//Youtube Video:

--My Youtube Channel: youtube.com/c/joycekayavila

--Link to the specific video associated with this tutorial: https://youtu.be/zb89rvl4_Xs

---

//PREWORK - ENVIRONMENT SETUP

//Let's set the context for this worksheet
//Note: for simplicity, the ACCOUNTADMIN role will be used for this tutorial.
--but this would not likely be the case in a real production environment.
```
USE ROLE ACCOUNTADMIN;
USE WAREHOUSE COMPUTE_WH;
```

//At the time of this tutorial, there were 15 million records in this table
```
SELECT COUNT(1) FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.ORDERS;
```

//At the time of this tutorial, there were 1.5 million records in this table
```
SELECT COUNT(1) FROM
SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.CUSTOMER;
```

//Create two virtual warehouses, one for generic use
--and the other for use in the raw data layer for data pipelines.

//Note: To make it easy to go back to any section of the tutorial,
--we'll use the 'CREATE OR REPLACE' command to create new objects.
--The 'CREATE OR REPLACE' command is not recommended for common use in the production environment.

//Note: We are creating two new virtual warehouses but you can create as many virtual warehouses,
--in varying sizes and configurations, that you need.

//Tip: You may want to consider creating separate virtual warehouses for different layers
--in the Data Vault architecture such as DV_BDV_WH and DV_INFO_WH

```
CREATE OR REPLACE WAREHOUSE DV_GENERIC_WH
```

```
   WITH WAREHOUSE_SIZE = 'XSMALL' AUTO_SUSPEND = 300
AUTO_RESUME = TRUE INITIALLY_SUSPENDED = TRUE;
CREATE OR REPLACE WAREHOUSE DV_RDV_WH
   WITH WAREHOUSE_SIZE = 'XSMALL' AUTO_SUSPEND = 300
AUTO_RESUME = TRUE INITIALLY_SUSPENDED = TRUE;

//Use the new Generic virtual warehouse now for the tutorial
USE WAREHOUSE DV_GENERIC_WH;

//Create new Snowflake database and schemas to be used in the tutorial
CREATE OR REPLACE DATABASE DV_TUTORIAL;
CREATE OR REPLACE SCHEMA L00_STG  COMMENT = 'Schema for Staging
Area objects';
CREATE OR REPLACE SCHEMA L10_RDV  COMMENT = 'Schema for Raw Data
Vault objects';
CREATE OR REPLACE SCHEMA L20_BDV  COMMENT = 'Schema for Business
Data Vault objects';
CREATE OR REPLACE SCHEMA L30_INFO COMMENT = 'Schema for Information
Delivery objects';



--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------


//STAGING AREA SETUP - NATION & REGION

//Set Context.  Use the schema for Staging area objects - L00.STG
USE SCHEMA L00_STG;

//Create two new staging tables for static reference data
--LDTS = Load Data Timestamp
--RSCR = Reference Source (Static Reference Data)

//Create Nation Stage Table
CREATE OR REPLACE TABLE STG_NATION
AS
SELECT  SRC.*,
       CURRENT_TIMESTAMP() LDTS,
       'STATIC REFERENCE DATA' RSCR
FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.NATION SRC;

//Validate
```

```
SELECT * FROM STG_NATION;

//Create Region Table
CREATE OR REPLACE TABLE STG_REGION
AS
SELECT  SRC.*,
        CURRENT_TIMESTAMP() LDTS,
        'STATIC REFERENCE DATA' RSCR
FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.REGION SRC;

//Validate
SELECT * FROM STG_REGION;

//STAGING AREA SETUP - CUSTOMER & ORDER

//Create two new tables to be used by Snowpipe to
--drip-feed the data as it lands in the stage

//Note that the full payload of JSON data will be loaded into the raw_json
--column.  We'll use the special VARIANT data type for this use case

//Note that we'll also add some columns for metadata like
--load data timestamp (ldts) and file row number

//Create the Customer Stage Table
CREATE OR REPLACE TABLE STG_CUSTOMER
   (RAW_JSON       VARIANT,
    FILENAME        STRING NOT NULL,
    FILE_ROW_SEQ    NUMBER NOT NULL,
    LDTS            STRING NOT NULL,
    RSCR            STRING NOT NULL);

//No records yet entered so query produces no results
SELECT * FROM STG_CUSTOMER;

//Create the ORDER Stage Table
CREATE OR REPLACE TABLE STG_ORDER
    (O_ORDERKEY         NUMBER,
     O_CUSTKEY          NUMBER,
     O_ORDERSTATUS      STRING,
     O_TOTALPRICE       NUMBER,
     O_ORDERDATE        DATE,
     O_ORDERPRIORITY    STRING,
     O_CLERK            STRING,
```

```
    O_SHIPPRIORITY     NUMBER,
    O_COMMENT          STRING,
    FILENAME           STRING NOT NULL,
    FILE_ROW_SEQ       NUMBER NOT NULL,
    LDTS               STRING NOT NULL,
    RSCR               STRING NOT NULL);

 //No records yet entered so query produces no results
 SELECT * FROM STG_ORDER;

//STAGING AREA SETUP - STREAMS

//Create streams on the staging tables in order to easily detect and
--incrementally process the new portion of data
CREATE OR REPLACE STREAM STG_CUSTOMER_STRM ON TABLE
STG_CUSTOMER;
CREATE OR REPLACE STREAM STG_ORDER_STRM ON TABLE STG_ORDER;


//STAGING AREA SETUP - SAMPLE DATA


//We'll be producing sample data by unloading a subset of data from the TPCH
sample dataset then use Snowpipe to load it back
--into the Data Vault tutorial, simulating the streaming feed.

//Every Snowflake account provides access to sample data sets.  You can find
corresponding schemas in SNOWFLAKE_SAMPLE_DATA
--database in your object explorer.  For this tutorial, we are going to use a subset of
objects from TPC-H set, representing
--customer and their order.  We are also going take some reference data about
nations and regions.


//Create two stages, one for each data class type - order and customer data.
//Note: In production environment, these would likely be internal or external stages.
--Alternatively, these feeds could be sourced via a Kafka connector.
CREATE OR REPLACE STAGE CUSTOMER_DATA FILE_FORMAT = (TYPE =
JSON);
CREATE OR REPLACE STAGE ORDER_DATA   FILE_FORMAT = (TYPE = CSV) ;

//Generate and unload sample data.
```

```
//Use object_construct as a quick way to create an object or document from all
columns
--and subsets of rows for the customer data and then offload it into
CUSTOMER_DATA stage.

//ORDER data would be extracted into compressed CSV files.

//There are many additional options in COPY INTO stage construct but we are using
INCLUDE_QUERY_ID
--to make it easier to generate new incremental files as we are going to run these
commands
--over and over again, without a need to deal with file overloading.

