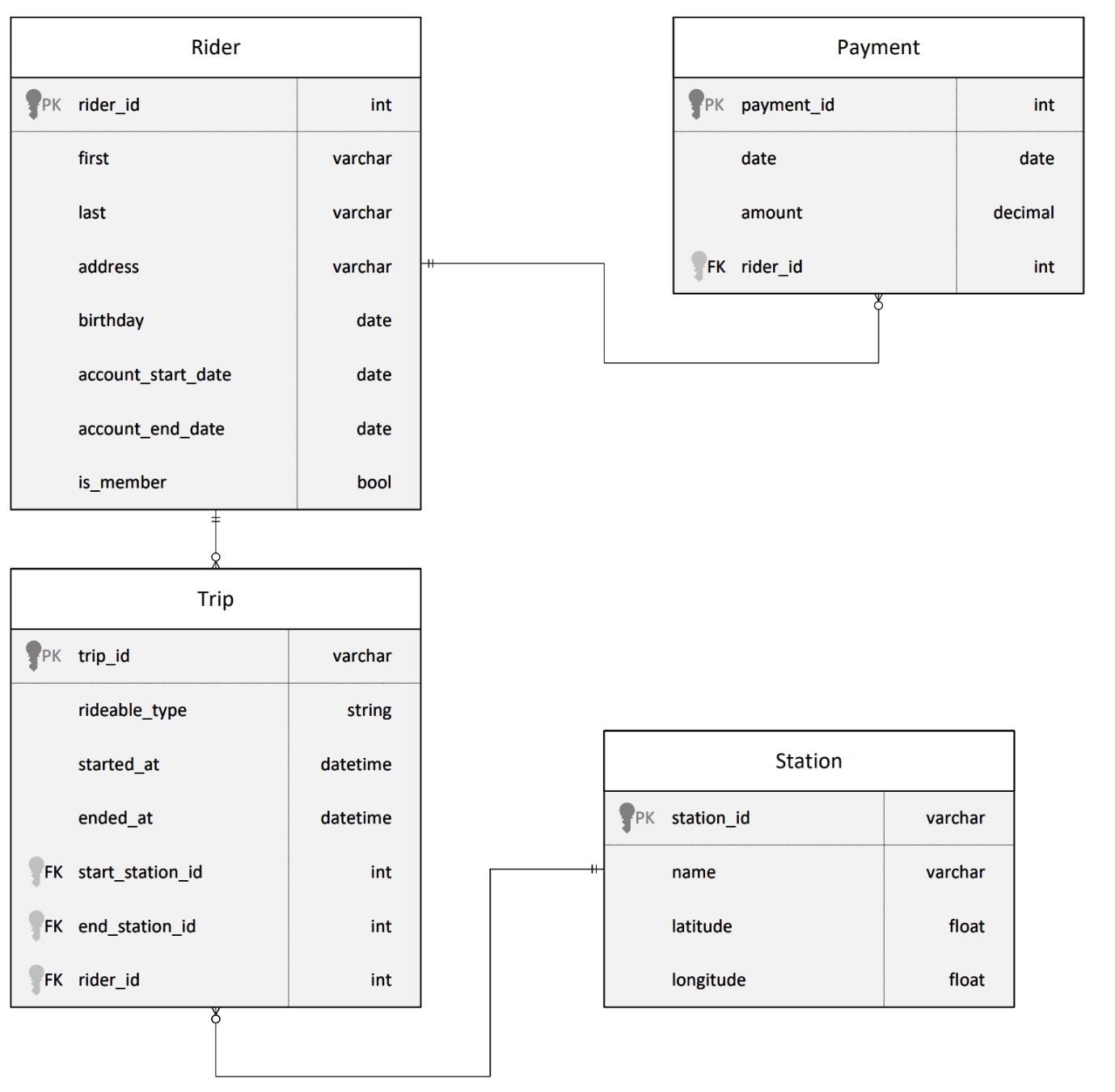
Divvy Bike Data Lake Project



The dimensional model should have two fact tables sharing common dimensions where applicable.

One should be related to **trip facts** and another should be related to **payment facts**.

The **trip fact** should have a fields for **trip duration** and **rider age** at time of trip.

The **payment fact** should have a field related to **amount of payment**.

The star schema should have dimensions related to the **trip fact**

table that are related to: riders, stations, and dates.

The schema should have dimensions related to the **payment fact** table that are related to: dates and riders.

