DATABASE USER MANUAL

AUTHOR: Omkar Sunkersett, University of Michigan, Ann Arbor

COMPILED FOR UChicago Argonne, LLC

PURPOSE: Summer Internship 2017

Introduction:

This document has information about the NOT-PRINTED database, including useful guidelines about the database design, build, configuration, backup and recovery. It also explains the process to be followed to load the database with new data from various sources across the public Internet and contains some scripts in Python, R and SQL for performing useful actions such as extract-transform-load (ETL) data, generate reports in comma-separated value (CSV) format etc.

Python Scripts:

The scripts use Python version 3.6.1. Please make sure that you have downloaded and installed Python 3.6.1 (https://www.python.org/downloads/release/python-361/).

You would also need to install two modules using the PIP utility in the Windows command prompt. The module names are "requests" and "mysqlclient".

The commands to install the respective modules are as follows –

1. pip install requests
2. pip install mysqlclient

The scripts for fetching and loading new data into the database are attached below with some special instructions for guidance –

Final Scripts.zip

Market Specific Information

1. California Independent System Operator (CAISO)

Before you run the script, you need to uncomment the code in the main() function of the script.

“prog_dir” is the main directory under which the CSV files will be stored

#prog_dir = "C:\\Users\\Omkar Sunkersett\\Downloads\\markets"

The “print()” command initializes the cache file once the “prog_dir” variable has been set.

#print (init_cache(prog_dir))

The below two variables set the start and end dates (in MM-DD-YYYY format) for fetching the CSV files from the server.

#startdatetime = "MM-DD-YYYY"

#enddatetime = "MM-DD-YYYY"

“fetch_files()” is a method that fetches the CSV files from the server and stores them on your local disk.

#each_stream.fetch_files(base_url, '\\'+str(year)+'\\'+str(month).zfill(2)+'\\'+str(day).zfill(2))

The “print()” function writes to the cache file, which contains the absolute paths of the downloaded files. The cache file is used by the program to load the CSV data into the database.

#print (each_stream)

“etl_file_data()” is a method that extracts the data from the CSV files, transforms it into the desired form and loads it into the database.

#print ("\nLoading the new data into the database...\n")

#load_db = Caiso('', '', prog_dir)

#load_db.etl_file_data(prog_dir + "\\cache\\caiso\\caiso-cache.txt")

Important Note: Please make sure that you have the latest backup of the database taken before you run the script in case you need to restore it to an earlier point-in-time.

2. Midcontinent Independent System Operator (MISO)

Before you run the script, you need to uncomment the code in the main() function of the script.

“prog_dir” is the main directory under which the CSV files will be stored

#prog_dir = "C:\\Users\\Omkar Sunkersett\\Downloads\\markets"

The below two variables set the start and end dates (in MM-DD-YYYY format) for fetching the CSV files from the server.

#startdatetime = "MM-DD-YYYY"

#enddatetime = "MM-DD-YYYY"

“fetch_files()” is a method that fetches the CSV files from the server and stores them on your local disk.

asm_da_off.fetch_files("da", startdatetime, enddatetime)

asm_rt_off.fetch_files("rt", startdatetime, enddatetime)

The “print()” function writes to the cache file, which contains the absolute paths of the downloaded files. The cache file is used by the program to load the CSV data into the database.

#print (asm_da_off)

#print (asm_rt_off)

“etl_file_data()” is a method that extracts the data from the CSV files, transforms it into the desired form and loads it into the database.

#print ("\nLoading the new data into the database...\n")

#asm_da_off.etl_file_data(prog_dir + "\\cache\\miso\\asm-da-co.txt")

#asm_rt_off.etl_file_data(prog_dir + "\\cache\\miso\\asm-rt-co.txt")

Important Note: Please make sure that you have the latest backup of the database taken before you run the script in case you need to restore it to an earlier point-in-time.

3. Independent System Operator New England (ISO-NE)

Before you run the script, you need to uncomment the code in the main() function of the script.

“prog_dir” is the main directory under which the CSV files will be stored

#prog_dir = "C:\\Users\\Omkar Sunkersett\\Downloads\\markets"

The below two variables set the start and end dates (in MM-DD-YYYY format) for fetching the CSV files from the server.

#startdatetime = "MM-DD-YYYY"

#enddatetime = "MM-DD-YYYY"

“fetch_files()” is a method that fetches the CSV files from the server and stores them on your local disk.

#reg_offers.fetch_files(file_dt)

The “print()” function writes to the cache file, which contains the absolute paths of the downloaded files. The cache file is used by the program to load the CSV data into the database.

```
#print (reg_offers)
```

“etl_file_data()” is a method that extracts the data from the CSV files, transforms it into the desired form and loads it into the database.

```
#print ("\nLoading the new data into the database...\n")

#reg_offers.etl_file_data(prog_dir + "\\cache\\iso-ne\\isone-cache.txt")
```

Important Note: Please make sure that you have the latest backup of the database taken before you run the script in case you need to restore it to an earlier point-in-time.

4. Southwest Power Pool (SPP)

Before you run the script, you need to uncomment the code in the main() function of the script.

“prog_dir” is the main directory under which the CSV files will be stored

```
#prog_dir = "C:\\Users\\Omkar Sunkersett\\Downloads\\markets"
```

The below two variables set the start and end dates (in MM-DD-YYYY format) for fetching the CSV files from the server.

```
#startdatetime = "MM-DD-YYYY"

#enddatetime = "MM-DD-YYYY"
```

“fetch_files()” is a method that fetches the CSV files from the server and stores them on your local disk.

#histoff_or = SPP("pubftp.spp.org", "/Markets/HistoricalOffers/", startdatetime, enddatetime, prog_dir)

#histoff_or.fetch_files("/Markets/HistoricalOffers", "")

The “print()” function writes to the cache file, which contains the absolute paths of the downloaded files. The cache file is used by the program to load the CSV data into the database.

#print(histoff_or)

“etl_file_data()” is a method that extracts the data from the CSV files, transforms it into the desired form and loads it into the database.

