IMS Health & Quintiles are now IQVIA®

Real-world Data Processing

with Open-source Tools and Standards

Masafumi Okada M.D. Ph.D.
IQVIA Solutions Japan K.K.

Table of Contents

Disclaimer: the contents in this presentation does not represent the thoughts and opinions of author's organization.

Motivation

- To handle *Real World Data* such as administrative claims database, the **first barrier** is often its large data size. Usually this kind of dataset will be stored in a relational database system, but it is sometimes troublesome **for statistical programmers** to operate the unfamiliar database system using SQL.

- And the **second barrier** might be its non-standardized column names. Programmers do **not** like much of definition documents written in Word or PDF.

- In addition to cope with these barriers, we need **a new way** to run our patient-level algorithm such as phenotyping on million-sized number of patient.

To enable handling data without learning SQL

- SQL is an old but good computer language. It is easy to learn, complete to describe any set from the database.
- However, when we would like to use table joins or sub-queries, the number of nested sentences will grow, thus it will be difficult to be read.
- Many programming language, including R, have same ”nested” problem. R users invented ”pipe” expression to solve this.
- If we can use pipe expression to query database, it would improve readability of code.
- For example, compare these two formulas that represents same query to the database:
  - `SELECT COUNT(*) AS subject_count, concept_name FROM @cdm.person INNER JOIN @cdm.concept ON person.gender_concept_id = concept.concept_id GROUP BY concept_name;`
  - `cdm.person %>% inner.join(cdm.concept, by=c(“gender_concept_id” = “concept_id”)) %>% group_by(concept_name) %>% summarise(subject_count=n())`

Large Data Manipulation without SQL – Apache Spark

- Apache Spark is an open-source distributed general-purpose cluster-computing framework.
- Spark's first programming language binding is scala, but it has R API.
- Using the API, a R package named sparklyr provides "pipe" syntax to query Spark dataframe.
- With sparklyr, clinical programmer …
  - Can load ~50GB CSV/TSV text into Spark cluster.
  - Can Extract/Transform data using normal dplyr functions. No SQL required.
  - With Spark cluster, can handle ~50GB data from 8GB memory PC.
  - Can apply an algorithm for each patient's data in parallel on Spark cluster.

Distributed R

- If you are heavy user of tidyverse, you might say: “We do not need to use Spark cluster to handle large data in pipe manner, we have dbplyr.”
- We have one more large motivation to use Spark: Distributed R.
- This is a new way to run our patient-level algorithm on million-sized number of patient.
- Details will be introduced later.

Simple Standard for variables and vocabularies

- To avoid learning new definition of variables with variety of column names for every study, WE NEED **STANDARDS**!
- Yes, we already have CDISC standards, but it is somewhat complicated for observational studies.
- For those who do not need standardization of study protocol, CRF, non-clinical data, TFL…
- There is a standard that only covers **variables** and **vocabularies**. Developed for database studies.

OMOP CDM

Simple Standard for Column Names/Types – OMOP CDM

- Standardization of variables
  - Definition of column names
  - Definition of column types: numeric, text, or date
- Standardization for vocabularies,
  - SNOMED for observations / procedures
  - RxNorm for prescriptions
  - LOINC for measurements
- That's all, no PDF standard document.
- All documents are public domain, located at GitHub

