ECE 356 – Database Systems

Final Project

Group 34

Connor Sweet - cssweet@uwaterloo.ca

Salmaan Khan – s448khan@uwaterloo.ca

Amio Rahman – am4rahma@uwaterloo.ca

Repository: https://github.com/connorssweet/ece356_databases_project

Table of Contents

1. Command-Line Client Application

Ideal CLI

The ideal CLI should strive for two main goals, complexity in the type and quantity of data that can be shown, and simplicity and readability when it comes to using the CLI and outputting the results of the commands. A good CLI should be able to provide a forum for the client to get the data that they choose, while abstracting away the databases from which the data is obtained, all while keeping things complex. All these things in tandem are difficult, as they are in direct conflict, but a good CLI can find a balance between complexity of input and simplicity of use and output.

The reason an ideal CLI should have complexity in the acquisition of data is due to the clients' probable need of complex data. The average client should be able to make any necessary restrictions on the data they need so that it may actually be of some use to them. Most clients do not want to see all the data at once, they want to look at a subset of data that concerns their needs. Given that this is the most probable use of the CLI by the client, it is expected that the client should be allowed as much flexibility as possible in this regard, so that the CLI can accommodate many clients all of whom have vastly different expectations of what they want out of the CLI.

An ideal CLI should also have immense simplicity when it comes to usability. The vast majority of clients for this CLI would not be those who are tremendously versed in SQL and databases, so it is of great benefit to have the CLI abstract away the difficulty that comes with SQL and allow the user to execute commands of high complexity in a language they can understand.

In addition to this, a good CLI can permit and restrict access when needed. This allows for order when it comes to the tables and permits only certain administrators can drastically change the underlying tables.

In a realistic, practical situation involving the developed CLI in question, an Internet Service Provider may utilize the application to aggregate their traffic flow analytics, which will help them adjust their plans to fit the consumer base. The interface permits two different user roles to ensure a general user would not be able to adversely modify entries within the database. The admin, in this situation, would be a System Administrator presiding over the database, and they are given full access privileges, as it is assumed the person in this role is qualified to manage the entries within the database, and hence is given full access privileges to delete entries. A regular user, on the other hand, can create, read and update, as these will likely be necessary operations when handling customer requests on the job. A malicious user might intend to delete entries within the database however, and as such, they are prohibited from using the delete command.

In the scope of the situation explained above, queries would likely pertain to the accumulated statistics associated with each Flow, as this describes the quality of transmission over a connection. This information is useful to an ISP, as they can use this data to improve the conditions of their connections, as well as query for specific areas of interest where anomalies may be detected. It can also be assumed that the most frequently used command, from a user standpoint, is read, and only a smaller portion of clients will be interested in creating, updating and deleting. As such, read has the highest level of complexity among the four commands, to ensure there is ample verbosity to handle the many use cases it will likely experience.

CLI Implementation

The CLI we planned to implement was built on the basis of the ideal CLI outlined above. The two main facets of our CLI were complexity of commands and ease of use for the client. This prospect was a difficult one; given the time constraints and the difficulty of this problem, an ideal CLI is difficult to produce, but the CLI we were able to execute is a version of the ideal CLI with a reasonable balance between complexity and ease of use.

Our proposed CLI was formed around four main operations: Create, Read, Update, and Delete. These four commands were vital to the function of a basic CLI. Each command would require additional flags for the CLI to identify the exact operation to conduct on the underlying tables.

We also proposed for the CLI to have some auxiliary operations. These would include Help, Relog, and Exit.

Create Command

We proposed for our Create command to allow the user to add either a Host, a Connection, or a Flow to the dataset. To differentiate between the three possible entities that could be created, we used a flag to indicate which item is to be created. Initially we proposed that the Create command would disallow certain creations of data that would cause issues in our tables related to invalid foreign key constraints. An example of this would be rejecting a create command of a connection between two hosts, one of which does not exist in the Host table. During implementation, we discovered that these checks would be difficult and time-consuming to do, so we found a solution to the issue that was simple to implement. Due to time constraints, we decided to implement our CLI to implicitly attempt to create required Connections and Hosts when creating an entity that depends on them. If the Host/Connection already exists, then the command to create them does nothing. However, if they do not exist, they are created, thus allowing the Flow/Connection to be created without issue. Given more time, this workaround would be removed, and the initially proposed solution would be implemented.

The Create command we proposed would have a few mandatory flags. For Host, the flags would consist only of an IP and a Port. For Connection, the flags would consist of two sets of IPs and hosts, one set for the source host, and one for the destination host. We have implemented both as proposed. An example of a valid Create command for the host would be:

```
create –host –ip=’255.207.168.0’ -port=8080
```

The proposed Create command for a Flow would have several mandatory flags. The first would be source and destination IPs and ports. This should correspond to a valid connection. Next, a valid flowKey and flowStart should be provided. This pair of information would provide a unique identifier for the flow. Lastly, a protocolType should be provided. This is the basic information required for a Flow, and all other statistical information is not necessary.

