
        criteria <- function(input, ...) {
          filter(input, TRUE)}
      }

      out <- dbobject %>%
        source_table(.) %>%
        select(., all_of(element_list))  %>%
        collect(.) %>%
        criteria(.)

      out
  }
```

You might generalize it to something like this…

Querying Your Database

Now we have a general function that can be used to pull data out of any database and any table with the following parameters:

- `EMS_dbObject = Name of the database`
- `source_schema = Name of schema (if applicable)`
- `table_name = Name of the table`
- `element_list = A vector containing the names of desired data elements (if applicable; default is everything())`
- `key_name = Name of the table key (if applicable)`
- `value_list = A vector of keys to match against (if applicable)`

Querying Your Database

- But where do these inputs come from?
- Here is an example of how the previous query could be performed using values from the data dictionary
- But what if I want to automate?
- This is where your data dictionary comes in!

```
############################################
##
##            Fact Incident                #
##
############################################

fact_elements <- dd %>% filter(., Table.Name == "Fact_Incident") %>% arrange(., query_order)

Fact_Incident <-  extract_db_table(EMS_dbObject,
                                    source_schema = fact_elements %>% distinct(., Schema) %>% pull(),
                                    table_name = "Fact_Incident",
                                    element_list = fact_elements %>% pull(Column.Name),
                                    key_name = "Dim_Incident_Date_FK",
                                    value_list = date_key) %>%
  filter(., Agency_ID_Internal %in% agency_key) %>%
  set_names(., fact_elements %>% pull(Element_Name))
```

```r
One2One_table_extract <- function(db, dictionary, crosswalk, table_name){
  ## Pulls the an extract of data elements in the specified table and puts them in order for query
  table_elements <- dictionary %>% filter(., Table.Name == table_name) %>% arrange(query_order)
  ## Extracts the name of the schema for elements in the list
  schema <- table_elements %>% distinct(., Schema) %>% pull()
  ## Pull the row from the crosswalk containing keys linking fact incident to specified table
  table_keys <- crosswalk %>%
    filter(., db_Table2_Name == table_name)
  ## Pull the key for the source table (in this case Fact_Incident)
  source_key <- table_keys %>% pull(., Table1_Foreign_Key)
  ## Pull the table name for the source table (in this case Fact_Incident)
  source_table_name <- table_keys %>% pull(., db_Table1_Name)
  ## Pull the key for the specified table being pulled
  table_key <- table_keys %>% pull(., Table2_Foreign_Key)
  ## Pull a list of keys from the source table to
  key_list <- source_table_name %>% get() %>% pull(., source_key)
  ## Call to extract_db_table that uses the information above to pull specified table from the
  temp <- extract_db_table(db,
                           source_schema = schema,
                           table_name = table_name,
                           element_list = table_elements %>% pull(Column.N
                           key_name = table_key,
                           value_list = key_list) %>%
    ## Rename data elements using NEMSIS/Standard Element #s
    set_names(., table_elements %>% pull(Element_Name))
  ## Join the new 1:1 Table with the analytic dataset
  analytic_dataset <<- analytic_dataset %>%
    left_join(., temp,
              by = setNames(table_key, source_key))
}
```

You can build a wrapper around extract_db_table like this!

Querying Your Database

Now we have a general function that will take the name of a table, pull the parameters out of your data dictionary and builds your query for you with the following parameters:

- `EMS_dbObject` = Name of the database
- `dictionary` = Name of an object containing your data dictionary
- `crosswalk` = Name of an object containing a crosswalk between tables
- `table_name` = Name of the table

Querying Your Database

- Here is a call to this new wrapper function
- But we have so many tables to pull data from!
- Won’t we have to call the function over and over?
- Can’t we build a function for that?