#print ("\nLoading the new data into the database...\n")

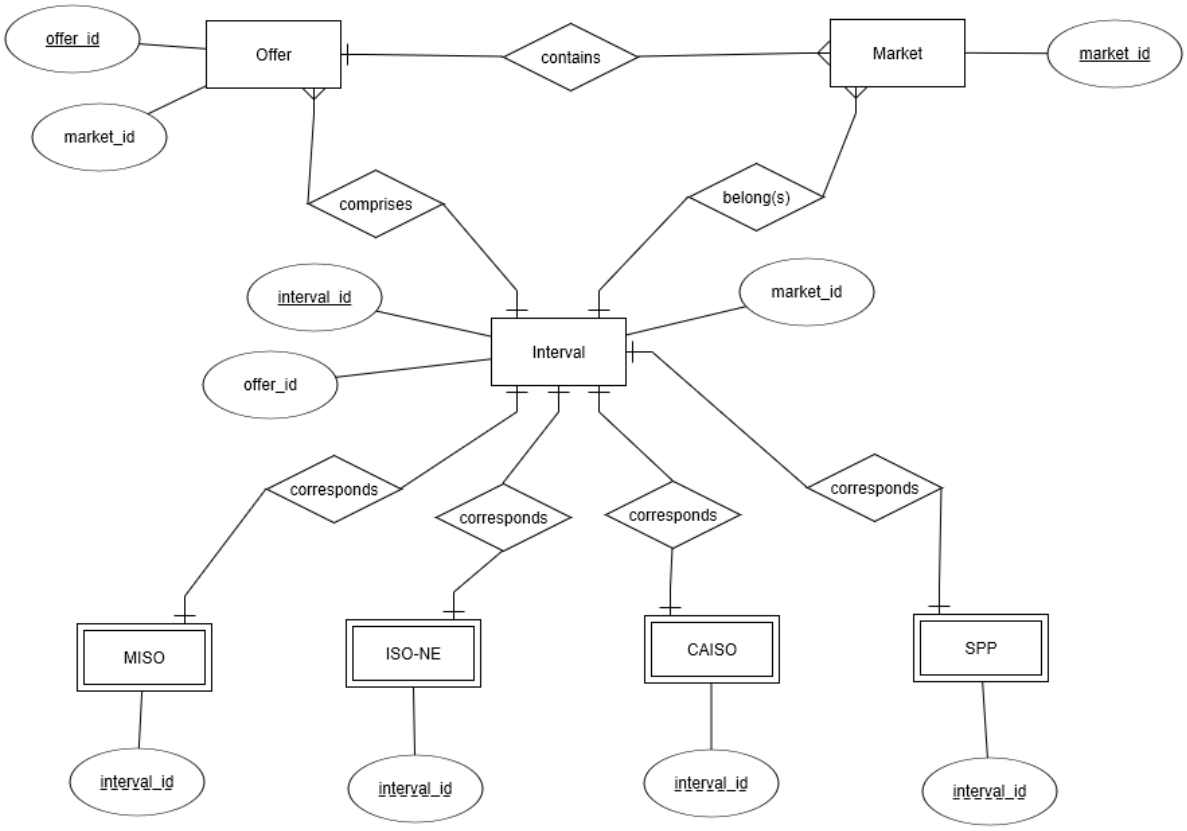
#etl_file_data(prog_dir + "\\cache\\spp\\Markets\HistoricalOffers.txt")

Important Note: Please make sure that you have the latest backup of the database taken before you run the script in case you need to restore it to an earlier point-in-time.

Additional Note: The scripts have been designed in a manner that data does not get duplicated in the database. Each script will indicate the feasible start and end dates to the user whenever the user enters a date for which data already exists in the database.

Database Design *

The Entity Relationship Diagram (ERD) for the database is as follows –



The rectangles in the diagram represent entity sets and the oval-shaped circles represent their respective attributes. The primary and foreign keys have been only shown since we have limited space in the diagram. A primary key is an attribute that can be used to uniquely identify a given entity of an entity set. Similarly, a foreign key is an attribute that can be used to uniquely identify a given entity of another entity set that does not have its own primary key.

A strong entity set is an entity set that can have a primary key, whereas a weak entity set is an entity set that cannot have a primary key. A single-bordered rectangle represents a strong entity set, whereas a double-bordered rectangle represents a weak entity set. A weak entity set must borrow the primary key of a strong entity set in the form of a foreign key. This ensures that every entity within the weak entity set can be uniquely identified with the help of that foreign key. Primary key attributes have a bold or dashed line below their respective names. The bold line represents a primary key attribute that belongs to the same entity set, whereas the dashed line represents a primary key attribute that has been borrowed from a strong entity set.

Further, a foreign key may also be used to enforce an integrity constraint between two entity sets. A pair of entity sets can be linked together to form a relationship set that is denoted by a diamond-shaped square. The relationship set describes the relationship between the two-participating entity sets in the form of the mapping cardinalities between both the entity sets. The different types of mapping cardinalities can be one-to-one, one-to-many, many-to-one and many-to-many. For instance, a one-to-one mapping cardinality indicates that each entity from a given set is linked to only one entity from the other set participating in the relationship. A single participation is denoted by a perpendicular bar, whereas multiple participation is denoted by three diverging bars as shown in the above figure. It must be noted that the depiction is opposite in nature. For example, each market contains multiple offers, so the mapping cardinality is one-to-many. Therefore, there is a perpendicular bar near the "offer" entity set and there are three diverging bars near the "market" entity set.

***Source:** Silberschatz, A., Korth, H. F., & Sudarshan, S. (2005). Database system concepts. Boston: McGraw-Hill.

The database structure for the above diagram is as follows:

The database is known by the schema "NOT-PRINTED", which is the name of the database. This database has two users: root and dbadmin. The "root" user is used to shut down and startup the MySQL server for maintenance work such as server upgrades, whereas the "dbadmin" user is used for database administration activities such as data import, backup and recovery. The database has been configured to use the InnoDB storage engine which supports the relational data model (rows and columns) and complies with the ACID (atomicity, consistency, isolation, durability) properties of modern relational database management systems (RDBMS).