While the ideal Create command for a Flow would have a simple way of creating a flow with all its accompanying statistics, we found that this was unfeasible to implement while trying to keep the CLI simple. It is for this reason that we decided to make all the statistical information optional. Our proposed Create command stripped back some of the complexity afforded to the client by SQL and allowed for a simpler process to create Flows. We decided that the expectation for the client would be to add only the bare minimum required for a flow during the creation phase, and for the non-essential statistical information to be added during the update phase if the client deems it necessary.

Read Command

We proposed for our Read command to allow for the most user complexity out of all our CLI commands. As stated in the ideal CLI portion, we have identified that most of our client base would use the CLI for read commands. We proposed that the read command should allow the user to read any set of data they want about the flow, and that they may make restrictions on the data they want to see as well. To accomplish this, we required the user to select up to two different sets of flags to identify both the selected columns and the selection criteria. To enforce structure on the command, the format of the read command was as follows:

```
read –s <select flags> -w <operator> <condition flags> <limit>
```

The select flags allow the user to select the columns that they want to see. We also proposed a flag that allowed the user to select all columns at once, as this is a desirable feature. We initially proposed for the conditions on the data to be connected with any chosen operator, but in practice we found that this added too much additional complexity to the CLI. To simplify this issue, we proposed an operator flag which allows for all the subsequent conditions to be connected using that operator. This reduces the complexity with which the read command can function, but it increases the useability. Lastly, we proposed an optional limit flag which limits the output of the read. This is useful since the databases are very large and the clients usually only want to look at a small portion of the data. Given the restrictions and time constraints, we have implemented a read command whose format is outlined above.

Update Command

We proposed for a versatile Update command which allows for any statistic belonging to any Flow to be updated. The proposed format for the Update command is:

```
update –flowKey=VALUE –flowStart=VALUE <optional flags>
```

This would allow for a user to update any statistic given that they enter the correct flowKey and flowStart so that the CLI can uniquely identify the flow being referred to. The ideal update command would allow for more flexibility as to which flow(s) would be updated with one command. For example, a client may want to update the protocolType of all flows who have a given source IP and port. Given the complexity of this issue and the time constraints at hand, we did not implement this more sophisticated version of the Update command.

Our simple implementation of the Update command allows a user to identify the flow that they want to update and to provide new values for the statistics/control information they want to change.

Delete Command

We proposed for our Delete command to be simple. The Delete command would take flowKey and flowStart flags, and would delete the entry that corresponds to that flowKey and flowStart. Given that flowKey and flowStart in conjunction are a unique identifier for a Flow, requesting these values for deleting a flow would be mandatory. The ideal delete command would also allow for deleting any entry that satisfies a given condition (i.e. delete any flows that has flowDuration < 0.01). Given the complexity of this issue and the time constraints at hand, we implemented a simple delete which allows only for deletion of individual flows via their unique identifiers.

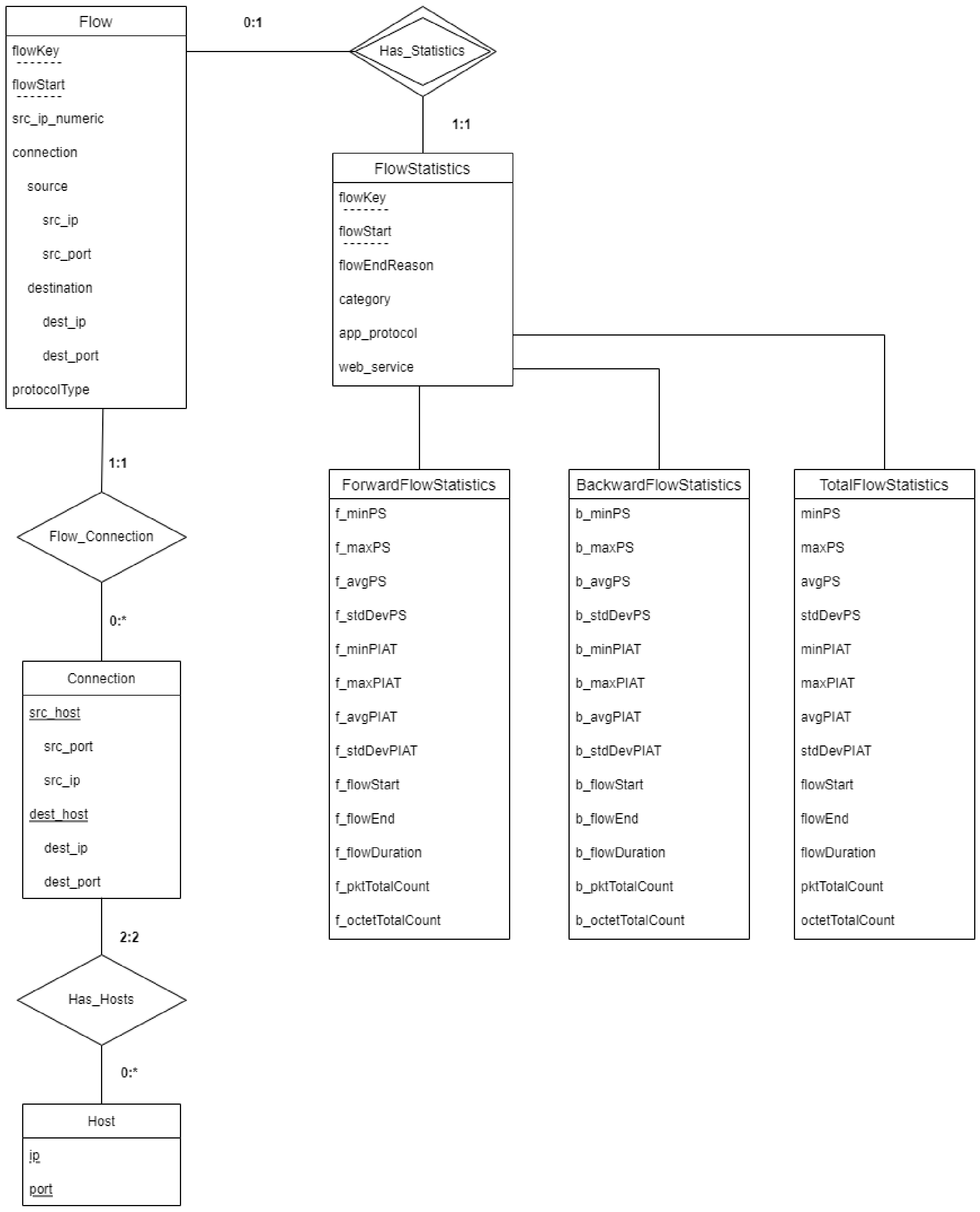
Additionally, we have setup the delete command to only be used by users with administrative privileges. We made this decision to prevent potential malicious attacks on the database from someone with client privileges. By allowing clients to only create, read, and update, the amount of damage a malicious user could do would be minimal. Allowing only admins to delete data from the dataset would be ideal for the health of the data.