```
One2One_table_extract(db <- EMS_dbObject,
                      dictionary <- dd,
                      crosswalk <- tc,
                      table_name <- "Dim_Disposition")
```

```r
extract_all_one2one_tables <- function(db, dictionary, crosswalk) {
  ## Use crosswalk file to identify tables in the db that have a 1:1 relationship to the core
  table_list <- crosswalk %>%
    ## Filter step to pull only 1:1 tables
    filter(., Correspondence == "\`1:1") %>%
    ## Placeholder for the "Core" indicator
    filter(., db_Table2_Name %in% c("Dim_Agency", "DSV_Dim_Incident_Date") ==     LSE) %>%
    pull(., db_Table2_Name)
  ## Loop iterates through the list of tables created above and feeds them       2One_table_extract
  ## Also outputs a status update to tell you what table it is processing
  for(i in 1:length(table_list)){
    cat("Processing:", table_list[i], "\n")
    One2One_table_extract(db = db,
                          dictionary = dictionary,
                          crosswalk = crosswalk,
                          table_name = table_list[i])
  }
}
```

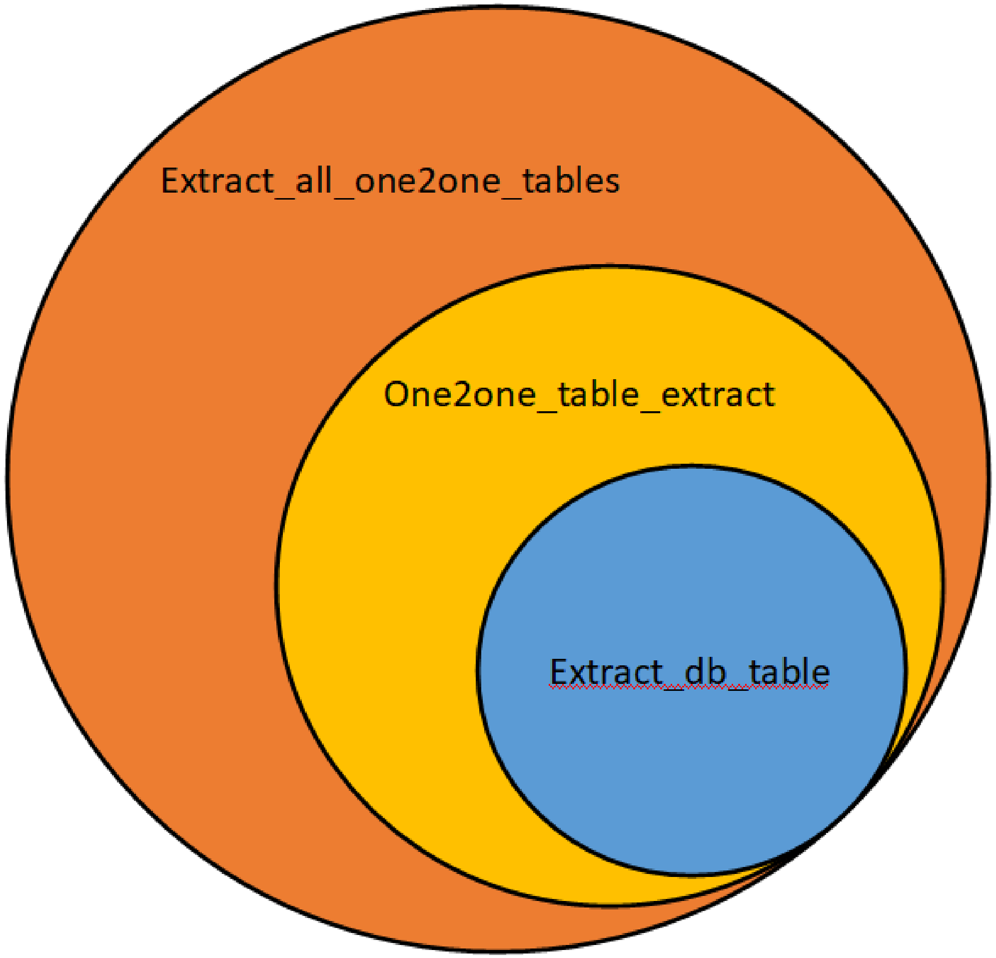
Here is a function that iterates through all 1:1 tables in your data dictionary!