The credentials of the users of the database are as follows –

Username	**Password**
root	NOT-PRINTED
dbadmin	NOT-PRINTED

The configuration file of the database is called "my.ini" and is located in the path "C:\ProgramData\MySQL\MySQL Server 5.7". The important initialization parameters of the MySQL server are as follows –

Parameter	Value	Significance
port	NOT-PRINTED	This represents the Windows port number that the client should use to connect to the MySQL server.
default-character-set	utf8	This represents the default character set supported by the MySQL server for encoding and decoding data sent between the server and the client.
datadir	C:/ProgramData/MySQL/MySQL Server 5.7\Data	This represents the path in which the database files (i.e. data files) are stored by the MySQL server.
default-storage-engine	INNODB	This represents the default storage engine of the database.
sql-mode	STRICT_TRANS_TABLES, NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION	This represents the current SQL Mode of the MySQL server.
log-output	FILE	This represents the manner in which the MySQL server logs are stored. They are stored as files.
general-log	1	This represents the status of general query logging. The “1” indicates that general query logging does occur.
general_log_file	NOT-PRINTED-DATA.log	This represents the name of the general query log file. It is important to rename this file and compress it each time after

		you perform a large data import to avoid disk space issues.
slow-query-log	1	This represents the status of slow query logging. The “1” indicates that slow query logging does occur.
slow_query_log_file	NOT-PRINTED-DATA-slow.log	This represents the name of the slow query log file.
long_query_time	10	This represents the minimum number of seconds that the MySQL server will wait before considering a query to be a slow-running one.
log-bin	NOT-PRINTED-DATA-bin	This represents the name of the binary log file that stores the uncommitted data during transactions.
log-error	NOT-PRINTED-DATA.err	This represents the name of the error log file that contains a historical record of all errors that occur.
server-id	1	This indicates whether the MySQL server can accept connections from “slave” servers for data replication. A “1” value indicates that this server can accept such connections.
secure-file-priv	C:\ProgramData\MySQL\MySQL Server 5.7\Uploads	This indicates the default location of the dump

		destination for performing data export and import operations using the command-line client.
max_connections	151	This represents the maximum number of simultaneous connections that the MySQL server can support at any given point of time.
query_cache_size	0	This indicates the default cache size for each query. A “0” value indicates that a default query cache size is not configured.
table_open_cache	2000	This indicates the maximum number of open tables for all threads of the MySQL server.
tmp_table_size	2048M	This indicates the maximum size of an in-memory temporary table used by a query.
thread_cache_size	10	This represents the maximum number of threads that the MySQL server keeps in the cache for client-connection reuse.
innodb_flush_log_at_trx_commit	1	This indicates whether the InnoDB storage engine flushes the binary logs to disk each time a commit happens. A “1” value indicates that this behavior does happen.

innodb_log_buffer_size	512M	This indicates the maximum size of the binary log buffer, which is flushed to disk by default when it becomes full.
innodb_buffer_pool_size	8G	This indicates the maximum size of the buffer pool, which caches indexes and row data for queries.
innodb_log_file_size	2048M	This indicates the size of each binary log file in a log group.
innodb_thread_concurrency	17	This indicates the maximum number of concurrent threads allowed inside the InnoDB kernel at any given point of time.
innodb_autoextend_increment	64	This indicates the increment size (in MB) for extending the size of an auto-extend InnoDB system tablespace file when it becomes full.
innodb_buffer_pool_instances	16	This indicates the number of regions that the InnoDB buffer pool is divided into.
innodb_concurrency_tickets	5000	This indicates the number of threads that can enter InnoDB concurrently
innodb_old_blocks_time	1000	This specifies how long in milliseconds (ms) a block inserted into the old sub-list must stay there after its first

		access before it can be moved to the new sub-list.
innodb_open_files	300	This specifies the maximum number of .ibd files that MySQL can keep open at one time.
innodb_stats_on_metadata	0	This specifies whether InnoDB updates statistics during metadata statements. A “0” value indicates a “no”.
innodb_file_per_table	1	This specifies whether InnoDB stores the data and indexes for each newly created table in a separate “.ibd” file rather than in the system tablespace. A “1” value indicates a “yes”.
innodb_checksum_algorithm	0	This specifies the default checksum algorithm used by InnoDB. A “0” value represents CRC32 (cyclic code redundancy 32-bit).
back_log	80	This represents the number of outstanding connection requests MySQL can have.
flush_time	0	This indicates the number of seconds after every time the log buffer writes uncommitted data to disk. A “0” value signifies that this parameter is disabled.
join_buffer_size	2048M	This specifies the minimum size of the buffer that is used for

		plain index scans, range index scans, and joins that do not use indexes and thus perform full table scans.
max_allowed_packet	128M	This represents the maximum size of each packet of data during transmission using the TCP/IP protocol.
max_connect_errors	100	This represents the maximum number of unsuccessful successive connection requests that the MySQL server allows from a given host before it blocks that host from performing further connections.
open_files_limit	4161	This represents the maximum number of file descriptors available to the MySQL server.
query_cache_type	0	This specifies the query cache type. A “0” value indicates that there is no particular cache type for a query.
sort_buffer_size	2048M	This specifies the minimum sort buffer size for ORDER BY and/or GROUP BY operations in queries.
table_definition_cache	1400	This indicates the maximum number of table definitions (from .frm files) that can be stored in the definition cache.

binlog_row_event_max_size	8K	This specifies the maximum size of a row-based binary log event, in bytes. It must be a multiple of 256.
innodb_strict_mode	Not Applicable	If this parameter is included in the configuration file, then it indicates that strict mode is set to “ON” for the InnoDB storage engine.
net_read_timeout	90	This specifies the maximum number of seconds to wait for more data from a connection before aborting the read.

The data dictionary of the database contains the metadata about the various tables, including their indexes and constraints. The data dictionary has been attached below –

Data_Dictionary.xlsx

The commands that have been used to create the data structures are as follows –

1. **For creating the tablespaces for storing the respective tables and their data files –**

```
CREATE TABLESPACE TAB_MKT_META ADD DATAFILE 'tabmktmeta.ibd' FILE_BLOCK_SIZE = 8K ENGINE = InnoDB;

CREATE TABLESPACE TAB_OFF_BASE ADD DATAFILE 'taboffbase.ibd' FILE_BLOCK_SIZE = 8K ENGINE = InnoDB;

CREATE TABLESPACE TAB_INTV_META ADD DATAFILE 'tabintvmeta.ibd' FILE_BLOCK_SIZE = 8K ENGINE = InnoDB;

CREATE TABLESPACE TAB_CAISO_RES ADD DATAFILE 'tabcaisores.ibd' FILE_BLOCK_SIZE = 8K ENGINE = InnoDB;
```

```
CREATE TABLESPACE TAB_MISO_RES ADD DATAFILE 'tabmisores.ibd' FILE_BLOCK_SIZE = 8K ENGINE = InnoDB;

CREATE TABLESPACE TAB_ISONE_RES ADD DATAFILE 'tabisoneres.ibd' FILE_BLOCK_SIZE = 8K ENGINE = InnoDB;

CREATE TABLESPACE TAB_SPP_RES ADD DATAFILE 'tabsppres.ibd' FILE_BLOCK_SIZE = 8K ENGINE = InnoDB;
```