Other Commands

We proposed for our CLI to include three more commands: help, relog, and exit. At startup, our CLI prompts the user to login using their credentials. An ideal CLI would have a complex system of storing usernames and passwords, but out simple implementation has just one username and password for each of the client and the admin. After a successful login, the user can execute commands. If the user executes the relog command, they will be prompted to login again. This command is useful if a user logs in as a client and wants to relog in as an admin, given they have the requisite credentials.

The help command simply provides information on the available commands. Help provides descriptions and flag information for the create, read, update, and delete commands, along with information about the format and flags for these commands. Lastly, the exit command simply exits the CLI and terminates the program. All three of these commands are simple and were implemented exactly as proposed.

2. Entity-Relationship Design



The entity-relationship diagram included above describes the various entity sets and their respective relationships within the constructed database system. Relevant attributes from the provided data set were categorized to form seven entity sets that are necessary for modeling the infrastructure of the database.

Primarily, the Flow entity stores general information relating to the flow traffic in question, including references to its relevant connection, and information surrounding the protocol. Given that many of the later attributes act as statistics pertaining to certain flows, it is no surprise that Flow has relations with other relevant entity sets, as an entry will be added for every performed flow. All other entity sets are related to Flow, and many rely on the primary key of Flow, flowKey, in correlation with flowStart, to ensure all their entries are exclusive. This flowKey is a unique hash-based identifier for a given flow, and thus is a necessary foreign key for many remaining entity sets. It was decided that this flowKey, in conjunction with the flowStart time, would be sufficient as to make all recorded entities exclusive, as it is impossible for two items to be transmitted over the connection at the exact same time. The attributes included in this entity set are detailed in the table below:

Attributes in the "Flow" Entity Set		
Attribute	**Type**	**Description**
flowKey	varchar(250)	Hash-based unique flow identifier
flowStart	Decimal(40,25)	Start time in seconds of flow in UNIX time format
src_ip_numeric	bigint	Decimal format of source IP
src_ip	varchar(250)	Network format of source IP
src_port	int	Port number for the source IP
dest_ip	varchar(250)	Network format of destination IP
dest_port	int	Port number for the destination IP
protocolType	int	IANA transport protocol number

Please note that, in the provided design, the IPs and Ports for both the Source and Destination Hosts are recognized as a Connection. A connection can have zero or many associated Flows. This establishes a cardinality of 0:*on the Flow_Connection relationship set, regarding the Connection entity set. Likewise, it is known that every flow must have a connection. Each flow encompasses the transmission of packets over a known Connection, and because of this definition, it is impossible to omit the necessary Connection attributes. Additionally, a Flow cannot have more than one Connection, since it is impossible to transmit across multiple Connections.

Both Source and Destination Hosts, grouped under the Connection entity set, are necessary inclusions in the entity-relationship diagram due to their IP and Port attributes, which, in combination, serve as the primary key for the aforementioned set. Each Connection includes two Hosts, one being the Source and the other being a Destination. The Connection attributes residing in the Flow entity set, conveying the IP and Port of both the Source and Destination, will serve as foreign keys, given that their values are derived from related entries in the Connections entity set.

Attributes in the "Connection" Entity Set		
Attribute	**Type**	**Description**
src_ip	varchar(250)	Network format of source IP
src_port	int	Port number for the source IP
dest_ip	varchar(250)	Network format of destination IP
dest_port	int	Port number for the destination IP

As mentioned previously, every Connection has two associated Host entity sets. This affirms another relationship in the diagram, known as Has_Hosts. This relationship, establishing a link between Host and Connection, is supported by the notion that two Host entities must exist for a Connection to be created. It is not possible to have a Connection between more than two Hosts, and similarly, an assumption is made that a Host cannot connect to itself. Thus, in the perspective of this relationship set, the cardinality of Connection is 2:2. There are no restrictions imposed on the Host entity set by the remainder of the system, given that there can be as many or as few Hosts as necessary for the established Flows. Thus, the cardinality is 0:*.