The goal of this project is to develop a data lake solution using Azure Databricks using a lake house architecture. You will:

* Design a star schema based on the business outcomes listed below;
* Import the data into Azure Databricks using Delta Lake to create a Bronze data store;
* Create a gold data store in Delta Lake tables;
* Transform the data into the star schema for a Gold data store;

The business outcomes you are designing for are as follows:

1. Analyze how much time is spent per ride
   * Based on date and time factors such as day of week and time of day
   * Based on which station is the starting and / or ending station
   * Based on age of the rider at time of the ride
   * Based on whether the rider is a member or a casual rider
2. Analyze how much money is spent
   * Per month, quarter, year
   * Per member, based on the age of the rider at account start
3. EXTRA CREDIT - Analyze how much money is spent per member
   * Based on how many rides the rider averages per month
   * Based on how many minutes the rider spends on a bike per month

On the next page are instructions for logging into an Azure account where you can configure the resources, Azure Synapse Workspace, and data storage to complete the project.

Microsoft Azure Databrics Configuration.

* Hint: To view your DBFS files, enable the DBFS file browser in Databricks by going to Admin Console -> Workspace Settings -> Advanced
* Hint: If you are going to use PySpark Pandas, make sure you create your Spark Cluster using a Databricks runtime >= 10.0

# **Project: Azure Data Lakehouse Project**

## **Star Schema Design**

| **Success Criteria** | **Specifications** |
| --- | --- |
| The student will be able to generate fact tables based on a business need and a relational model | The dimensional model should have two fact tables sharing common dimensions where applicable. One should be related to trip facts and another should be related to payment facts. The trip fact should have a fields for trip duration and rider age at time of trip. The payment fact should have a field related to amount of payment. |
| The student will be able to generate dimension tables based on business needs and a relational model | The star schema should have dimensions related to the trip fact table that are related to: riders, stations, and dates. The schema should have dimensions related to the payment fact table that are related to: dates and riders. |

## **Extract Step**

| **Success Criteria** | **Specifications** |
| --- | --- |
| Produce Spark code in Databricks using Jupyter Notebooks and Python scripts | The notebook should contain Python code to extract information from CSV files stored in Databricks and write it to the Delta file system. |
| Use distributed data storage using Azure Data Storage options | The notebook should contain Python code that picks files up from the Databricks file system storage and writes it out to Delta file locations. |

## **Load Step**

| **Success Criteria** | **Specifications** |
| --- | --- |
| Implement key features of data lakes on Azure | The notebook should contain code that creates tables and loads data from Delta files. The learner should use spark.sql statements to create the tables and then load data from the files that were extracted in the Extract step. |

## **Transform Step**

| **Success Criteria** | **Specifications** |
| --- | --- |
| Use Spark and Databricks to run ELT processes by creating fact tables | The fact table Python scripts should contain appropriate keys from the dimensions. In addition, the fact table scripts should appropriately generate the correct facts based on the diagrams provided in the first step. |
| Use Spark and Databricks to run ELT processes by creating dimension tables | The dimension Python scripts should match the schema diagram. Dimensions should generate appropriate keys and should not contain facts. |
| Produce Spark code in Databricks using Jupyter Notebooks and Python scripts | The transform scripts should at minimum adhere to the following: should write to delta; should use overwrite mode; save as a table in delta. |

{"code":"AuthorizationFailed","message":"The client 'odl\_user\_244421@udacityhol.onmicrosoft.com' with object id '4979a48f-c0f8-488b-994f-5f363b0353d5' does not have authorization to perform action 'Microsoft.Resources/deployments/validate/action' over scope '/subscriptions/8b29b552-f66a-4753-a52b-4c6cda8e9680/resourceGroups/ODL-DataEng-244421/providers/Microsoft.Resources/deployments/ODL-DataEng-244421\_louis-databricks-ws1' or the scope is invalid. If access was recently granted, please refresh your credentials."}

**Working with Hive**

df=spark.sql("show databases")

df.show()

tables = spark.sql("show tables").show()

df1=spark.sql("select \* from drivers\_table limit 5").show()

from pyspark.sql import HiveContext

hive\_context = HiveContext(sc)

bank = hive\_context.table("default.bank")

bank.show()