//Using relatively small number of records (i.e., LIMIT) as an example in this tutorial

//Customer Data
COPY INTO @CUSTOMER_DATA
FROM
   (SELECT OBJECT_CONSTRUCT(*)
      FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.CUSTOMER LIMIT 20000)
      INCLUDE_QUERY_ID=TRUE;

//Validate
SELECT * FROM @CUSTOMER_DATA;

//Validate
SELECT METADATA$FILENAME,$1 FROM @CUSTOMER_DATA;

//Validate
LIST @CUSTOMER_DATA;

//Order Data
COPY INTO @ORDER_DATA
FROM
   (SELECT *
      FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.ORDERS LIMIT 50000)
      INCLUDE_QUERY_ID=TRUE;

//Validate
//Expected error from the next SQL statement
//SELECT * FROM @ORDER_DATA;

//Validate
SELECT METADATA$FILENAME,$1 FROM @ORDER_DATA;
```

```
//Validate
LIST @ORDER_DATA;


//STAGING AREA SETUP - SAMPLE DATA

//Setup Snowpipe to load data from files in a stage into staging tables.

//Note: The REFRESH functionality is intended for short term use to resolve specific issues
--when Snowpipe fails to load a subset of files and is not intended for regular use.

//We're using the REFRESH in this tutorial to trigger Snowpipe explicitly to scan for new files.
--In production environment, you'll likely enable AUTO_INGEST, connecting it with your
--cloud storage events (like AWS SNS) and process new files automatically,

//Create pipe for order
CREATE OR REPLACE PIPE STG_ORDER_PP
AS
COPY INTO STG_ORDER
   FROM
      (SELECT $1,$2,$3,$4,$5,$6,$7,$8,$9,
       METADATA$FILENAME, METADATA$FILE_ROW_NUMBER,
       CURRENT_TIMESTAMP(), 'ORDER SYSTEM'
       FROM @ORDER_DATA);

ALTER PIPE STG_ORDER_PP   REFRESH;

//Validate - no data
SELECT * FROM STG_ORDER;
SELECT METADATA$FILENAME,$1 FROM @ORDER_DATA;


//Create pipe for customer
CREATE OR REPLACE PIPE STG_CUSTOMER_PP
AS
COPY INTO STG_CUSTOMER
   FROM
      (SELECT $1,
      METADATA$FILENAME, METADATA$FILE_ROW_NUMBER,
      CURRENT_TIMESTAMP(), 'CUSTOMER SYSTEM'
```

```
        FROM @CUSTOMER_DATA);

ALTER PIPE STG_CUSTOMER_PP REFRESH;


//Validate - no data
SELECT * FROM STG_CUSTOMER;
SELECT METADATA$FILENAME,$1 FROM @CUSTOMER_DATA;

//You should be able to see data appearing in the target tables and the stream on
these tables.

//As you would expect, the number of rows in a stream is exactly the same as in the
base table.
--This is because we didn't process or consume the delta of that stream yet.

//Note: it may take a few seconds before the counts are reflected in the table

USE SCHEMA L00_STG;

SELECT 'STG_CUSTOMER', COUNT(1) FROM STG_CUSTOMER
    UNION ALL
SELECT 'STG_ORDER', COUNT(1) FROM STG_ORDER
    UNION ALL
SELECT 'STG_ORDER_STRM', COUNT(1) FROM STG_ORDER_STRM
    UNION ALL
SELECT 'STG_CUSTOMER_STRM', COUNT(1) FROM STG_CUSTOMER_STRM;

//We've established the basics and now, new data will be available via stream.

//Next, we'll see if we can derive some of the business keys for the Data Vault entries
in the model.

//In our example, we will model it as a view on top of the stream that should allow us
to perform
--data parsing (raw_json -> columns) and business_key, hash_diff derivation on the
fly.
--Another thing to notice here is the use of SHA1_binary as hashing function.
--There are many articles on choosing between MD5/SHA1(2)/other hash functions,
so we won't focus on this.

//For this tutorial, we are going to use fairly common SHA1 and its BINARY version
from Snowflake arsenal
--of functions that use less bytes to encode value than STRING.
```

```
//Create Outbound view for Customer Stream
CREATE OR REPLACE VIEW STG_CUSTOMER_STRM_OUTBOUND
AS
SELECT SRC.*,
    raw_json:C_CUSTKEY::NUMBER          C_CUSTKEY,
    raw_json:C_NAME::STRING             C_NAME,
    raw_json:C_ADDRESS::STRING          C_ADDRESS,
    raw_json:C_NATIONKEY::NUMBER        C_NATIONCODE,
    raw_json:C_PHONE::STRING            C_PHONE,
    raw_json:C_ACCTBAL::NUMBER          C_ACCTBAL,
    raw_json:C_MKTSEGMENT::STRING       C_MKTSEGMENT,
    raw_json:C_COMMENT::STRING          C_COMMENT,
-----------------------------------------------------------------
-- derived business key
-----------------------------------------------------------------
    SHA1_BINARY(UPPER(TRIM(C_CUSTKEY)))  SHA1_HUB_CUSTOMER,
    SHA1_BINARY(UPPER(ARRAY_TO_STRING
                    (ARRAY_CONSTRUCT
        (NVL(TRIM(C_NAME)       ,'-1'),
         NVL(TRIM(C_ADDRESS)    ,'-1'),
         NVL(TRIM(C_NATIONCODE) ,'-1'),
         NVL(TRIM(C_PHONE)      ,'-1'),
         NVL(TRIM(C_ACCTBAL)    ,'-1'),
         NVL(TRIM(C_MKTSEGMENT) ,'-1'),
         NVL(TRIM(C_COMMENT)    ,'-1')),
        '^')))
    AS CUSTOMER_HASH_DIFF
    FROM STG_CUSTOMER_STRM SRC;


//Query the view to validate the results.
SELECT * FROM STG_CUSTOMER_STRM_OUTBOUND;


//Create Outbound view for Order Stream
CREATE OR REPLACE VIEW STG_ORDER_STRM_OUTBOUND
AS
SELECT SRC.*,
-----------------------------------------------------------------
-- derived business key
-----------------------------------------------------------------
    SHA1_BINARY(UPPER(TRIM(O_ORDERKEY))) SHA1_HUB_ORDER,
    SHA1_BINARY(UPPER(TRIM(O_CUSTKEY)))  SHA1_HUB_CUSTOMER,
```

```
    SHA1_BINARY(UPPER(ARRAY_TO_STRING(ARRAY_CONSTRUCT
       (NVL(TRIM(O_ORDERKEY)       ,'-1'),
        NVL(TRIM(O_CUSTKEY)        ,'-1')),
       '^')))
    AS SHA1_LNK_CUSTOMER_ORDER,
    SHA1_BINARY(UPPER(ARRAY_TO_STRING(ARRAY_CONSTRUCT
       (NVL(TRIM(O_ORDERSTATUS)    , '-1'),
        NVL(TRIM(O_TOTALPRICE)     , '-1'),
        NVL(TRIM(O_ORDERDATE)      , '-1'),
        NVL(TRIM(O_ORDERPRIORITY)  , '-1'),
        NVL(TRIM(O_CLERK)          , '-1'),
        NVL(TRIM(O_SHIPPRIORITY)   , '-1'),
        NVL(TRIM(O_COMMENT)        , '-1')),
       '^')))
    AS ORDER_HASH_DIFF
    FROM STG_ORDER_STRM SRC;


//Query the view to validate the results.
SELECT * FROM STG_ORDER_STRM_OUTBOUND;

//We've built out staging / inbound pipeline, ready to accommodate streaming data
and
--derived business keys that we are going to use in our Raw Data Vault


//OPTIONAL -- VALIDATE

USE SCHEMA L00_STG;
SELECT * FROM STG_CUSTOMER;
SELECT * FROM STG_NATION;
SELECT * FROM STG_ORDER;
SELECT * FROM STG_REGION;

SELECT * FROM STG_CUSTOMER_STRM_OUTBOUND;
SELECT * FROM STG_ORDER_STRM_OUTBOUND;

SELECT * FROM @CUSTOMER_DATA;
//SELECT * FROM @ORDER_DATA;

SELECT * FROM STG_CUSTOMER_STRM;
SELECT * FROM STG_ORDER_STRM;
```