Each Host is composed of two attributes, known as ip and port. Together, these serve as the primary key for this entity set. These are also foreign keys in Connection, when the related attributes from both Hosts are combined, and in Flow, through the established Connection.

Attributes in the "Host" Entity Set		
Attribute	**Type**	**Description**
ip	varchar(250)	Network format of IP
port	int	Port number for IP

When considering practical uses for the system being implemented, it was decided that it could be assumed that most of the queries users would perform would relate to the Flow statistics available for every logged Flow. Thus, categorization of the available statistics associated with each Flow would be a valuable asset to the system as a whole. A new entity set was added to the design, called FlowStatistics, which is connected to Flow through the weak relationship set, Has_Statistics. It is assumed that it is not necessary for all Flow entities to have associated statistics available, and there can only ever be one set of statistics for each provided Flow. In the scope of this relationship, the cardinality of Flow is 0:1. FlowStatistics entities, on the other hand, can only correlate to one Flow. It must have a related Flow and does not apply to multiple Flows. Thus, the cardinality of FlowStatistics is 1:1.

FlowStatistics is further specialized into three additional entity sets, which segregate the analytics of the total trip from those belonging to the forward or backward trips individually. There is value to specializing the entity set in this manner, given that much of the information a user will intend to query is likely going to be housed in the TotalFlowStatistics table, and this allows for an efficient way to isolate many of the required indexes to this entity, which will eventually be represented as a table. It is anticipated that users will be less interested in statistics correlating to the forward and backward trips individually, but the attributes still hold significant merit to system users that may be surveying connection problems or examining anomalies during packet transmission. The general FlowStatistics entity set is defined below, followed by its three specializations:

Attributes in the "FlowStatistics" Entity Set		
Attribute	**Type**	**Description**
flowKey	varchar(250)	Hash-based unique flow identifier
flowStart	Decimal(40,25)	Start time in seconds of flow in UNIX time format
flowEndReason	int	Backward packet arrival time standard deviation
category	varchar(250)	Type of nDPI communication
app_protocol	varchar(250)	Type of nDPI-detected application protocol
web_service	varchar(250)	Type of nDPI-detected web-service

Attributes in the "ForwardFlowStatistics" Entity Set		
Attribute	**Type**	**Description**
f_minPS	Int	Forward direction minimum packet size
f_maxPS	Int	Forward direction maximum packet size
f_avgPS	Decimal(40,25)	Forward direction average packet size
f_stdDevPS	Decimal(40,25)	Forward direction standard deviation of packet size
f_minPIAT	Decimal(40,25)	Forward direction minimum packet interarrival time
f_maxPIAT	Decimal(40,25)	Forward direction maximum packet interarrival time
f_avgPIAT	Decimal(40,25)	Forward direction average packet interarrival time
f_stdDevPIAT	Decimal(40,25)	Forward direction standard deviation of packet interarrival time
f_flowStart	Decimal(40,25)	Start time in seconds of flow in UNIX time format on forward trip
f_flowEnd	Decimal(40,25)	End time in seconds of flow in UNIX time format on forward trip
f_flowDuration	Decimal(40,25)	Duration in seconds of flow in UNIX time format on forward trip

f_pktTotalCount	Int	Tally of all packets on forward trip
f_octetTotalCount	bigint	Tally of transmitted bytes with attention to the IP payload on forward trip

Attributes in the "BackwardFlowStatistics" Entity Set		
Attribute	**Type**	**Description**
b_minPS	Int	Backward direction minimum packet size
b_maxPS	Int	Backward direction maximum packet size
b_avgPS	Decimal(40,25)	Backward direction average packet size
b_stdDevPS	Decimal(40,25)	Backward direction standard deviation of packet size
b_minPIAT	Decimal(40,25)	Backward direction minimum packet interarrival time
b_maxPIAT	Decimal(40,25)	Backward direction maximum packet interarrival time
b_avgPIAT	Decimal(40,25)	Backward direction average packet interarrival time
b_stdDevPIAT	Decimal(40,25)	Backward direction standard deviation of packet interarrival time
b_flowStart	Decimal(40,25)	Start time in seconds of flow in UNIX time format on backward trip
b_flowEnd	Decimal(40,25)	End time in seconds of flow in UNIX time format on backward trip
b_flowDuration	Decimal(40,25)	Duration in seconds of flow in UNIX time format on backward trip
b_pktTotalCount	Int	Tally of all packets on backward trip
b_octetTotalCount	bigint	Tally of transmitted bytes with attention to the IP payload on backward trip