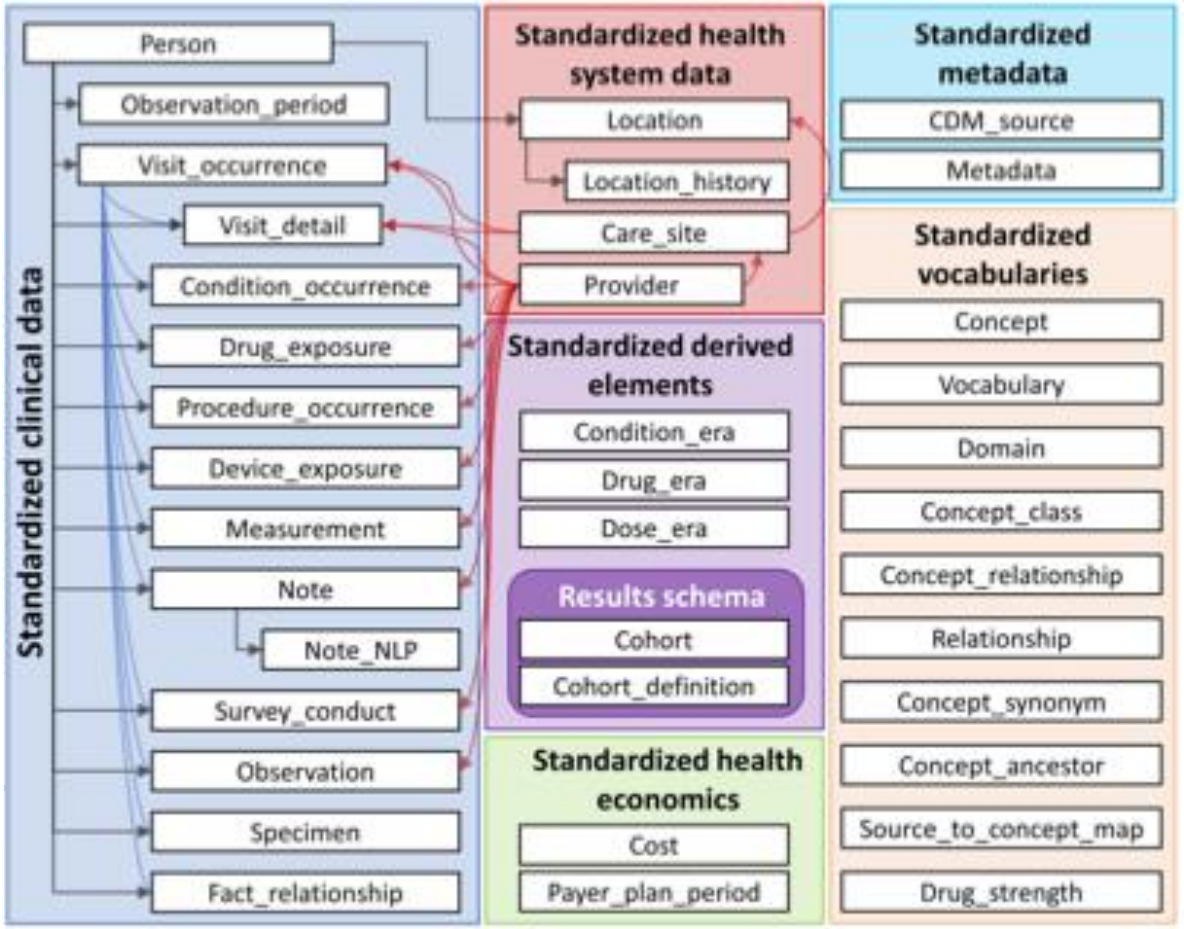


Figure Quoted from the Book of OHDSI (Public Domain)

Open source data analyze scripts, assuming the standard CDM

- “Variable name and type are standardized” means “Same script can be run on any data source”.
- OHDSI Community: a multi-stakeholder, interdisciplinary collaborative, develops many data handling scripts and implementation of statistical methods, assuming the data is stored in OMOP CDM database. We can copy it from GitHub, then can run without modification if we also use OMOP CDM database.

Example – Preparation

- Now demonstrate using Centers for Medicare and Medicaid Services (CMS) Medicare Claims Synthetic Public Use Files (SynPUF) 5% sample converted to OMOP CDM v5, draw histogram of age among CKD patients.
- This code uses standardized name of columns only. Thus we can re-use this code for any other data sources which follows OMOP CDM.

```
> library(sparklyr)
> library(dplyr)
> spark_install()                          # Download and Install spark for local use automatically
Installing Spark 2.4.3 for Hadoop 2.7 or later.
Downloading from:
---Snip---
Installation complete.
> sc <- spark_connect(master="local")      # Connect to spark local instance.
* Using Spark: 2.4.3
> object.size(sc)                          # spark object itself is very small
7208 bytes
```