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

//BUILD RAW DATA VAULT - CUSTOMER AND ORDER HUBS

//Deploy DDL for the HUBs, LINKs, and SATELLITE tables.
USE SCHEMA L10_RDV;

//Create Customer Hub
CREATE OR REPLACE TABLE HUB_CUSTOMER
   (SHA1_HUB_CUSTOMER       BINARY    NOT NULL,
    C_CUSTKEY               NUMBER    NOT NULL,
    LDTS                    TIMESTAMP NOT NULL,
    RSCR                    STRING    NOT NULL,
    CONSTRAINT PK_HUB_CUSTOMER    PRIMARY
KEY(SHA1_HUB_CUSTOMER));


//Create Order Hub
CREATE OR REPLACE TABLE HUB_ORDER
   (SHA1_HUB_ORDER          BINARY    NOT NULL,
    O_ORDERKEY              NUMBER    NOT NULL,
    LDTS                    TIMESTAMP NOT NULL,
    RSCR                    STRING    NOT NULL,
    CONSTRAINT PK_HUB_ORDER       PRIMARY KEY(SHA1_HUB_ORDER));


//BUILD RAW DATA VAULT - CUSTOMER AND ORDER SATELLITES

//Create Customer Satellite
CREATE OR REPLACE TABLE SAT_CUSTOMER
   (SHA1_HUB_CUSTOMER      BINARY    NOT NULL,
    LDTS                    TIMESTAMP NOT NULL,
    C_NAME                  STRING,
    C_ADDRESS               STRING,
    C_PHONE                 STRING,
    C_ACCTBAL               NUMBER,
    C_MKTSEGMENT            STRING,
    C_COMMENT               STRING,
    NATIONCODE              NUMBER,
    HASH_DIFF               BINARY    NOT NULL,
    RSCR                    STRING    NOT NULL,
```

```
    CONSTRAINT PK_SAT_CUSTOMER   PRIMARY
KEY(SHA1_HUB_CUSTOMER, LDTS),
    CONSTRAINT FK_SAT_CUSTOMER   FOREIGN
KEY(SHA1_HUB_CUSTOMER) REFERENCES HUB_CUSTOMER);


//Create ORDER Satellite
CREATE OR REPLACE TABLE SAT_ORDER
   (SHA1_HUB_ORDER        BINARY    NOT NULL,
    LDTS                  TIMESTAMP NOT NULL,
    O_ORDERSTATUS         STRING,
    O_TOTALPRICE          NUMBER,
    O_ORDERDATE           DATE,
    O_ORDERPRIORITY       STRING,
    O_CLERK               STRING,
    O_SHIPPRIORITY        NUMBER,
    O_COMMENT             STRING,
    HASH_DIFF             BINARY    NOT NULL,
    RSCR                  STRING    NOT NULL,
    CONSTRAINT PK_SAT_ORDER    PRIMARY KEY(SHA1_HUB_ORDER, LDTS),
    CONSTRAINT FK_SAT_ORDER    FOREIGN KEY(SHA1_HUB_ORDER)
REFERENCES HUB_ORDER);


//BUILD RAW DATA VAULT - CUSTOMER_ORDER Link

//Create the CUSTOMER_ORDER Link
CREATE OR REPLACE TABLE LNK_CUSTOMER_ORDER
   (SHA1_LNK_CUSTOMER_ORDER BINARY     NOT NULL,
    SHA1_HUB_CUSTOMER       BINARY,
    SHA1_HUB_ORDER          BINARY,
    LDTS                    TIMESTAMP  NOT NULL,
    RSCR                    STRING     NOT NULL,
    CONSTRAINT PK_LNK_CUSTOMER_ORDER  PRIMARY
KEY(SHA1_LNK_CUSTOMER_ORDER),
    CONSTRAINT FK1_LNK_CUSTOMER_ORDER FOREIGN
KEY(SHA1_HUB_CUSTOMER) REFERENCES HUB_CUSTOMER,
    CONSTRAINT FK2_LNK_CUSTOMER_ORDER FOREIGN
KEY(SHA1_HUB_ORDER)    REFERENCES HUB_ORDER);