**This is how I initialized  sc to get the hive table records and not just the metadata of it**

from pyspark import SparkConf, SparkContext

conf = SparkConf().setMaster("yarn-client")

sc = SparkContext(conf =conf)

from pyspark import HiveContext

hive\_context=HiveContext(sc)

data=hive\_context.table("database\_name.table\_name")

data.registerTempTable("temp\_table\_name")

hive\_context.sql("select \* from temp\_table\_name limit 10").show()

# Change Schema with CSV file

# Import the libraries SparkSession, StructType,

# StructField, StringType, IntegerType

from pyspark.sql.types import \*

#Create PySpark dataframes with data from /FileStore/tables/udacity

# Define the structure for the data frame

payments\_schema = StructType([ \

    StructField('payments\_id', IntegerType(), False), \

    StructField('date', DateType(), True), \

    StructField('amount', DecimalType(), True), \

    StructField('rider\_id', IntegerType(), True) \

])

payments\_df = spark.read.format("csv") \

    .option("inferSchema", "False") \

    .schema(payments\_schema)  \

    .option("header", "True") \

    .option("sep", ",") \

.load("/FileStore/tables/udacity/payments.csv")

# -----------------------------------------------

riders\_schema = StructType([ \

    StructField('rider\_id', IntegerType(), False), \

    StructField('first', VarcharType(256), True), \

    StructField('last', VarcharType(256), True), \

    StructField('address', VarcharType(256), True), \

    StructField('birtday', DateType(), True), \

    StructField('account\_start\_date', DateType(), True), \

    StructField('account\_end\_date', DateType(), True), \

    StructField('is\_member', BooleanType(), True) \

])

riders\_df = spark.read.format("csv") \

    .option("inferSchema", "False") \

    .schema(riders\_schema\_schema)  \

    .option("header", "TRUE") \

    .option("sep", ",") \

.load("/FileStore/tables/udacity/riders.csv")

# ------------------------------------------------

stations\_schema = StructType([ \

    StructField('station\_id', VarcharType(256), False), \

    StructField('name', Varchartype(256), True), \

    StructField('latitude', FloatType(), True), \

    StructField('longitude', FloatType(), True) \

])

stations\_df = spark.read.format("csv") \

    .option("inferSchema", "False") \

    .schema(stations\_schema\_schema)  \

    .option("header", "True") \

    .option("sep", ",") \

.load("/FileStore/tables/udacity/stations.csv")

# --------------------------------------------------

trips\_schema = StructType([ \

    StructField('trip\_id', VarcharType(256), False), \

    StructField('rideable\_type', StringType(), True), \

    StructField('started\_at', TimestampNTZType(), True), \

    StructField('ended\_at', TimestampNTZType, True) \

])

trips\_df = spark.read.format("csv") \

    .option("inferSchema", "False") \

    .schema(trips\_schema)  \

    .option("header", "True") \

    .option("sep", ",") \

.load("/FileStore/tables/udacity/trips.csv")

# Change Schema with Hive

[Database(name='default', catalog='spark\_catalog', description='Default Hive database', locationUri='dbfs:/user/hive/warehouse')]

#Create PySpark dataframes with data from /FileStore/tables/udacity

# Define the structure for the data frame

payments\_schema = StructType([ \

    StructField('payments\_id', IntegerType(), False), \

    StructField('date', DateType(), True), \

    StructField('amount', DecimalType(), True), \

    StructField('rider\_id', IntegerType(), True) \

])

payments\_df = spark.read.format("Hive") \

    .schema(payments\_schema) \

    .option("header", "True") \

.load("/delta/bronze\_payments")

# Change Schema with Dataframe

from pyspark.sql.types import \*

# Define the structure for the data frame

payments\_schema = StructType([ \

    StructField('payments\_id', IntegerType(), False), \

    StructField('date', DateType(), True), \

    StructField('amount', DecimalType(), True), \

    StructField('rider\_id', IntegerType(), True) \

])

silver\_payments\_df = spark.createDataFrame(df.rdd, payments\_schema)

#Create PySpark dataframes with data from /FileStore/tables/udacity

payments\_df = spark.read.format("csv") \

    .option("inferSchema", "false") \

    .option("header", "false") \

    .option("sep", ",") \

.load("/FileStore/tables/udacity/payments.csv")

riders\_df = spark.read.format("csv") \

    .option("inferSchema", "false") \

    .option("header", "false") \

    .option("sep", ",") \

.load("/FileStore/tables/udacity/riders.csv")

stations\_df = spark.read.format("csv") \

    .option("inferSchema", "false") \

    .option("header", "false") \

    .option("sep", ",") \

.load("/FileStore/tables/udacity/stations.csv")

trips\_df = spark.read.format("csv") \

    .option("inferSchema", "false") \

    .option("header", "false") \

    .option("sep", ",") \

.load("/FileStore/tables/udacity/trips.csv")