Example – Load text data into spark

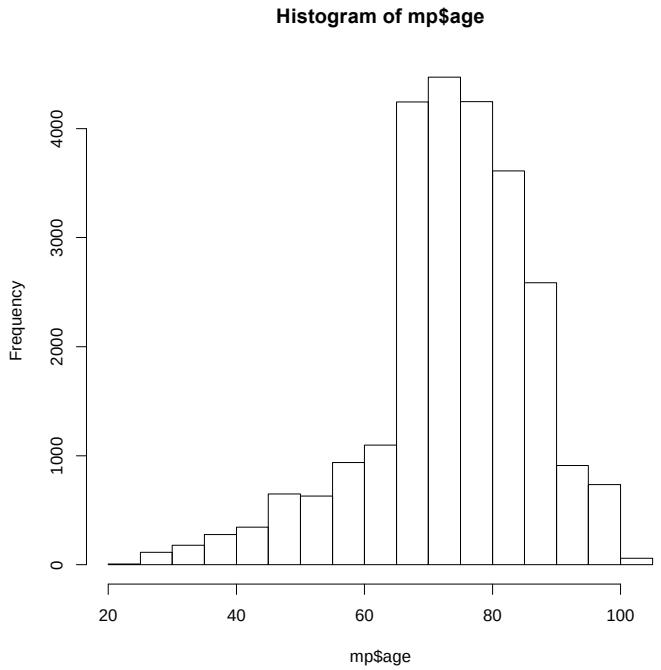
```
> # Load CSV large text file(1,637,181,751bytes) to spark. It took ~83 secs on my Desktop PC. This file has “Condition_occurence” standardized table.
> system.time(condition_occurence <- spark_read_csv(sc,"~/Downloads/synpuf5pct_20180710/condition_occurrence.csv", delimiter='¥t', header=FALSE, memory=FALSE))
   ユーザ   システム      経過
    0.469     0.134     82.827
> # Data was loaded into Spark, not R. So the size of object is only 10,808 bytes.
> object.size(condition_occurence)
10808 bytes
> # DeSynPUF CSV file does not have headers, but the order of columns follow the OMOP CDM v5 standard. So we can name these safely.
> co2 <- condition_occurence %>%
select( `person_id`=V2 ,`condition_concept_id`=V3 ,`condition_start_date`=V4 , `condition_end_date`=V6 )
```

Example – Load another data

```
> # Load another file (7,811,593bytes) to spark. This file has “Person” standardized table.
> person <- spark_read_csv(sc, "~/Downloads/synpuf5pct_20180710/person.csv", delimiter='¥t',header=FALSE)
> # birth_datetime is not mandatory field in CDM v5, so generate it from year_of_birth, month_of_birth, and
day_of_birth using R function paste()
> ps <- person %>% select(person_id=V1, year_of_birth=V3, month_of_birth=V4, day_of_birth=V5) %>%
mutate(birth_datetime=to_date(paste(year_of_birth, month_of_birth, day_of_birth, sep="-")))
```

Example – Select patients, calculate age, then visualize

```
> # From Condition_occurence table, first select CKD diagnosis (concept ID: 46271022), then pick up first date of diagnosis: min(condition_start_date). Next join this to Person table, calculate age by SparkSQL function months_between(). Finally copy the result in spark into R workspace by collect()
> mp <- co2 %>% filter(condition_concept_id == 46271022 ) %>% group_by(person_id) %>% summarise(first=min(condition_start_date)) %>% left_join(ps, by='person_id') %>% mutate(age=months_between(first, birth_datetime)/12) %>% select(person_id, age) %>% collect
> # Resulted data has 25101 patients. This is normal R dataframe, so we can use hist() to draw histogram.
> nrow(mp)
[1] 25101
> hist(mp$age)
```



Example – Distributed R

```
> # For Hypothyroidism patient(concept id=140673), calculate number of diagnosis and IQR of duration. Though this simple calculation can also be done by SQL only, you can extend this function to any more complex algorithms.
> co2 %>%
  filter(condition_concept_id == 140673) %>%
  mutate(conddays = datediff(condition_end_date, condition_start_date)) %>%
  spark_apply(function(df) {  # in this function(), you can pass normal R function                                                    #
and let spark R run the function distributed with
                                                      # cluster nodes.
    data.frame(IQR = IQR(df$conddays), n = nrow(df))
  }, group_by = "person_id")
# Source: spark<?> [?? x 3]
   person_id   IQR     n
       <int> <dbl> <int>
 1    100028     0     2
 2    100066     0     3
 3    100085     0     1
 4    100153     0     1
 5    100220     0     1
```

Summary & Further Information

- With OHDSI OMOP Clinical Data Models along with Spark+R environment, we can build up analytic environment with following feature by tools these were totally open-sourced.
  - Extract/Transform/Load large data without learning or writing SQL syntax
  - Write and reuse analysis scripts without learning various variable definitions for each protocol
  - Run your algorithms in parallel for each patient, visit, or treatment
- These will help clinical programmers' communication and collaborative work.

- For further information, please refer to:
  - The Book of OHDSI : https://ohdsi.github.io/TheBookOfOhdsi/
  - *Javier Luraschi, Kevin Kuo, Edgar Ruiz.* Mastering Apache Spark with R. https://therinspark.com