2. **For creating and connecting to the schema of the database –**

```
CREATE DATABASE NOT-PRINTED;

CONNECT NOT-PRINTED;
```

3. **For creating the respective tables along with their indexes, constraints and views, and initializing the market metadata –**

TABLE MARKET_META:

```
CREATE TABLE IF NOT EXISTS MARKET_META

(

market_id TINYINT UNSIGNED AUTO_INCREMENT,

market_name VARCHAR(256) NOT NULL,

PRIMARY KEY(market_id)

)

ENGINE = InnoDB

AUTO_INCREMENT = 1

KEY_BLOCK_SIZE = 8
```

STATS_AUTO_RECALC = DEFAULT

STATS_PERSISTENT = DEFAULT

STATS_SAMPLE_PAGES = 20

TABLESPACE TAB_MKT_META STORAGE DISK

;

TABLE OFFER_BASE:

CREATE TABLE IF NOT EXISTS OFFER_BASE

(

offer_id INT UNSIGNED AUTO_INCREMENT,

identifier_1 VARCHAR(256) NOT NULL,

identifier_2 VARCHAR(256) NOT NULL,

region_name VARCHAR(256),

market_id TINYINT UNSIGNED,

PRIMARY KEY (offer_id),

UNIQUE KEY IDX_OFFER_BASE_ID1_ID2 (identifier_1, identifier_2),

INDEX IDX_OFFER_BASE_MARKET_ID (market_id),

CONSTRAINT FK_OFFER_BASE_MARKET_ID FOREIGN KEY (market_id) references MARKET_META(market_id) ON UPDATE CASCADE ON DELETE CASCADE

)

ENGINE = InnoDB

AUTO_INCREMENT = 1

KEY_BLOCK_SIZE = 8

STATS_AUTO_RECALC = DEFAULT

STATS_PERSISTENT = DEFAULT

STATS_SAMPLE_PAGES = 20

TABLESPACE TAB_OFF_BASE STORAGE DISK

;

TABLE INTERVAL_META:

CREATE TABLE IF NOT EXISTS INTERVAL_META

(

interval_id VARCHAR(256),

offer_id INT UNSIGNED NOT NULL,

market_id TINYINT UNSIGNED NOT NULL,

mkt_run_id VARCHAR(256) NOT NULL,

interval_dt DATE NOT NULL,

interval_start DATETIME,

interval_end DATETIME,

opr_hour TINYINT NOT NULL,

opr_interval TINYINT NOT NULL,

PRIMARY KEY (interval_id),

INDEX IDX_INTERVAL_META_OFFER_ID (offer_id),

CONSTRAINT FK_INTERVAL_META_OFFER_ID FOREIGN KEY (offer_id) REFERENCES OFFER_BASE(offer_id) ON UPDATE CASCADE ON DELETE CASCADE,

INDEX IDX_INTERVAL_META_MARKET_ID (market_id),

```
CONSTRAINT FK_INTERVAL_META_MARKET_ID FOREIGN KEY (market_id) REFERENCES MARKET_META(market_id) ON UPDATE CASCADE ON DELETE CASCADE,

CHECK (mkt_run_id IN ('DAM', 'RTM', 'HASP')),

CHECK (opr_hour > 0 AND opr_hour < 25),

CHECK (opr_interval >= 0 AND opr_interval <= 4)

)

ENGINE = InnoDB

KEY_BLOCK_SIZE = 8

STATS_AUTO_RECALC = DEFAULT

STATS_PERSISTENT = DEFAULT

STATS_SAMPLE_PAGES = 20

TABLESPACE TAB_INTV_META STORAGE DISK

;
```

Table CAISO_RESULTS:

```
CREATE TABLE IF NOT EXISTS CAISO_RESULTS

(

interval_id VARCHAR(256) NOT NULL,

nsreq_max FLOAT(10, 2),

nsreq_min FLOAT(10, 2),

rdreq_max FLOAT(10, 2),

rdreq_min FLOAT(10, 2),

rmdreq_max FLOAT(10, 2),
```

rmdreq_min FLOAT(10, 2),

rureq_max FLOAT(10, 2),

rureq_min FLOAT(10, 2),

rmureq_max FLOAT(10, 2),

rmureq_min FLOAT(10, 2),

spreq_max FLOAT(10, 2),

spreq_min FLOAT(10, 2),

nsproc_cap FLOAT(10, 2),

nsself_cap FLOAT(10, 2),

nscost_line FLOAT(10, 2),

nsclr_price FLOAT(10, 2),

nstot_cap FLOAT(10, 2),

rdproc_cap FLOAT(10, 2),

rdself_cap FLOAT(10, 2),

rdcost_line FLOAT(10, 2),

rdclr_price FLOAT(10, 2),

rdtot_cap FLOAT(10, 2),

rmdproc_cap FLOAT(10, 2),

rmdself_cap FLOAT(10, 2),

rmdcost_line FLOAT(10, 2),

rmdclr_price FLOAT(10, 2),

rmdtot_cap FLOAT(10, 2),

ruproc_cap FLOAT(10, 2),

```
ruself_cap FLOAT(10, 2),
rucost_line FLOAT(10, 2),
ruclr_price FLOAT(10, 2),
rutot_cap FLOAT(10, 2),
rmuproc_cap FLOAT(10, 2),
rmuself_cap FLOAT(10, 2),
rmucost_line FLOAT(10, 2),
rmuclr_price FLOAT(10, 2),
rmutot_cap FLOAT(10, 2),
spproc_cap FLOAT(10, 2),
spself_cap FLOAT(10, 2),
spcost_line FLOAT(10, 2),
spclr_price FLOAT(10, 2),
sptot_cap FLOAT(10, 2),
INDEX IDX_CAISO_RESULTS_INTERVAL_ID (interval_id),
CONSTRAINT FK_CAISO_RESULTS_INTERVAL_ID FOREIGN KEY (interval_id) REFERENCES INTERVAL_META(interval_id) ON UPDATE CASCADE ON DELETE CASCADE
)
ENGINE = InnoDB
KEY_BLOCK_SIZE = 8
STATS_AUTO_RECALC = DEFAULT
STATS_PERSISTENT = DEFAULT
STATS_SAMPLE_PAGES = 20
```

TABLESPACE TAB_CAISO_RES STORAGE DISK

;