Attributes in the "TotalFlowStatistics" Entity Set		
Attribute	**Type**	**Description**
minPS	Int	Total minimum packet size
maxPS	Int	Total maximum packet size
avgPS	Decimal(40,25)	Total average packet size

stdDevPS	Decimal(40,25)	Total standard deviation of packet size
minPIAT	Decimal(40,25)	Total minimum packet interarrival time
maxPIAT	Decimal(40,25)	Total maximum packet interarrival time
avgPIAT	Decimal(40,25)	Total average packet interarrival time
stdDevPIAT	Decimal(40,25)	Total standard deviation of packet interarrival time
flowStart	Decimal(40,25)	Start time in seconds of flow in UNIX time format
flowEnd	Decimal(40,25)	End time in seconds of flow in UNIX time format
flowDuration	Decimal(40,25)	Duration in seconds of flow in UNIX time format
pktTotalCount	Int	Tally of all packets
octetTotalCount	bigint	Tally of transmitted bytes with attention to the IP payload

3. Relational Schema

The following schema was developed in accordance with the intended functionality established by the entity-relationship diagram proposed above. It has been separated into smaller subsections which will be further examined to convey the infrastructure surrounding the database system. To begin, any pre-existing tables are dropped to ensure the following creations are successful:

```
DROP TABLE IF EXISTS TotalFlowStatistics;
DROP TABLE IF EXISTS BackwardFlowStatistics;
DROP TABLE IF EXISTS ForwardFlowStatistics;
DROP TABLE IF EXISTS FlowStatistics;
DROP TABLE IF EXISTS Flow;
DROP TABLE IF EXISTS Connection;
DROP TABLE IF EXISTS Host;
DROP TABLE IF EXISTS InternetTrafficInternal;
```

A requirement of the relational schema was that the dataset located on the provided csv had to be loaded into the system. To accomplish this efficiently, a table was proposed called InternetTrafficInternal. This table serves no purpose for the end users of the system, and rather is used solely for the purpose of loading the required data. The following code snippet outlines the process of importing this data:

```
CREATE TABLE InternetTrafficInternal(
    flow_key varchar(250),
    src_ip_numeric bigint,
    src_ip varchar(250),
    src_port int,
    dst_ip varchar(250),
    dst_port int,
    proto int,
    pktTotalCount int,
    octetTotalCount bigint,
    min_ps int,
    max_ps int,
    avg_ps decimal(40,25),
    std_dev_ps decimal(40,25),
    flowStart decimal(40,25),
    flowEnd decimal(40,25),
    flowDuration decimal(40,25),
    min_piat decimal(40,25),
    max_piat decimal(40,25),
    avg_piat decimal(40,25),
    std_dev_piat decimal(40,25),
    f_pktTotalCount int,
    f_octetTotalCount bigint,
    f_min_ps int,
    f_max_ps int,
    f_avg_ps decimal(40,25),
    f_std_dev_ps decimal(40,25),
    f_flowStart decimal(40,25),
    f_flowEnd decimal(40,25),
    f_flowDuration decimal(40,25),
    f_min_piat decimal(40,25),
    f_max_piat decimal(40,25),
    f_avg_piat decimal(40,25),
    f_std_dev_piat decimal(40,25),
    b_pktTotalCount int,
    b_octetTotalCount bigint,
    b_min_ps int,
    b_max_ps int,
    b_avg_ps decimal(40,25),
    b_std_dev_ps decimal(40,25),
    b_flowStart decimal(40,25),
    b_flowEnd decimal(40,25),
    b_flowDuration decimal(40,25),
    b_min_piat decimal(40,25),
    b_max_piat decimal(40,25),
    b_avg_piat decimal(40,25),
    b_std_dev_piat decimal(40,25),
    flowEndReason int,
    category varchar(250),
    application_protocol varchar(250),
    web_service varchar(250)
);

LOAD DATA INFILE '/var/lib/mysql-files/21-Network-Traffic/Unicauca-dataset-
April-June-2019-Network-flows.csv' INTO TABLE InternetTrafficInternal
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\n'
  IGNORE 1 ROWS
  (
    flow_key,
    src_ip_numeric,
    src_ip,
    src_port,
    dst_ip,
    dst_port,
    proto,
    pktTotalCount,
    octetTotalCount,
    min_ps,
    max_ps,
    avg_ps,
    std_dev_ps,
    flowStart,
    flowEnd,
    flowDuration,
    min_piat,
    max_piat,
    avg_piat,
    std_dev_piat,
    f_pktTotalCount,
    f_octetTotalCount,
    f_min_ps,
    f_max_ps,
    f_avg_ps,
    f_std_dev_ps,
    f_flowStart,
    f_flowEnd,
    f_flowDuration,
    f_min_piat,
    f_max_piat,
    f_avg_piat,
    f_std_dev_piat,
    b_pktTotalCount,
    b_octetTotalCount,
    b_min_ps,
    b_max_ps,
    b_avg_ps,
    b_std_dev_ps,
    b_flowStart,
    b_flowEnd,
    b_flowDuration,
    b_min_piat,
    b_max_piat,
    b_avg_piat,
    b_std_dev_piat,
    flowEndReason,
    category,
    application_protocol,
    web_service
  );
```

The entries collected in this table will be obtained by the subsequently created tables, where the entries will remain accessible to the user.