//BUILD RAW DATA VAULT - REFERENCE TABLES

//Create Region Reference
CREATE OR REPLACE TABLE REF_REGION
```

```
    (REGIONCODE           NUMBER,
     LDTS                 TIMESTAMP,
     RSCR                 STRING NOT NULL,
     R_NAME               STRING,
     R_COMMENT            STRING,
     CONSTRAINT PK_REF_REGION PRIMARY KEY (REGIONCODE))
AS
    SELECT R_REGIONKEY, LDTS, RSCR, R_NAME, R_COMMENT
    FROM L00_STG.STG_REGION;

//Validate
SELECT * FROM REF_REGION;


//Create Nation Reference
CREATE OR REPLACE TABLE REF_NATION
    (NATIONCODE           NUMBER,
     REGIONCODE           NUMBER,
     LDTS                 TIMESTAMP,
     RSCR                 STRING NOT NULL,
     N_NAME               STRING,
     N_COMMENT            STRING,
     CONSTRAINT PK_REF_NATION PRIMARY KEY (NATIONCODE),
     CONSTRAINT FK_REF_REGION FOREIGN KEY (REGIONCODE)
REFERENCES REF_REGION(REGIONCODE))
AS
    SELECT N_NATIONKEY, N_REGIONKEY, LDTS, RSCR, N_NAME,
N_COMMENT
    FROM L00_STG.STG_NATION;

//Validate
SELECT * FROM REF_NATION;



//BUILD RAW DATA VAULT - CREATE TASKS


//Now we have source data waiting in our staging streams and views and we have
target Raw Data Vaults tables.
--We now need to connect the dots.  We are going to create tasks, one per each
stream so whenever
--there is new records coming in a stream, that delta will be incrementally
propagated to all
```

--dependent RDV models in one go.

//To achieve that, we are going to use multi-table insert functionality
--As you can see, tasks can be set up to run on a
--pre-defined frequency (every 1 minute in our example) and use dedicated virtual warehouse
--as a compute power (in our tutorial we are going to use same warehouse for all tasks,
--though this could be as granular as needed).
--Also, before waking up compute resource, tasks are going to check that there is data
--in a corresponding stream to process.

//Create Task for Customer Stream
```
CREATE OR REPLACE TASK CUSTOMER_STRM_TSK
   WAREHOUSE = DV_RDV_WH
   SCHEDULE = '1 minute'
WHEN
   SYSTEM$STREAM_HAS_DATA('L00_STG.STG_CUSTOMER_STRM')
AS INSERT ALL
WHEN (SELECT COUNT(1) FROM HUB_CUSTOMER TGT WHERE
TGT.SHA1_HUB_CUSTOMER = SRC_SHA1_HUB_CUSTOMER) = 0
THEN INTO HUB_CUSTOMER
   (SHA1_HUB_CUSTOMER, C_CUSTKEY, LDTS, RSCR)
   VALUES (SRC_SHA1_HUB_CUSTOMER, SRC_C_CUSTKEY, SRC_LDTS,
SRC_RSCR)
WHEN (SELECT COUNT(1) FROM SAT_CUSTOMER TGT WHERE
TGT.SHA1_HUB_CUSTOMER = SRC_SHA1_HUB_CUSTOMER AND
   TGT.HASH_DIFF = SRC_CUSTOMER_HASH_DIFF) = 0
THEN INTO SAT_CUSTOMER
   (SHA1_HUB_CUSTOMER, LDTS, C_NAME, C_ADDRESS, C_PHONE,
C_ACCTBAL, C_MKTSEGMENT, C_COMMENT,
       NATIONCODE, HASH_DIFF, RSCR)
   VALUES (SRC_SHA1_HUB_CUSTOMER, SRC_LDTS, SRC_C_NAME,
SRC_C_ADDRESS, SRC_C_PHONE, SRC_C_ACCTBAL,
       SRC_C_MKTSEGMENT, SRC_C_COMMENT, SRC_NATIONCODE,
SRC_CUSTOMER_HASH_DIFF, SRC_RSCR)
SELECT
   SHA1_HUB_CUSTOMER   SRC_SHA1_HUB_CUSTOMER,
   C_CUSTKEY          SRC_C_CUSTKEY,
   C_NAME             SRC_C_NAME,
   C_ADDRESS          SRC_C_ADDRESS,
   C_NATIONCODE       SRC_NATIONCODE,
   C_PHONE            SRC_C_PHONE,
```

```
    C_ACCTBAL           SRC_C_ACCTBAL,
    C_MKTSEGMENT        SRC_C_MKTSEGMENT,
    C_COMMENT           SRC_C_COMMENT,
    CUSTOMER_HASH_DIFF  SRC_CUSTOMER_HASH_DIFF,
    LDTS                SRC_LDTS,
    RSCR                SRC_RSCR
  FROM L00_STG.STG_CUSTOMER_STRM_OUTBOUND SRC;

ALTER TASK CUSTOMER_STRM_TSK RESUME;