TABLE MISO_RESULTS:

CREATE TABLE IF NOT EXISTS MISO_RESULTS

(

interval_id VARCHAR(256) NOT NULL,

reg_max FLOAT(10, 2),

reg_min FLOAT(10, 2),

regoff_price FLOAT(10, 2),

regself_limit FLOAT(10, 2),

spinoff_price FLOAT(10, 2),

spinself_limit FLOAT(10, 2),

onsupp_price FLOAT(10, 2),

onsuppself_limit FLOAT(10, 2),

offsupp_price FLOAT(10, 2),

offsuppself_limit FLOAT(10, 2),

regavg_mcp FLOAT(10, 2),

regavg_cap FLOAT(10, 2),

spinavg_mcp FLOAT(10, 2),

spinavg_cap FLOAT(10, 2),

suppavg_mcp FLOAT(10, 2),

suppavg_cap FLOAT(10, 2),

```
INDEX IDX_MISO_RESULTS_INTERVAL_ID (interval_id),

CONSTRAINT FK_MISO_RESULTS_INTERVAL_ID FOREIGN KEY (interval_id) REFERENCES INTERVAL_META(interval_id) ON UPDATE CASCADE ON DELETE CASCADE

)

ENGINE = InnoDB

KEY_BLOCK_SIZE = 8

STATS_AUTO_RECALC = DEFAULT

STATS_PERSISTENT = DEFAULT

STATS_SAMPLE_PAGES = 20

TABLESPACE TAB_MISO_RES STORAGE DISK

;
```

TABLE ISONE_RESULTS:

```
CREATE TABLE IF NOT EXISTS ISONE_RESULTS

(

interval_id VARCHAR(256) NOT NULL,

reglimit_low FLOAT(10, 2),

reglimit_high FLOAT(10, 2),

reg_status VARCHAR(256),

autoresp_rate FLOAT(10, 2),

regoff_price FLOAT(10, 2),

regserv_price FLOAT(10, 2),

regcap_price FLOAT(10, 2),
```

```
regito_cost FLOAT(10, 2),

INDEX IDX_ISONE_RESULTS_INTERVAL_ID (interval_id),

CONSTRAINT FK_ISONE_RESULTS_INTERVAL_ID FOREIGN KEY (interval_id) REFERENCES INTERVAL_META(interval_id) ON UPDATE CASCADE ON DELETE CASCADE,

CHECK (reg_status IN ('AVAILABLE', 'UNAVAILABLE'))

)

ENGINE = InnoDB

KEY_BLOCK_SIZE = 8

STATS_AUTO_RECALC = DEFAULT

STATS_PERSISTENT = DEFAULT

STATS_SAMPLE_PAGES = 20

TABLESPACE TAB_ISONE_RES STORAGE DISK

;
```

TABLE SPP_RESULTS:

```
CREATE TABLE IF NOT EXISTS SPP_RESULTS

(

interval_id VARCHAR(256) NOT NULL,

coreg_down FLOAT(10, 2),

coreg_up FLOAT(10, 2),

mfreg_down FLOAT(10, 2),

mfreg_up FLOAT(10, 2),

moreg_down FLOAT(10, 2),
```

moreg_up FLOAT(10, 2),

spin_price FLOAT(10, 2),

supp_price FLOAT(10, 2),

INDEX IDX_SPP_RESULTS_INTERVAL_ID (interval_id),

CONSTRAINT FK_SPP_RESULTS_INTERVAL_ID FOREIGN KEY (interval_id) REFERENCES INTERVAL_META(interval_id) ON UPDATE CASCADE ON DELETE CASCADE

)

ENGINE = InnoDB

KEY_BLOCK_SIZE = 8

STATS_AUTO_RECALC = DEFAULT

STATS_PERSISTENT = DEFAULT

STATS_SAMPLE_PAGES = 20

TABLESPACE TAB_SPP_RES STORAGE DISK

;

INDEX IDX_INTERVAL_META_IVDT_MRID:

CREATE INDEX IDX_INTERVAL_META_IVDT_MRID ON INTERVAL_META (interval_dt, mkt_run_id) USING BTREE;

INDEX IDX_INTERVAL_META_IVDT_MRID:

CREATE INDEX IDX_INTERVAL_META_MKID_MRID_IVDT ON INTERVAL_META (market_id, mkt_run_id, interval_dt) USING BTREE;

VIEW OFFER_META:

```
CREATE OR REPLACE ALGORITHM=MERGE VIEW OFFER_META AS

SELECT * FROM OFFER_BASE WHERE REGION_NAME NOT IN

('AS_CAISO', 'AS_CAISO_EXP', 'AS_SP26', 'AS_SP26_EXP', 'AS_SP26_EXP_P', 'AS_SP26_P',
'AS_CAISO_NP26_P', 'AS_NP26_EXP', 'NP26 EXP Part', 'AS_NP26_EXP_P', 'AS_CAISO_SP26_P',
'AS_NP15', 'AS_SP15');
```

INITIALIZING THE MARKET_META TABLE WITH THE MARKET NAMES:

```
INSERT INTO MARKET_META (MARKET_NAME) VALUES ('CAISO');

INSERT INTO MARKET_META (MARKET_NAME) VALUES ('MISO');

INSERT INTO MARKET_META (MARKET_NAME) VALUES ('ISO-NE');

INSERT INTO MARKET_META (MARKET_NAME) VALUES ('SPP');

COMMIT;
```

STEPS TO SETUP THE DATABASE SERVER

1. Download MySQL Installer from https://dev.mysql.com/downloads/windows/installer/ and perform a "Full" installation of MySQL, installing all prerequisites and ignoring all warnings that pop up during the installation. Set the "root" user's password as "NOT-PRINTED" upon installation and create a new user called "dbadmin" having the password "NOT-PRINTED" having complete rights.
2. Start the MySQL command-line client and run the following commands sequentially —
    a. `CREATE TABLESPACE TAB_MKT_META ADD DATAFILE 'tabmktmeta.ibd' FILE_BLOCK_SIZE = 8K ENGINE = InnoDB;`
    b. `CREATE TABLESPACE TAB_OFF_BASE ADD DATAFILE 'taboffbase.ibd' FILE_BLOCK_SIZE = 8K ENGINE = InnoDB;`
    c. `CREATE TABLESPACE TAB_INTV_META ADD DATAFILE 'tabintvmeta.ibd' FILE_BLOCK_SIZE = 8K ENGINE = InnoDB;`

d. CREATE TABLESPACE TAB_CAISO_RES ADD DATAFILE 'tabcaisores.ibd' FILE_BLOCK_SIZE = 8K ENGINE = InnoDB;
e. CREATE TABLESPACE TAB_MISO_RES ADD DATAFILE 'tabmisores.ibd' FILE_BLOCK_SIZE = 8K ENGINE = InnoDB;
f. CREATE TABLESPACE TAB_ISONE_RES ADD DATAFILE 'tabisoneres.ibd' FILE_BLOCK_SIZE = 8K ENGINE = InnoDB;
g. CREATE TABLESPACE TAB_SPP_RES ADD DATAFILE 'tabsppres.ibd' FILE_BLOCK_SIZE = 8K ENGINE = InnoDB;
h. CREATE DATABASE NOT-PRINTED;