Querying Your Database

- The function for 1:M tables is similar:
  - You can choose to roll the values up to the patient level and join to the analytic dataset at the end like the 1:1 tables
  - I prefer to keep them in separate tables linked by a key or unique identifier

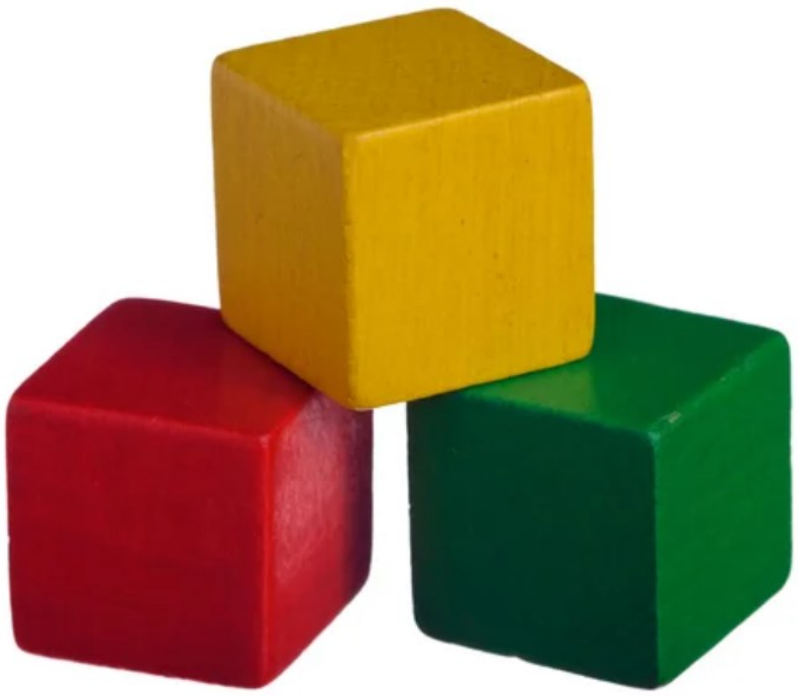
Takeaways: 1:1 Table Functions

- Functions are nested
- At the outermost circle fewest arguments:
  - db
  - dictionary
  - crosswalk
- All other arguments are created within the data pipeline as you move deeper



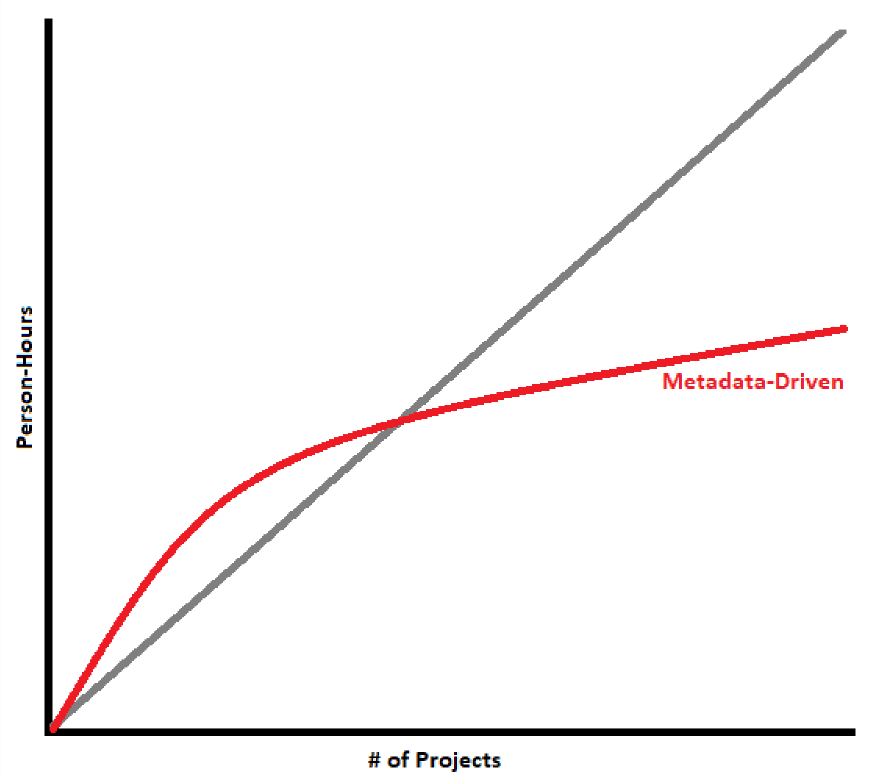
Modularity

- Functions may be used modularly- stacked like blocks to build data products and applications



Efficiency

- Initial time investment may be a bit greater to set up metadata-driven project workflows
- Over time increased efficiency creates an ever-widening gap representing time savings
- Makes current demand for data products sustainable for a small team



Data Pipelines

In Oregon we have stood up a Posit Team server for analytics and reporting.

Goals:

- Streamline data pipelines from source to publication
- DataMart > Posit Team > The Internet
- Automated update of data products (dashboards, reports, APIs, etc.)