//Create task for ORDER Stream
CREATE OR REPLACE TASK ORDER_STRM_TSK
    WAREHOUSE = DV_RDV_WH
    SCHEDULE = '1 minute'
WHEN
    SYSTEM$STREAM_HAS_DATA('L00_STG.STG_ORDER_STRM')
AS INSERT ALL
WHEN (SELECT COUNT(1) FROM HUB_ORDER TGT WHERE
TGT.SHA1_HUB_ORDER = SRC_SHA1_HUB_ORDER) = 0
THEN INTO HUB_ORDER
    (SHA1_HUB_ORDER, O_ORDERKEY, LDTS, RSCR)
    VALUES (SRC_SHA1_HUB_ORDER, SRC_O_ORDERKEY, SRC_LDTS,
SRC_RSCR)
WHEN (SELECT COUNT(1) FROM SAT_ORDER TGT WHERE
TGT.SHA1_HUB_ORDER = SRC_SHA1_HUB_ORDER AND TGT.HASH_DIFF =
SRC_ORDER_HASH_DIFF) = 0
THEN INTO SAT_ORDER
    (SHA1_HUB_ORDER, LDTS, O_ORDERSTATUS, O_TOTALPRICE,
O_ORDERDATE, O_ORDERPRIORITY, O_CLERK,
        O_SHIPPRIORITY, O_COMMENT, HASH_DIFF, RSCR)
    VALUES (SRC_SHA1_HUB_ORDER, SRC_LDTS, SRC_O_ORDERSTATUS,
SRC_O_TOTALPRICE, SRC_O_ORDERDATE,
        SRC_O_ORDERPRIORITY, SRC_O_CLERK, SRC_O_SHIPPRIORITY,
SRC_O_COMMENT, SRC_ORDER_HASH_DIFF, SRC_RSCR)
WHEN (SELECT COUNT(1) FROM LNK_CUSTOMER_ORDER TGT
      WHERE TGT.SHA1_LNK_CUSTOMER_ORDER =
SRC_SHA1_LNK_CUSTOMER_ORDER) = 0
THEN INTO LNK_CUSTOMER_ORDER
    (SHA1_LNK_CUSTOMER_ORDER, SHA1_HUB_CUSTOMER,
SHA1_HUB_ORDER, LDTS, RSCR)
```

```
    VALUES (SRC_SHA1_LNK_CUSTOMER_ORDER,
SRC_SHA1_HUB_CUSTOMER, SRC_SHA1_HUB_ORDER, SRC_LDTS,
SRC_RSCR)
SELECT
    SHA1_HUB_ORDER          SRC_SHA1_HUB_ORDER,
    SHA1_LNK_CUSTOMER_ORDER SRC_SHA1_LNK_CUSTOMER_ORDER,
    SHA1_HUB_CUSTOMER       SRC_SHA1_HUB_CUSTOMER,
    O_ORDERKEY              SRC_O_ORDERKEY,
    O_ORDERSTATUS           SRC_O_ORDERSTATUS,
    O_TOTALPRICE            SRC_O_TOTALPRICE,
    O_ORDERDATE             SRC_O_ORDERDATE,
    O_ORDERPRIORITY         SRC_O_ORDERPRIORITY,
    O_CLERK                 SRC_O_CLERK,
    O_SHIPPRIORITY          SRC_O_SHIPPRIORITY,
    O_COMMENT               SRC_O_COMMENT,
    ORDER_HASH_DIFF         SRC_ORDER_HASH_DIFF,
    LDTS                    SRC_LDTS,
    RSCR                    SRC_RSCR
  FROM L00_STG.STG_ORDER_STRM_OUTBOUND SRC;

ALTER TASK ORDER_STRM_TSK    RESUME;




//Once tasks are created and resumed (by default, they are initially suspended),
//let's have a look on the task execution history to see how the process will start.
SELECT *
  FROM TABLE(INFORMATION_SCHEMA.TASK_HISTORY())
  ORDER BY SCHEDULED_TIME DESC;

//Notice how after successful execution, next two tasks run were automatically
SKIPPED as there were nothing in the stream and there is nothing to do.

//We can also check content and stats of the objects involved.
--Please notice that views on streams in our staging area are no longer returning any
rows.
--This is because that delta of changes was consumed by a successfully completed
DML transaction (in our case, embedded in tasks).
--This way you don't need to spend any time implementing incremental detection /
processing logic on the application side.
SELECT 'HUB_CUSTOMER', COUNT(1) FROM HUB_CUSTOMER
UNION ALL
SELECT 'HUB_ORDER', COUNT(1) FROM HUB_ORDER
```

```
UNION ALL
SELECT 'SAT_CUSTOMER', COUNT(1) FROM SAT_CUSTOMER
UNION ALL
SELECT 'SAT_ORDER', COUNT(1) FROM SAT_ORDER
UNION ALL
SELECT 'LNK_CUSTOMER_ORDER', COUNT(1) FROM
LNK_CUSTOMER_ORDER
UNION ALL
SELECT 'L00_STG.STG_CUSTOMER_STRM_OUTBOUND', COUNT(1) FROM
L00_STG.STG_CUSTOMER_STRM_OUTBOUND
UNION ALL
SELECT 'L00_STG.STG_ORDER_STRM_OUTBOUND', COUNT(1) FROM
L00_STG.STG_ORDER_STRM_OUTBOUND;

//We now have data in our Raw Data Vault core structures.  Let's move on and talk
about the concept of virtualization for building your near-real time Data Vault
solution.

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

//VIEWS FOR AGILE REPORTING
//One of the great benefits of having the compute power from Snowflake is that now
it is totally possible to have most of your business vault and information marts in a
Data Vault architecture be built exclusively from views. There is no longer a need to
have the argument that there are "too many joins" or that the response won't be fast
enough. The elasticity of the Snowflake virtual warehouses combined with our
dynamic optimization engine have solved that problem. (For more details, see this
post)

//If you really want to deliver data to the business users and data scientists in NRT, in
our opinion using views is the only option. Once you have the streaming loads built
to feed your Data Vault, the fastest way to make that data visible downstream will be
views. Using views allows you to deliver the data faster by eliminating any latency
that would be incurred by having additional ELT processes between the Data Vault
and the data consumers downstream.
//All the business logic, alignment, and formatting of the data can be in the view
code. That means fewer moving parts to debug, and reduces the storage needed as
well.

//Looking at the diagram above you will see an example of how virtualization could fit
in the architecture. Here, solid lines are representing physical tables and dotted lines
- views. You incrementally ingest data into Raw Data Vault and all downstream
```

transformations are applied as views. From a data consumer perspective when working with a virtualized information mart, the query always shows everything known by your data vault, right up to the point the query was submitted.
//With Snowflake you have the ability to provide as much compute as required, on-demand, without a risk of causing performance impact on any surrounding processes and pay only for what you use. This makes materialization of transformations in layers like Business Data Vault and Information delivery an option rather than a must-have. Instead of "optimizing upfront" you can now make this decision based on the usage pattern characteristics, such as frequency of use, type of queries, latency requirements, readiness of the requirements etc.

//Many modern data engineering automation frameworks are already actively supporting virtualization of logic. Several tools offer a low-code or configuration-like ability to switch between materializing an object as a view or a physical table, automatically generating all required DDL & DML. This could be applied on specific objects, layers or/and be environment specific. So even if you start with a view, you can easily refactor to use a table if user requirements evolve.
//As said before, virtualization is not only a way to improve time-to-value and provide near real time access to the data, given the scalability and workload isolation of Snowflake, virtualization also is a design technique that could make your Data Vault excel: minimizing cost-of-change, accelerating the time-to-delivery and becoming an extremely agile, future proof solution for ever growing business needs.

//BUILD: Business Data Vault
//As a quick example of using views for transformations we just discussed, here is how enrichment of
--customer descriptive data could happen in Business Data Vault, connecting data received from
--source with some reference data.

//Let's create a view that will perform these additional derivations on the fly.
//Assuming non-functional capabilities are satisfying our requirements,
--deploying (and re-deploying a new version) transformations in this way is super easy.