# View dataframes

# dataframe.printSchema()

# dataframe.show(2)

# display(riders\_df)

# Write dataframes into Delta Lake environment

payments\_df.write.format("delta") \

    .mode("overwrite") \

.save("/delta/bronze")

riders\_df.write.format("delta") \

    .mode("overwrite") \

.save("/delta/bronze")

stations\_df.write.format("delta") \

    .mode("overwrite") \

.save("/delta/bronze")

trips\_df.write.format("delta") \

    .mode("overwrite") \

.save("/delta/bronze")

# Create databases within Delta Lake environment

spark.sql("CREATE TABLE payments(payment\_id INT, date DATE, amount DECIMAL, rider\_id INT) USING DELTA LOCATION '/delta/bronze\_payments'")

spark.sql("CREATE TABLE riders(rider\_id INT, first\_name VARCHAR(256), last\_name VARCHAR(256), address VARCHAR(256), birthday DATE, account\_start\_date DATE, account\_end\_date DATE, is\_member BOOLEAN)" + \

          "USING DELTA LOCATION '/delta/bronze\_riders'")

spark.sql("CREATE TABLE stations(station\_id VARCHAR(256), name VARCHAR(256), latitude FLOAT, longitude FLOAT) USING DELTA LOCATION '/delta/bronze\_stations'")

spark.sql("CREATE TABLE trips(trip\_id VARCHAR(256), rideable\_type STRING, started\_at TIMESTAMP, ended\_at TIMESTAMP, rider\_id INT) USING DELTA LOCATION '/delta/bronze\_trips'")

ChatGPT

from pyspark.sql import SparkSession

from pyspark.sql.types import StructType, StructField, IntegerType, DateType, DecimalType

# Create a Spark session

spark = SparkSession.builder.appName("example").getOrCreate()

# Define the schema

payments\_schema = StructType([

StructField('payments\_id', IntegerType(), False),

StructField('date', DateType(), True),

StructField('amount', DecimalType(), True),

StructField('rider\_id', IntegerType(), True)

])

# Load the DataFrame from Delta Lake

bronze\_payments\_df = spark.read.format("delta").load("/delta/bronze\_payments")

# Apply the schema using select to create the silver\_payments\_df DataFrame

silver\_payments\_df = bronze\_payments\_df.select(

bronze\_payments\_df['payments\_id'],

bronze\_payments\_df['date'].cast('date'), # Cast the 'date' column to DateType

bronze\_payments\_df['amount'].cast('decimal(10,2)'), # Cast the 'amount' column to DecimalType

bronze\_payments\_df['rider\_id'].cast('int') # Cast the 'rider\_id' column to IntegerType

)

# Save as a Delta table

silver\_payments\_df.write.format("delta").mode("overwrite").saveAsTable("silver\_payments\_tbl")

from pyspark.sql import SparkSession

from pyspark.sql.types import StructType, StructField, IntegerType, DateType, DecimalType

from datetime import datetime

# Create a Spark session

spark = SparkSession.builder.appName("example").getOrCreate()

# Sample data in RDD

data = [(1, '2022-01-01', '10.5', 100),

(2, '2022-02-01', '15.75', 200)]

# Define the schema

payments\_schema = StructType([

StructField('payments\_id', IntegerType(), False),

StructField('date', DateType(), True),

StructField('amount', DecimalType(), True),

StructField('rider\_id', IntegerType(), True)

])

# Create an RDD

rdd = spark.sparkContext.parallelize(data)

# Define a function to cast the elements

def cast\_and\_transform(row):

payments\_id, date\_str, amount\_str, rider\_id = row

# Perform the casting within the transformation function

return (payments\_id, datetime.strptime(date\_str, '%Y-%m-%d').date(), float(amount\_str), rider\_id)

# Apply the transformation using map

casted\_rdd = rdd.map(cast\_and\_transform)

# Create a DataFrame from the casted RDD

df = spark.createDataFrame(casted\_rdd, schema=payments\_schema)

# Save as a Delta table

df.write.format("delta").mode("overwrite").saveAsTable("silver\_payments\_tbl")