IMAGETREND

DataMart

posit

POSIT TEAM

Monitoring

- Near real-time monitoring tools will enable us to track progress on data quality goals on an ongoing basis
- Dashboards will allow us to close the feedback loop providing data insights back to agencies and hospitals
  - Dashboards at local level
  - Comparison against statewide averages
  - Restricted access to agency only

Data Quality Dashboards

Oregon EMS & Trauma Systems Program, Public Health Division, Oregon Health Authority Last Updated: 2023-04-11

Summary | Reporting Volume | Timeliness | Completeness | Validity | Consistency

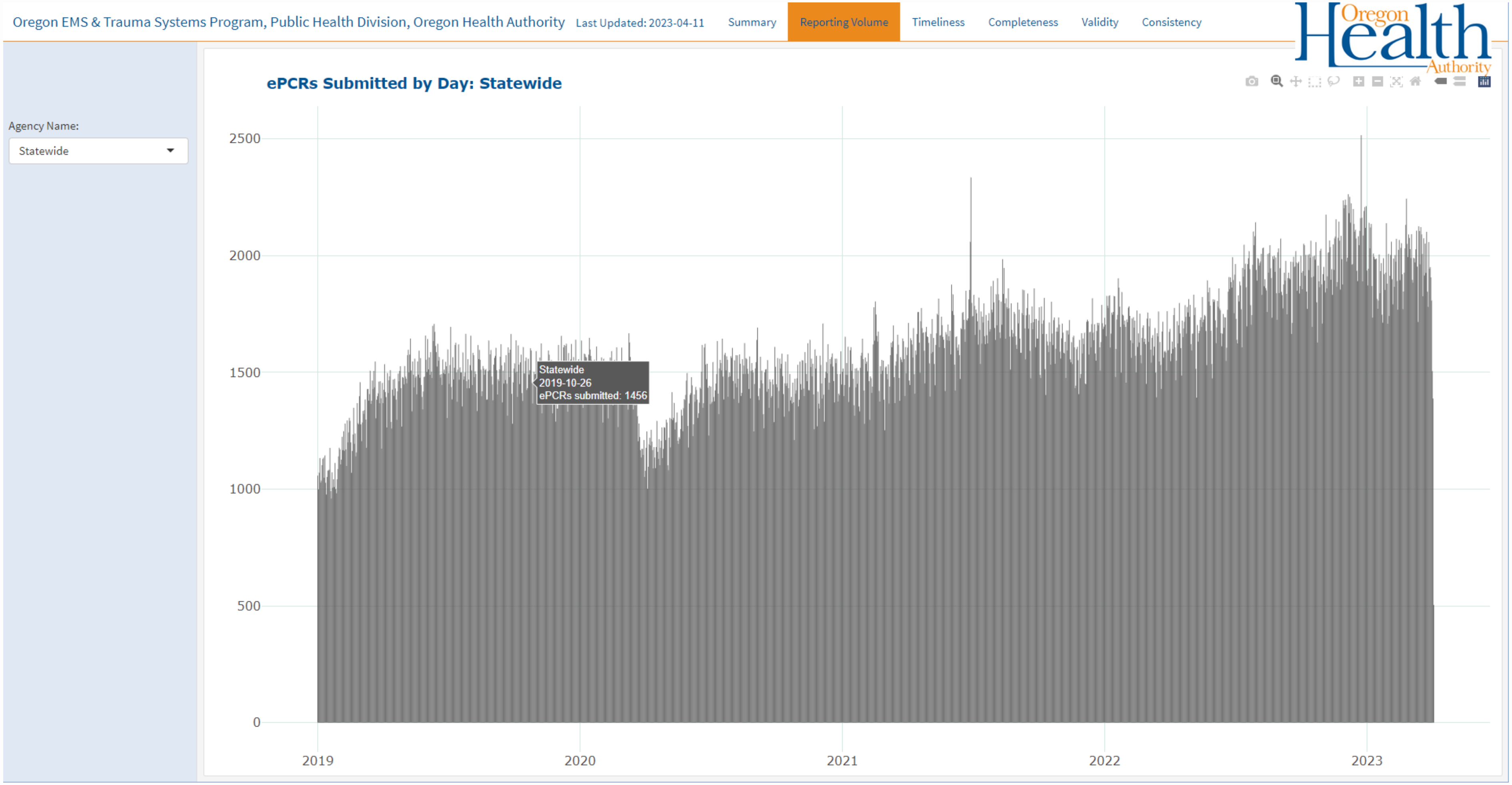
Agency Name:

Statewide

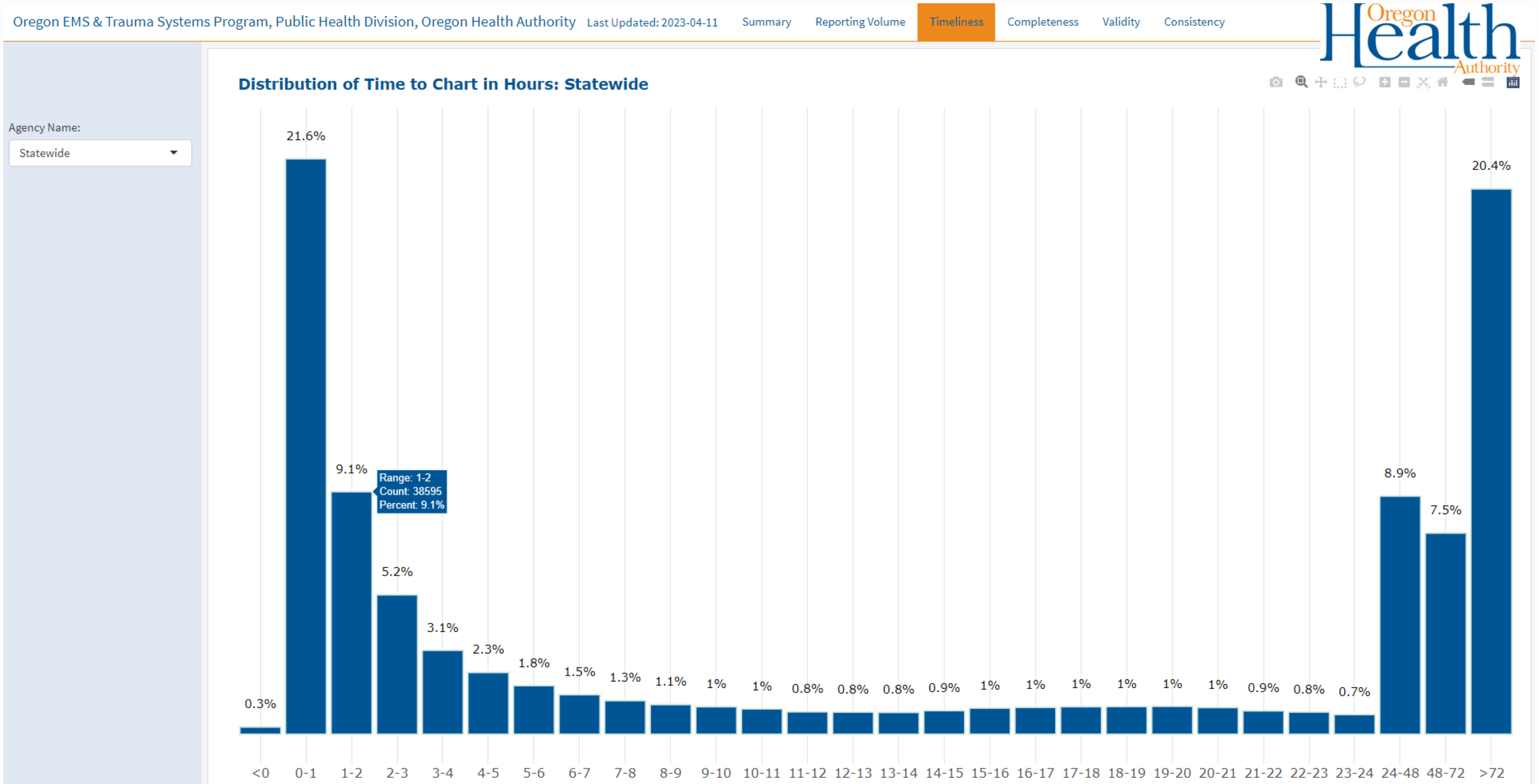
- Statewide
- Adventist Health Tillamook
- Adventure Medics LLC
- Agness Illahe Rural Fire Protection District
- American Medical Response Northwest Inc
- Banks Fire District

Label	Period	Value
Most Recent Submission	NA	2023-04-03
Last Month Submissions	Mar, 2023	61234
Last Year Submissions	2022	666982
Percent of ePCRs Under 24 Hours	2022	61.2 %

Data Quality Dashboards



Data Quality Dashboards



Data Quality Dashboards

Data Validation

- For each field check to see that values fall within specific constraints:
  - Length
  - Pattern
  - Etc.
- Check to see that imported data contains values that are in the state data set
- Highlight potential issues using bar and text color

Questions?

Oregon EMS & Trauma Program
EMS.TRAUMA@odhsoha.oregon.gov

Oregon Health Authority

(Enter) DEPARTMENT (ALL CAPS)
(Enter) Division or Office (Mixed Case)