```
USE SCHEMA L20_BDV;
CREATE OR REPLACE VIEW SAT_CUSTOMER_BV
AS
```

```
SELECT RSC.SHA1_HUB_CUSTOMER, RSC.LDTS, RSC.C_NAME,
RSC.C_ADDRESS, RSC.C_PHONE,
   RSC.C_ACCTBAL, RSC.C_MKTSEGMENT, RSC.C_COMMENT,
RSC.NATIONCODE, RSC.RSCR,
   -- derived
   RRN.N_NAME      NATION_NAME,
   RRR.R_NAME      REGION_NAME
FROM L10_RDV.SAT_CUSTOMER   RSC
   LEFT OUTER JOIN L10_RDV.REF_NATION RRN
      ON (RSC.NATIONCODE = RRN.NATIONCODE)
   LEFT OUTER JOIN L10_RDV.REF_REGION RRR
      ON (RRN.REGIONCODE = RRR.REGIONCODE);

//Verify
SELECT * FROM SAT_CUSTOMER_BV;

//Now let's imagine we have a heavier transformation to perform that it would make
more sense
--to materialize it as a table.  It could be more data volume, could be more complex
logic,
--PITs, bridges or even an object that will be used frequently and by many users.
--For this case, let's first build a new business satellite that for illustration purposes
--will be deriving additional classification / tiering for orders based on the conditional
logic.
CREATE OR REPLACE TABLE SAT_ORDER_BV
   (SHA1_HUB_ORDER     BINARY    NOT NULL,
    LDTS               TIMESTAMP NOT NULL,
    O_ORDERSTATUS      STRING,
    O_TOTALPRICE       NUMBER,
    O_ORDERDATE        DATE,
    O_ORDERPRIORITY    STRING,
    O_CLERK            STRING,
    O_SHIPPRIORITY     NUMBER,
    O_COMMENT          STRING,
    HASH_DIFF          BINARY  NOT NULL,
    RSCR               STRING  NOT NULL,
   -- additional attributes
   ORDER_PRIORITY_BUCKET  STRING,
   CONSTRAINT PK_SAT_ORDER PRIMARY KEY(SHA1_HUB_ORDER, LDTS),
   CONSTRAINT FK_SAT_ORDER FOREIGN KEY(SHA1_HUB_ORDER)
REFERENCES L10_RDV.HUB_ORDER)
AS
SELECT SHA1_HUB_ORDER, LDTS, O_ORDERSTATUS, O_TOTALPRICE,
O_ORDERDATE, O_ORDERPRIORITY,
```

```
    O_CLERK, O_SHIPPRIORITY, O_COMMENT, HASH_DIFF, RSCR,
    -- derived additional attributes
    CASE    WHEN O_ORDERPRIORITY IN ('2-HIGH', '1-URGENT') AND
O_TOTALPRICE >= 200000 THEN 'Tier-1'
            WHEN O_ORDERPRIORITY IN ('3-MEDIUM', '2-HIGH', '1-URGENT') AND
O_TOTALPRICE BETWEEN 150000 AND 200000 THEN 'Tier-2'
            ELSE 'Tier-3'
    END ORDER_PRIORITY_BUCKET
FROM L10_RDV.SAT_ORDER;


//VALIDATE
SELECT * FROM SAT_ORDER_BV;

//What we are going to do from processing/orchestration perspective is extending our
--ORDER processing pipeline so that when the task populates a l10_rdv.sat_ORDER
this
--will generate a new stream of changes and these changes are going to be
propagated
--by a dependent task to l20_bdv.sat_ORDER_bv. This is super easy to do as tasks
in
--Snowflake can be not only schedule-based but also start automatically once the
--parent task is completed.
CREATE OR REPLACE STREAM L10_RDV.SAT_ORDER_STRM ON TABLE
L10_RDV.SAT_ORDER;


ALTER TASK L10_RDV.ORDER_STRM_TSK SUSPEND;


CREATE OR REPLACE TASK
L10_RDV.HUB_ORDER_STRM_SAT_ORDER_BV_TSK
  WAREHOUSE = DV_RDV_WH
  AFTER L10_RDV.ORDER_STRM_TSK
AS
INSERT INTO L20_BDV.SAT_ORDER_BV
SELECT
    SHA1_HUB_ORDER, LDTS, O_ORDERSTATUS, O_TOTALPRICE,
O_ORDERDATE, O_ORDERPRIORITY,
    O_CLERK, O_SHIPPRIORITY, O_COMMENT, HASH_DIFF, RSCR,
    -- derived additional attributes
    CASE
        WHEN O_ORDERPRIORITY IN ('2-HIGH', '1-URGENT') AND O_TOTALPRICE
>= 200000 THEN 'Tier-1'
```

```
        WHEN O_ORDERPRIORITY IN ('3-MEDIUM', '2-HIGH', '1-URGENT') AND
O_TOTALPRICE BETWEEN 150000 AND 200000 THEN 'Tier-2'
        ELSE 'Tier-3'
    END ORDER_PRIORITY_BUCKET
FROM SAT_ORDER_STRM;

ALTER TASK L10_RDV.HUB_ORDER_STRM_SAT_ORDER_BV_TSK RESUME;
ALTER TASK L10_RDV.ORDER_STRM_TSK RESUME;


//Now let's go back to our staging area to process another slice of data to test the
task
USE SCHEMA L00_STG;
COPY INTO @ORDER_DATA
FROM
    (SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.ORDERS LIMIT
125000)
--Removed the limit
INCLUDE_QUERY_ID=TRUE;

ALTER PIPE STG_ORDER_PP REFRESH;