It is noteworthy to indicate that an anomaly was found in the given dataset that prompted an adjustment to the structure of the schema. It was found that many Flows could share the same flowKey, despite being separate entities, although this flowKey is described as a hash-based unique identifier for each flow. To ensure there was exclusivity for each entry in the Flow table, flowStart was added as part of the primary key, on the assumption that it is impossible to commence transmission of two items over one Connection at the same time. However, there appears to be a further error in the gathered dataset, as there was a recognized pair of entries that shared a flowKey and flowStart time. It was determined that this is a physically impossible occurrence, and thus conditions were imposed when importing all items from InternetTrafficInternal to ignore duplicate entries, ensuring unnatural occurrences like these entries are filtered from the resulting dataset. It was concluded that this impossibility was most likely due to the inaccuracy of the UNIX timestamp for the provided data, causing there to be multiple flows starting from the same host with the same flowKey at the same time.

The next table created is Host, which is representative of the described entity set of the same name.

```
CREATE TABLE Host (
    ip varchar(250) NOT NULL,
    port int NOT NULL CHECK (port>=0 AND port<=65535),
    PRIMARY KEY(ip,port)
);

INSERT IGNORE INTO Host (ip, port)
SELECT DISTINCT src_ip, src_port FROM InternetTrafficInternal;

INSERT IGNORE INTO Host (ip, port)
SELECT DISTINCT dst_ip, dst_port FROM InternetTrafficInternal;
```

It was determined that a host needed to include both an IP address and port number to be a valid Host, and so both of these specified attributes are enforced to be not null. There is an additional check on the port to ensure the entered integer is a valid port number, residing in the range of 0 and 65535. Given additional time to continue elaborating on the finalized schema, it was planned to introduce a regular expression to check that the IP address is also in valid form, but this proposal was withdrawn due to time constraints. It is assumed, given that the gathered data is legitimate, that the entered IP addresses will be accurate in format. Both attributes of the Host table form a primary key together, as two Hosts cannot share an IP and port. This Host table is populated with the sources and destinations associated with all distinct Flows.

Next, a table is made for the Connection entity set, which contains keys referencing the foreign Host tables mentioned above. Each Connection, modelled after its corresponding entity set, contains attributes for the IP and port for both the Source and Destination locations.

```
CREATE TABLE Connection (
    src_ip varchar(250) NOT NULL,
    src_port int NOT NULL,
    dest_ip varchar(250) NOT NULL,
    dest_port int NOT NULL,
    PRIMARY KEY(src_ip, src_port, dest_ip, dest_port),
    FOREIGN KEY (src_ip, src_port) REFERENCES Host(ip,port) ON DELETE CASCADE
ON UPDATE CASCADE,
    FOREIGN KEY (dest_ip, dest_port) REFERENCES Host(ip,port) ON DELETE CASCADE
ON UPDATE CASCADE
);

INSERT IGNORE INTO Connection (src_ip, src_port, dest_ip, dest_port)
SELECT DISTINCT src_ip, src_port, dst_ip, dst_port FROM
InternetTrafficInternal;
```

The connection table uses the combination of all four of its attributes as its primary key, given that it is defined as a distinct relationship between two Hosts. However, the pairs of IP address and port for source and destination respectively are foreign keys, which reference related entries in the Host table. Because the Connection table is composed solely of keys utilized in the Host table, it may be considered a weak entity set. Each of these foreign keys include instructions to cascade upon deletion or updating, which was a design decision allowing the modification of any of these entries in Host to propagate to the correlating entries within the Connection table.

Subsequently, the Flow table is created and administered the same attributes as defined in the proposed entity-relationship diagram. Entries to this table are representative of individual Flows over respective Connections. The contained IP and port attributes reference the Connection on which the Flow exists, and thus, when combined, is a foreign key referencing an entry in the Connection table. The propagational delete and update used for attributes in Connection upon modifications made to Host is employed in the Flow table as well, to automatically recognize and adjust for any modifications to the Host entities.

```
CREATE TABLE Flow (
    flowKey varchar(250) NOT NULL,
    flowStart decimal(40,25) NOT NULL,
    src_ip_numeric bigint NOT NULL,
    src_ip varchar(250) NOT NULL,
    src_port int NOT NULL,
    dest_ip varchar(250) NOT NULL,
    dest_port int NOT NULL,
    protocolType int NOT NULL CHECK (protocolType=17 OR protocolType=6 OR
protocolType=1),
    PRIMARY KEY (flowKey, flowStart),
    FOREIGN KEY (src_ip, src_port, dest_ip, dest_port) REFERENCES
Connection(src_ip, src_port, dest_ip, dest_port) ON DELETE CASCADE ON UPDATE
CASCADE
);
```

```
INSERT IGNORE INTO Flow (flowKey, flowStart, src_ip_numeric, src_ip, src_port,
dest_ip, dest_port, protocolType)
SELECT flow_key, flowStart, src_ip_numeric, src_ip, src_port, dst_ip, dst_port,
proto FROM InternetTrafficInternal;
```

Unique identifiers for the Flow and its associated Connection are critical to recognize a Flow entity, and so these related attributes are prohibited from being null. Additionally, given that protocolType can only be 1, 6, or 17, an additional check was added to remove any invalid protocols.