3. Restart MySQL using the new configuration file (my.ini).
    a. Stop MySQL Server using the "MySQL Notifier" utility.
    b. Copy the below file (my.ini) to the location "C:\ProgramData\MySQL\MySQL Server 5.7" overwriting the existing file.
    c. Start MySQL Server using the "MySQL Notifier" utility.
4. Start MySQL Workbench to perform the data import.
    a. Click the "Data Import/Restore" menu under the "Management" section of the navigator pane.
    b. Click the "Import from Disk" tab and select the radio button that corresponds to the "Import from Dump Project Folder" option and select the correct dump folder.
    c. Make sure that you import both the dump structure and data by selecting the "Dump Structure and Data" option from the drop-down list.
    d. Start the import operation by clicking on the "Start Import" button.
    e. Monitor the progress of the import operation by clicking on the "Import Progress" tab and wait till it finishes. It takes around 2-4 hours for the import to complete.
5. Run the following command to optimize the performance of the database on your machine. This command defragments your data files, analyzes all tables and rebuilds their indexes to improve performance.

    OPTIMIZE TABLE CAISO_RESULTS, MISO_RESULTS, ISONE_RESULTS, SPP_RESULTS, INTERVAL_META, OFFER_BASE, MARKET_META;

    Note: It is recommended to use this command every time you make a significant/large number of inserts/updates/deletes to the database. For example, if

you insert 1,000+ rows into the "interval_meta", you should run the command "optimize table interval_meta;" to optimize the performance of this table.

GUIDELINES TO SETUP THE MYSQL CLIENT INTERFACES

1. **Using R Programming Language in R Studio –**

   R can be downloaded from the URL: https://cran.r-project.org

   R Studio can be downloaded from the URL: https://www.rstudio.com/products/rstudio/download/

   R uses the "RMySQL" package to allow programmers to connect to a MySQL database server using R Studio. You can install this package in R Studio by running the following command:

   install.packages("RMySQL")

   To load the package in R Studio, use the following command:

   library(RMySQL)

   To establish a connection to the MySQL database server, use the following command:

   mydb <- dbConnect(MySQL(), user='dbadmin', password='NOT-PRINTED', dbname='not-printed', host='NOT-PRINTED', port=NOT-PRINTED)

   To extract a full report for a particular time period, use the following commands:

   report_start = 'YYYY-MM-DD'

   report_end = 'YYYY-MM-DD'

   Query for CAISO –

   caiso_qry = paste("SELECT w.market_id, w.market_name, x.offer_id, x.identifier_1, x.identifier_2, x.region_name, y.interval_id, y.mkt_run_id, y.interval_dt, y.interval_start, y.interval_end,

y.opr_hour, y.opr_interval, z.nsreq_max, z.nsreq_min, z.rdreq_max, z.rdreq_min, z.rmdreq_max, z.rmdreq_min, z.rureq_max, z.rureq_min, z.rmureq_max, z.rmureq_min, z.spreq_max, z.spreq_min, z.nsproc_cap, z.nsself_cap, z.nscost_line, z.nsclr_price, z.nstot_cap, z.rdproc_cap, z.rdself_cap, z.rdcost_line, z.rdclr_price, z.rdtot_cap, z.rmdproc_cap, z.rmdself_cap, z.rmdcost_line, z.rmdclr_price, z.rmdtot_cap, z.ruproc_cap, z.ruself_cap, z.rucost_line, z.ruclr_price, z.rutot_cap, z.rmuproc_cap, z.rmuself_cap, z.rmucost_line, z.rmuclr_price, z.rmutot_cap, z.spproc_cap, z.spself_cap, z.spcost_line, z.spclr_price, z.sptot_cap FROM market_meta w USE INDEX (primary) INNER JOIN (offer_base x USE INDEX (primary, idx_offer_base_market_id) INNER JOIN (interval_meta y USE INDEX (primary, idx_interval_meta_market_id) INNER JOIN caiso_results z USE INDEX (idx_caiso_results_interval_id) ON y.interval_id = z.interval_id) ON x.offer_id = y.offer_id) ON w.market_id = x.market_id WHERE lower(market_name) = 'caiso' AND y.interval_dt >= '",report_start,"' AND y.interval_dt <= '",report_end,"';", sep = "")

Query for MISO –

miso_qry = paste("SELECT w.market_id, w.market_name, x.offer_id, x.identifier_1, x.identifier_2, x.region_name, y.interval_id, y.mkt_run_id, y.interval_dt, y.interval_start, y.interval_end, y.opr_hour, y.opr_interval, z.reg_max, z.reg_min, z.regoff_price, z.regself_limit, z.spinoff_price, z.spinself_limit, z.onsupp_price, z.onsuppself_limit, z.offsupp_price, z.offsuppself_limit, z.regavg_mcp, z.regavg_cap, z.spinavg_mcp, z.spinavg_cap, z.suppavg_mcp, z.suppavg_cap FROM market_meta w USE INDEX (primary) INNER JOIN (offer_base x USE INDEX (primary, idx_offer_base_market_id) INNER JOIN (interval_meta y USE INDEX (primary, idx_interval_meta_market_id) INNER JOIN miso_results z USE INDEX (idx_miso_results_interval_id) ON y.interval_id = z.interval_id) ON x.offer_id = y.offer_id) ON w.market_id = x.market_id WHERE lower(market_name) = 'miso' AND y.interval_dt >= '",report_start,"' AND y.interval_dt <= '",report_end,"';", sep = "")