//Data is not automatically flowing through all the layers via asynchronous tasks.
--With the results, you can validate:
SELECT 'L00_STG.STG_ORDER', COUNT(1) FROM L00_STG.STG_ORDER
UNION ALL
SELECT 'L00_STG.STG_ORDER_STRM', COUNT(1) FROM
L00_STG.STG_ORDER_STRM
UNION ALL
SELECT 'L10_RDV.SAT_ORDER', COUNT(1) FROM L10_RDV.SAT_ORDER
UNION ALL
SELECT 'L10_RDV.SAT_ORDER_STRM', COUNT(1) FROM
L10_RDV.SAT_ORDER_STRM
UNION ALL
SELECT 'L20_BDV.SAT_ORDER_BV', COUNT(1) FROM
L20_BDV.SAT_ORDER_BV;


SELECT *
  FROM TABLE(INFORMATION_SCHEMA.TASK_HISTORY())
  ORDER BY SCHEDULED_TIME DESC;
```

```
//Now let's go back to our staging area to process another slice of data to test the
task
USE SCHEMA L00_STG;
COPY INTO @ORDER_DATA
FROM
    (SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.ORDERS)
--Removed the limit
INCLUDE_QUERY_ID=TRUE;

ALTER PIPE STG_ORDER_PP REFRESH;



//Data is not automatically flowing through all the layers via asynchronous tasks.
--With the results, you can validate:
SELECT 'L00_STG.STG_ORDER', COUNT(1) FROM L00_STG.STG_ORDER
UNION ALL
SELECT 'L00_STG.STG_ORDER_STRM', COUNT(1) FROM
L00_STG.STG_ORDER_STRM
UNION ALL
SELECT 'L10_RDV.SAT_ORDER', COUNT(1) FROM L10_RDV.SAT_ORDER
UNION ALL
SELECT 'L10_RDV.SAT_ORDER_STRM', COUNT(1) FROM
L10_RDV.SAT_ORDER_STRM
UNION ALL
SELECT 'L20_BDV.SAT_ORDER_BV', COUNT(1) FROM
L20_BDV.SAT_ORDER_BV;


SELECT *
  FROM TABLE(INFORMATION_SCHEMA.TASK_HISTORY())
  ORDER BY SCHEDULED_TIME DESC;



//Now let's go back to our staging area to process another slice of data to test the
task
USE SCHEMA L00_STG;
COPY INTO @ORDER_DATA
FROM
    (SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.ORDERS LIMIT
33333)
```

```
INCLUDE_QUERY_ID=TRUE;

ALTER PIPE STG_ORDER_PP REFRESH;



//Data is not automatically flowing through all the layers via asynchronous tasks.
--With the results, you can validate:
SELECT 'L00_STG.STG_ORDER', COUNT(1) FROM L00_STG.STG_ORDER
UNION ALL
SELECT 'L00_STG.STG_ORDER_STRM', COUNT(1) FROM
L00_STG.STG_ORDER_STRM
UNION ALL
SELECT 'L10_RDV.SAT_ORDER', COUNT(1) FROM L10_RDV.SAT_ORDER
UNION ALL
SELECT 'L10_RDV.SAT_ORDER_STRM', COUNT(1) FROM
L10_RDV.SAT_ORDER_STRM
UNION ALL
SELECT 'L20_BDV.SAT_ORDER_BV', COUNT(1) FROM
L20_BDV.SAT_ORDER_BV;


SELECT *
  FROM TABLE(INFORMATION_SCHEMA.TASK_HISTORY())
  ORDER BY SCHEDULED_TIME DESC;




//Now let's go back to our staging area to process another slice of data to test the
task
USE SCHEMA L00_STG;
COPY INTO @ORDER_DATA
FROM
    (SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.ORDERS)
--Removed the limit
INCLUDE_QUERY_ID=TRUE;

ALTER PIPE STG_ORDER_PP REFRESH;



//Data is not automatically flowing through all the layers via asynchronous tasks.
```

```
--With the results, you can validate:
SELECT 'L00_STG.STG_ORDER', COUNT(1) FROM L00_STG.STG_ORDER
UNION ALL
SELECT 'L00_STG.STG_ORDER_STRM', COUNT(1) FROM
L00_STG.STG_ORDER_STRM
UNION ALL
SELECT 'L10_RDV.SAT_ORDER', COUNT(1) FROM L10_RDV.SAT_ORDER
UNION ALL
SELECT 'L10_RDV.SAT_ORDER_STRM', COUNT(1) FROM
L10_RDV.SAT_ORDER_STRM
UNION ALL
SELECT 'L20_BDV.SAT_ORDER_BV', COUNT(1) FROM
L20_BDV.SAT_ORDER_BV;


SELECT *
  FROM TABLE(INFORMATION_SCHEMA.TASK_HISTORY())
  ORDER BY SCHEDULED_TIME DESC;




--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------


//BUILD: Information Delivery
//When it comes to the Information Delivery layer we are not changing the meaning
of data,
--but we may change format to simplify users to access and work with the
--data products/output interfaces. Different consumers may have different needs and
preferences,
--some would prefer star/snowflake dimensional schemas, some would adhere to
use
--flattened objects or even transform data into JSON/parquet objects.

//First things we would like to add to simplify working with satellites
--is creating views that shows latest version for each key.
-----------------------------------------------------------------
-- RDV curr views
-----------------------------------------------------------------
USE SCHEMA L10_RDV;
```

```
CREATE OR REPLACE VIEW SAT_CUSTOMER_CURR_VW
AS
   SELECT * FROM SAT_CUSTOMER
QUALIFY LEAD(LDTS) OVER (PARTITION BY SHA1_HUB_CUSTOMER ORDER
BY LDTS) IS NULL;


CREATE OR REPLACE VIEW SAT_ORDER_CURR_VW
AS
   SELECT * FROM SAT_ORDER
QUALIFY LEAD(LDTS) OVER (PARTITION BY SHA1_HUB_ORDER ORDER BY
LDTS) IS NULL;
-----------------------------------------------------------------
-- BDV curr views
-----------------------------------------------------------------
USE SCHEMA L20_BDV;

CREATE OR REPLACE VIEW SAT_ORDER_BV_CURR_VW
AS
   SELECT * FROM SAT_ORDER_BV
QUALIFY LEAD(LDTS) OVER (PARTITION BY SHA1_HUB_ORDER ORDER BY
LDTS) IS NULL;