FlowStatistics was established as a table of its own, containing prevalent information concerning details of each specific Flow. Like the Flow table, each entry is uniquely identified using the flowKey and flowStart.

```
CREATE TABLE FlowStatistics(
    flowKey varchar(250) NOT NULL,
    flowStart decimal(40,25) NOT NULL,
    flowEndReason int CHECK(flowEndReason>=0 AND flowEndReason<=5),
    category varchar(250) ,
    app_protocol varchar(250),
    web_service varchar(250),
    PRIMARY KEY (flowKey, flowStart),
    FOREIGN KEY (flowKey, flowStart) REFERENCES Flow(flowKey, flowStart) ON
DELETE CASCADE ON UPDATE CASCADE
);

INSERT IGNORE INTO FlowStatistics (flowKey, flowStart, flowEndReason, category,
app_protocol, web_service)
SELECT flow_key, flowStart, flowEndReason, category, application_protocol,
web_service FROM InternetTrafficInternal;
```

The valuable primary key of this created Table is identical to that of the Flow entity set. Because of this, and its foreign status in this table, this can be seen as a weak key. Because flowKey and flowStart function as a foreign key, the *ON DELETE CASCADE ON UPDATE CASCADE* clause was added so modifications to the flow identifiers are propagated to the subsequently connected tables, and the connection between each table's entries are not obstructed. Additional checks added in this table include the insurance that flowEndReason is between 0 and 5, given that these are the only valid values for this attribute.

Separate tables were created for each of the three specializations of the FlowStatistics, each including the same primary key combination of flowKey and flowStart to retain connectivity with their parent table, FlowStatistics, and hence with the rest of the Flow-centric system. Each of these tables house similar data, but segregate their statistics from each other due to the fact that each presides over a separate window of the Flow's total transmission duration. The first and second table, named ForwardFlowStatistics and BackwardFlowStatistics, hold attributes that convey analytics regarding their respective portions of the trip, while the third table, TotalFlowStatistics, contains statistical values pertaining to the entire flow duration. All three of these specialized tables contain the same primary and foreign keys as established in FlowStatistics, and include the necessary conditions that allow updates from related tables to propagate to these ones.

It is valuable to organize the specialized sets of statistics this way, in these separate tables, due to the benefits of keeping many of the indexable attributes under one specialized table. It is assumed that, from a practical standpoint, users that are interfacing with this database system and are interested in internet traffic analytics will be interested in the TotalFlowStatistics rather than the data that was collected individually for each portion of the trip. Thus, by organizing the information in this fashion, it is possible to coordinate several indexes that preside solely over the TotalFlowStatistics table.