Query for ISO-NE –

isone_qry = paste("SELECT w.market_id, w.market_name, x.offer_id, x.identifier_1, x.identifier_2, x.region_name, y.interval_id, y.mkt_run_id, y.interval_dt, y.interval_start,

y.interval_end, y.opr_hour, y.opr_interval, z.reglimit_low, z.reglimit_high, z.reg_status, z.autoresp_rate, z.regoff_price, z.regserv_price, z.regcap_price, z.regito_cost FROM market_meta w USE INDEX (primary) INNER JOIN (offer_base x USE INDEX (primary, idx_offer_base_market_id) INNER JOIN (interval_meta y USE INDEX (primary, idx_interval_meta_market_id) INNER JOIN isone_results z USE INDEX (idx_isone_results_interval_id) ON y.interval_id = z.interval_id) ON x.offer_id = y.offer_id) ON w.market_id = x.market_id WHERE lower(market_name) = 'iso-ne' AND y.interval_dt >= '",report_start,"' AND y.interval_dt <= '",report_end,"';", sep = "")

Query for SPP –

spp_qry = paste("SELECT w.market_id, w.market_name, x.offer_id, x.identifier_1, x.identifier_2, x.region_name, y.interval_id, y.mkt_run_id, y.interval_dt, y.interval_start, y.interval_end, y.opr_hour, y.opr_interval, z.coreg_down, z.coreg_up, z.mfreg_down, z.mfreg_up, z.moreg_down, z.moreg_up, z.spin_price, z.supp_price FROM market_meta w USE INDEX (primary) INNER JOIN (offer_base x USE INDEX (primary, idx_offer_base_market_id) INNER JOIN (interval_meta y USE INDEX (primary, idx_interval_meta_market_id) INNER JOIN spp_results z USE INDEX (idx_spp_results_interval_id) ON y.interval_id = z.interval_id) ON x.offer_id = y.offer_id) ON w.market_id = x.market_id WHERE lower(market_name) = 'spp' AND y.interval_dt >= '",report_start,"' AND y.interval_dt <= '",report_end,"';", sep = "")

Replace *"query_variable"* with the appropriate query-variable name in the below command:

rs <- dbSendQuery(mydb, *query_variable*)

data <- fetch(rs, n = -1)

Kindly note that the above command may require a few minutes to execute depending upon the amount of data being requested from the database server and the available network bandwidth.

To write the results to a CSV file, use the following command:

write.csv(data, "results.csv", sep = ",", row.names = FALSE, col.names = TRUE)

To clear the results from the memory and disconnect the session with the database, use the following commands:

dbClearResult(rs)

dbDisconnect(mydb)

2. **MySQL Shell Command-line Interface –**

The MySQL Shell command-line interface installable can be downloaded from the URL: https://dev.mysql.com/downloads/shell/

To log into the "not-printed" database on the MySQL server, use the following command (from the Terminal/Windows command prompt):

mysqlsh --dbuser dbadmin --dbpassword --host NOT-PRINTED --port NOT-PRINTED --schema not-printed –sql

To check the dates for all markets about which the "not-printed" database stores results, use the following SQL statement (it will take at least about 12-15 minutes to execute):

SELECT y.market_name AS market_name, min(x.interval_dt) AS oldest_dt, max(x.interval_dt) AS latest_dt FROM interval_meta x INNER JOIN market_meta y ON x.market_id = y.market_id GROUP BY y.market_name;

To check the dates for a particular market about which the "not-printed" database stores results, use the following SQL statement:

SELECT min(interval_dt) AS oldest_dt, max(interval_dt) AS latest_dt FROM interval_meta WHERE market_id = (SELECT DISTINCT market_id FROM market_meta WHERE lower(market_name) = '*db-market*');

Note: The literal *db-market* can take the values *caiso*, *miso*, *iso-ne*, or *spp*.

3. SQL Statements for Data Export

Connect to the below server using a remote desktop session for the output file –

Database Server: NOT-PRINTED

Data Export Location: C:\ProgramData\MySQL\MySQL Server 5.7\Uploads

However, the below SQL statements can be executed on your local machine using the MySQL Shell Command-line Interface.

Full Report for CAISO

```
SET @start_dt:='YYYY-MM-DD', @end_dt:='YYYY-MM-DD';

SELECT w.market_id, w.market_name, x.offer_id, x.identifier_1, x.identifier_2, x.region_name, y.interval_id, y.mkt_run_id, y.interval_dt, y.interval_start, y.interval_end, y.opr_hour, y.opr_interval, z.nsreq_max, z.nsreq_min, z.rdreq_max, z.rdreq_min, z.rmdreq_max, z.rmdreq_min, z.rureq_max, z.rureq_min, z.rmureq_max, z.rmureq_min, z.spreq_max, z.spreq_min, z.nsproc_cap, z.nsself_cap, z.nscost_line, z.nsclr_price, z.nstot_cap, z.rdproc_cap, z.rdself_cap, z.rdcost_line, z.rdclr_price, z.rdtot_cap, z.rmdproc_cap, z.rmdself_cap, z.rmdcost_line, z.rmdclr_price, z.rmdtot_cap, z.ruproc_cap, z.ruself_cap, z.rucost_line, z.ruclr_price, z.rutot_cap, z.rmuproc_cap, z.rmuself_cap, z.rmucost_line, z.rmuclr_price, z.rmutot_cap, z.spproc_cap, z.spself_cap, z.spcost_line, z.spclr_price, z.sptot_cap
FROM market_meta w USE INDEX (primary) INNER JOIN (offer_base x USE INDEX (primary, idx_offer_base_market_id) INNER JOIN (interval_meta y USE INDEX (primary, idx_interval_meta_market_id) INNER JOIN caiso_results z USE INDEX (idx_caiso_results_interval_id) ON y.interval_id = z.interval_id) ON x.offer_id = y.offer_id) ON w.market_id = x.market_id WHERE lower(market_name) = 'caiso' AND y.interval_dt >= @start_dt AND y.interval_dt <= @end_dt INTO OUTFILE "C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/caiso_results.csv" FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';
```