CREATE VIEW SAT_CUSTOMER_BV_CURR_VW
AS
   SELECT * FROM SAT_CUSTOMER_BV
QUALIFY LEAD(LDTS) OVER (PARTITION BY SHA1_HUB_CUSTOMER ORDER
BY LDTS) IS NULL;



//Let's create a simple dimensional structure. Again, we will keep it virtual(as views)
to start with,
--but you already know that depending on access characteristics required any of
these
--could be selectively materialized.
USE SCHEMA L30_INFO;

-- DIM TYPE 1
CREATE OR REPLACE VIEW DIM1_CUSTOMER
AS
SELECT
   HUB.SHA1_HUB_CUSTOMER                    AS DIM_CUSTOMER_KEY,
```

```
    SAT.LDTS                            AS EFFECTIVE_DTS,
    HUB.C_CUSTKEY                       AS CUSTOMER_ID,
    SAT.RSCR                            AS RECORD_SOURCE,
    SAT.*
FROM
    L10_RDV.HUB_CUSTOMER                HUB,
    L20_BDV.SAT_CUSTOMER_BV_CURR_VW     SAT
WHERE HUB.SHA1_HUB_CUSTOMER = SAT.SHA1_HUB_CUSTOMER;

-- DIM TYPE 1
CREATE OR REPLACE VIEW DIM1_ORDER
AS
SELECT

    HUB.SHA1_HUB_ORDER                  AS DIM_ORDER_KEY,
    SAT.LDTS                            AS EFFECTIVE_DTS,
    HUB.O_ORDERKEY                      AS ORDER_ID,
    SAT.RSCR                            AS RECORD_SOURCE,
    SAT.*
FROM
    L10_RDV.HUB_ORDER                   HUB,
    L20_BDV.SAT_ORDER_BV_CURR_VW        SAT
WHERE HUB.SHA1_HUB_ORDER = SAT.SHA1_HUB_ORDER;

-- FACT table
CREATE OR REPLACE VIEW FCT_CUSTOMER_ORDER
AS
SELECT
    LNK.LDTS                            AS EFFECTIVE_DTS,
    LNK.RSCR                            AS RECORD_SOURCE,
    LNK.SHA1_HUB_CUSTOMER               AS DIM_CUSTOMER_KEY,
    LNK.SHA1_HUB_ORDER                  AS DIM_ORDER_KEY
-- this is a factless fact, but here you can add any measures, calculated or derived
FROM L10_RDV.LNK_CUSTOMER_ORDER         LNK;
```

//All good so far?
//Now lets try to query fct_customer_order. You may find that the view does not return any rows. Why?
--If you remember, when we were unloading sample data, we took a subset of random orders and a subset of random customers.
--Thus, it is possible that there won't be any overlap, Therefore doing the inner join with dim1_order will likely result

```
--in all rows being eliminated from the resultset. Thankfully we are using Data Vault
and all we need to do is go and load
--the full customer dataset. Just think about it, there is no need to reprocess any links
or fact tables simply because
--customer/reference feed was incomplete. Lets go and see if we can resolve this.
USE SCHEMA L00_STG;
COPY INTO @CUSTOMER_DATA
FROM
   (SELECT OBJECT_CONSTRUCT(*)
   FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.CUSTOMER)
   -- removed LIMIT
INCLUDE_QUERY_ID=TRUE;

ALTER PIPE STG_CUSTOMER_PP REFRESH;

//All you need to do now is just wait a few seconds whilst our continuous data
pipeline will automatically propagate
--new customer data into Raw Data Vault. Quick check for the records count in
customer dimension now shows
--that there are 1.5Mn records:
USE SCHEMA L30_INFO;
SELECT COUNT(1) FROM DIM1_CUSTOMER;
//Note that if the result is zero, wait a few seconds and rerun the query


//Finally lets wear user's hat and run a query to break down ORDER by nation,
region and ORDER_PRIORITY_BUCKET which
-- are all attributes we derived in Business Data Vault. As we are using Snowsight,
why not quickly creating a chart
--from this result set to better understand the data. For this simply click on the ‘Chart'
section on the bottom pane
--and put attributes/measures as it is shown on the screenshot below.
SELECT DC.NATION_NAME, DC.REGION_NAME,
DO.ORDER_PRIORITY_BUCKET, COUNT(1) CNT_ORDER
FROM    FCT_CUSTOMER_ORDER      FCT,
       DIM1_CUSTOMER          DC,
       DIM1_ORDER             DO
WHERE   FCT.DIM_CUSTOMER_KEY = DC.DIM_CUSTOMER_KEY AND
       FCT.DIM_ORDER_KEY = DO.DIM_ORDER_KEY
GROUP BY 1,2,3;

SELECT * FROM FCT_CUSTOMER_ORDER;
SELECT * FROM DIM1_CUSTOMER;
SELECT * FROM DIM1_ORDER;
```

//Simplicity of engineering, openness, scalable performance, enterprise-grade governance enabled by the core of the Snowflake platform are now allowing teams to focus on what matters most for the business and build truly agile, collaborative data environments. Teams can now connect data from all parts of the landscape, until there are no stones left unturned. They are even tapping into new datasets via live access to the Snowflake Data Marketplace. The Snowflake Data Cloud combined with a Data Vault 2.0 approach is allowing teams to democratize access to all their data assets at any scale. We can now easily derive more and more value through insights and intelligence, day after day, bringing businesses to the next level of being truly data-driven.
//Delivering more usable data faster is no longer an option for today's business environment. Using the Snowflake platform, combined with the Data Vault 2.0 architecture it is now possible to build a world class analytics platform that delivers data for all users in near real-time.


```
//Option to suspend tasks
ALTER TASK L10_RDV.ORDER_STRM_TSK SUSPEND;
ALTER TASK L10.RDV.CUSTOMER_STRM_TSK SUSPEND;
ALTER TASK L10.RDV.HUB_ORDER_STRM_SAT_ORDER_BV_TSK SUSPEND;

//Cleanup
USE WAREHOUSE COMPUTE_WH;
DROP DATABASE DV_TUTORIAL;
DROP WAREHOUSE DV_GENERIC_WH;
DROP WAREHOUSE DV_RDV_WH;
```