Full Report for MISO

```
SET @start_dt:='YYYY-MM-DD', @end_dt:='YYYY-MM-DD';

SELECT w.market_id, w.market_name, x.offer_id, x.identifier_1, x.identifier_2, x.region_name, y.interval_id, y.mkt_run_id, y.interval_dt, y.interval_start, y.interval_end, y.opr_hour, y.opr_interval, z.reg_max, z.reg_min, z.regoff_price, z.regself_limit, z.spinoff_price, z.spinself_limit, z.onsupp_price, z.onsuppself_limit, z.offsupp_price, z.offsuppself_limit, z.regavg_mcp, z.regavg_cap, z.spinavg_mcp, z.spinavg_cap, z.suppavg_mcp, z.suppavg_cap FROM market_meta w USE INDEX (primary) INNER JOIN (offer_base x USE INDEX (primary, idx_offer_base_market_id) INNER JOIN (interval_meta y USE INDEX (primary, idx_interval_meta_market_id) INNER JOIN miso_results z USE INDEX (idx_miso_results_interval_id) ON y.interval_id = z.interval_id) ON x.offer_id = y.offer_id) ON w.market_id = x.market_id WHERE lower(market_name) = 'miso' AND y.interval_dt >= @start_dt AND y.interval_dt <= @end_dt INTO OUTFILE "C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/miso_results.csv" FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';
```

Full Report for ISO-NE

```
SET @start_dt:='YYYY-MM-DD', @end_dt:='YYYY-MM-DD';

SELECT w.market_id, w.market_name, x.offer_id, x.identifier_1, x.identifier_2, x.region_name, y.interval_id, y.mkt_run_id, y.interval_dt, y.interval_start, y.interval_end, y.opr_hour, y.opr_interval, z.reglimit_low, z.reglimit_high, z.reg_status, z.autoresp_rate, z.regoff_price, z.regserv_price, z.regcap_price, z.regito_cost FROM market_meta w USE INDEX (primary) INNER JOIN (offer_base x USE INDEX (primary, idx_offer_base_market_id) INNER JOIN (interval_meta y USE INDEX (primary, idx_interval_meta_market_id) INNER JOIN isone_results z USE INDEX (idx_isone_results_interval_id) ON y.interval_id = z.interval_id) ON x.offer_id = y.offer_id) ON w.market_id = x.market_id WHERE lower(market_name) = 'iso-ne' AND y.interval_dt >= @start_dt AND y.interval_dt <= @end_dt INTO OUTFILE "C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/isone_results.csv" FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';
```

Full Report for SPP

```
SET @start_dt:='YYYY-MM-DD', @end_dt:='YYYY-MM-DD';

SELECT w.market_id, w.market_name, x.offer_id, x.identifier_1, x.identifier_2, x.region_name, y.interval_id, y.mkt_run_id, y.interval_dt, y.interval_start, y.interval_end, y.opr_hour, y.opr_interval, z.coreg_down, z.coreg_up, z.mfreg_down, z.mfreg_up, z.moreg_down, z.moreg_up, z.spin_price, z.supp_price FROM market_meta w USE INDEX (primary) INNER JOIN (offer_base x USE INDEX (primary, idx_offer_base_market_id) INNER JOIN (interval_meta y USE INDEX (primary, idx_interval_meta_market_id) INNER JOIN spp_results z USE INDEX (idx_spp_results_interval_id) ON y.interval_id = z.interval_id) ON x.offer_id = y.offer_id) ON w.market_id = x.market_id WHERE lower(market_name) = 'spp' AND y.interval_dt >= @start_dt AND y.interval_dt <= @end_dt INTO OUTFILE "C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/spp_results.csv" FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';
```

Backup and Recovery Steps

The steps to back up the NOT-PRINTED database are as follows –

1. Open the MySQL Workbench tool and login through the "DBA" connection. You will be connected to the "not-printed" schema by default.
2. Click on the "Data Export" option in the management pane under the navigator menu.
3. Check mark the "not-printed" schema under the "Tables to Export" selection menu of the "Object Selection" tab. Ensure that you select the "Dump Structure and Data" from the drop-down list.
4. Check mark "Dump Stored Procedures and Functions", "Dump Events" and "Dump Triggers" from the "Objects to Export" selection menu.
5. Choose the radio button that corresponds to "Export to Dump Project Folder" and specify the destination of the dump in the "Export Options" selection menu. Check mark the "Include Create Schema" checkbox too.

6. Click the “Start Export” button and monitor the progress of the export backup by clicking on the “Export Progress” tab. Kindly ignore any warnings that might popup at the beginning of the export operation.
7. Your database will be backed up once the export operation completes successfully.

The steps to perform a point-in-time recovery of the database are as follows –

1. Restart MySQL using the existing configuration file (my.ini).
    a. Stop MySQL Server using the “MySQL Notifier” utility.
    b. Wait till the “MySQL Notifier” utility notifies you that the server’s status has changed from “Running” to “Stopped”.
    c. Start MySQL Server using the “MySQL Notifier” utility.
    d. Wait till the “MySQL Notifier” utility notifies you that the server’s status has changed from “Stopped” to “Running”.
2. Drop the existing schema by logging into the MySQL command-line client using the “root” user’s password and execute the following command: DROP DATABASE NOT-PRINTED;

    Note: Kindly ensure that you have closed MySQL Workbench before you execute the above statement.
3. Recreate the “NOT-PRINTED” schema by executing the command: CREATE DATABASE NOT-PRINTED;

    Note: The data import will fail if you skip this step, so be careful enough.
4. Start MySQL Workbench to perform the data import.
    a. Click the “Data Import/Restore” menu under the “Management” section of the navigator pane.
    b. Click the “Import from Disk” tab and select the radio button that corresponds to the “Import from Dump Project Folder” option and select the correct dump folder to restore.
    c. Make sure that you import both the dump structure and data by selecting the “Dump Structure and Data” option from the drop-down list.
    d. Start the import operation by clicking on the “Start Import” button.

e. Monitor the progress of the import operation by clicking on the “Import Progress” tab and wait till it finishes. It takes around 2-4 hours for the import to complete.

5. Run the following command to optimize the performance of the database on your machine. This command defragments your data files, analyzes all tables and rebuilds their indexes to improve performance.

   OPTIMIZE TABLE CAISO_RESULTS, MISO_RESULTS, ISONE_RESULTS, SPP_RESULTS, INTERVAL_META, OFFER_BASE, MARKET_META;

   Note: It is recommended to use this command every time you make a significant/large number of inserts/updates/deletes to the database. For example, if you insert 1,000+ rows into the “interval_meta”, you should run the command “optimize table interval_meta;” to optimize the performance of this table.

Results

The following R script can be used to extract full reports from the database –

Note: Kindly uncomment the following line before you run the script –

#fetchReport('CAISO', '2017-01-01', '2017-01-31')

You may modify the market name, start date and end date in the above line as per your requirement.