Due to the similarity of the three specialized tables and their attributes, the restrictions and constraints imposed on these entries will all be discussed generally, from the perspective of TotalFlowStatistics. Due to the fact that these values aggregated through a duration where packets are transmitted across a known connection, it is physically impossible for these values to be negative. Thus, checks are imposed on all numerical attributes in each table to ensure they are greater than or equal to zero.

```sql
CREATE TABLE ForwardFlowStatistics(
    flowKey varchar(250) NOT NULL,
    flowStart decimal(40,25) NOT NULL,
    f_minPS int CHECK (f_minPS>=0),
    f_maxPS int CHECK (f_maxPS>=0),
    f_avgPS decimal(40,25) CHECK (f_avgPS>=0),
    f_stdDevPS decimal(40,25) CHECK (f_stdDevPS>=0),
    f_minPIAT decimal(40,25) CHECK (f_minPIAT>=0),
    f_maxPIAT decimal(40,25) CHECK (f_maxPIAT>=0),
    f_avgPIAT decimal(40,25) CHECK (f_avgPIAT>=0),
    f_stdDevPIAT decimal(40,25) CHECK (f_stdDevPIAT>=0),
    f_flowStart decimal(40,25) CHECK (f_flowStart>=0),
    f_flowEnd decimal(40,25) CHECK (f_flowEnd>=0),
    f_flowDuration decimal(40,25) CHECK (f_flowDuration>=0),
    f_pktTotalCount int CHECK(f_pktTotalCount>=0),
    f_octetTotalCount bigint CHECK(f_octetTotalCount>=0),
    PRIMARY KEY (flowKey, flowStart),
    FOREIGN KEY (flowKey, flowStart) REFERENCES Flow(flowKey, flowStart) ON
DELETE CASCADE ON UPDATE CASCADE
);

INSERT IGNORE INTO ForwardFlowStatistics (flowKey, flowStart, f_minPS, f_maxPS,
f_avgPS, f_stdDevPS, f_minPIAT, f_maxPIAT, f_avgPIAT, f_stdDevPIAT,
f_flowStart, f_flowEnd, f_flowDuration, f_pktTotalCount, f_octetTotalCount)
SELECT flow_key, flowStart, f_min_ps, f_max_ps, f_avg_ps, f_std_dev_ps,
f_min_piat, f_max_piat, f_avg_piat, f_std_dev_piat, f_flowStart, f_flowEnd,
f_flowDuration, f_pktTotalCount, f_octetTotalCount FROM
InternetTrafficInternal;
```

```
CREATE TABLE BackwardFlowStatistics(
    flowKey varchar(250) NOT NULL,
    flowStart decimal(40,25) NOT NULL,
    b_minPS int CHECK (b_minPS>=0),
    b_maxPS int CHECK (b_maxPS>=0),
    b_avgPS decimal(40,25) CHECK (b_avgPS>=0),
    b_stdDevPS decimal(40,25) CHECK (b_stdDevPS>=0),
    b_minPIAT decimal(40,25) CHECK (b_minPIAT>=0),
    b_maxPIAT decimal(40,25) CHECK (b_maxPIAT>=0),
    b_avgPIAT decimal(40,25) CHECK (b_avgPIAT>=0),
    b_stdDevPIAT decimal(40,25) CHECK (b_stdDevPIAT>=0),
    b_flowStart decimal(40,25) CHECK (b_flowStart>=0),
    b_flowEnd decimal(40,25) CHECK (b_flowEnd>=0),
    b_flowDuration decimal(40,25) CHECK (b_flowDuration>=0),
    b_pktTotalCount int CHECK(b_pktTotalCount>=0),
    b_octetTotalCount bigint CHECK(b_octetTotalCount>=0),
    PRIMARY KEY (flowKey, flowStart),
    FOREIGN KEY (flowKey, flowStart) REFERENCES Flow(flowKey, flowStart) ON
DELETE CASCADE ON UPDATE CASCADE
);
INSERT IGNORE INTO BackwardFlowStatistics (flowKey, flowStart, b_minPS,
b_maxPS, b_avgPS, b_stdDevPS, b_minPIAT, b_maxPIAT, b_avgPIAT, b_stdDevPIAT,
b_flowStart, b_flowEnd, b_flowDuration, b_pktTotalCount, b_octetTotalCount)
SELECT flow_key, flowStart, b_min_ps, b_max_ps, b_avg_ps, b_std_dev_ps,
b_min_piat, b_max_piat, b_avg_piat, b_std_dev_piat, b_flowStart, b_flowEnd,
b_flowDuration, b_pktTotalCount, b_octetTotalCount FROM
InternetTrafficInternal;

CREATE TABLE TotalFlowStatistics(
    flowKey varchar(250) NOT NULL,
    flowStart decimal(40,25),
    minPS int CHECK (minPS>=0),
    maxPS int CHECK (maxPS>=0),
    avgPS decimal(40,25) CHECK (avgPS>=0),
    stdDevPS decimal(40,25) CHECK (stdDevPS>=0),
    minPIAT decimal(40,25) CHECK (minPIAT>=0),
    maxPIAT decimal(40,25) CHECK (maxPIAT>=0),
    avgPIAT decimal(40,25) CHECK (avgPIAT>=0),
    stdDevPIAT decimal(40,25) CHECK (stdDevPIAT>=0),
    flowEnd decimal(40,25) CHECK (flowEnd>=0),
    flowDuration decimal(40,25) CHECK (flowDuration>=0),
    pktTotalCount int CHECK(pktTotalCount>=0),
    octetTotalCount bigint CHECK(octetTotalCount>=0),
    PRIMARY KEY (flowKey, flowStart),
    FOREIGN KEY (flowKey, flowStart) REFERENCES Flow(flowKey, flowStart) ON
DELETE CASCADE ON UPDATE CASCADE
);

INSERT IGNORE INTO TotalFlowStatistics (flowKey, flowStart, minPS, maxPS,
avgPS, stdDevPS, minPIAT, maxPIAT, avgPIAT, stdDevPIAT, flowEnd, flowDuration,
pktTotalCount, octetTotalCount)
SELECT flow_key, flowStart, min_ps, max_ps, avg_ps, std_dev_ps, min_piat,
max_piat, avg_piat, std_dev_piat, flowEnd, flowDuration, pktTotalCount,
octetTotalCount FROM InternetTrafficInternal;

CREATE INDEX idx_avgPS on TotalFlowStatistics (avgPS);
CREATE INDEX idx_stdDevPS on TotalFlowStatistics (stdDevPS);
CREATE INDEX idx_avgPIAT on TotalFlowStatistics (avgPIAT);
CREATE INDEX idx_stdDevPIAT on TotalFlowStatistics (stdDevPIAT);
CREATE INDEX idx_flowDuration on TotalFlowStatistics (flowDuration);
CREATE INDEX idx_pktTotalCount on TotalFlowStatistics (pktTotalCount);
CREATE INDEX idx_octetTotalCount on TotalFlowStatistics (octetTotalCount);
```

Indexes were created for almost all numerical attributes in TotalFlowStatistics due to their use in describing the quality of a Flow over a given Connection. Indexes were omitted for minPS, maxPS, minPIAT, and maxPIAT, given that these values would have less practical use and are less descriptive of the conditions of the Flow transmission than the avgPS, stdDevPS, avgPIAT, and stdDevPIAT attributes. Additionally, an index was made for flowDuration, but not start and end times, as it is assumed that practical users of this system such as internet service providers and data analytics corporations would focus their interest on the duration of transmission rather than arbitrary starting and ending times. Fast lookup is desirable on the properties indexed above, given their presumed frequent usage in a production-based scenario.

4. Test Cases

We also tested that our tables were loaded with the correct number of rows such that we were not missing crucial data. The following sql queries were run on our mysql database to check if our tables were created with the correct number of rows:

SELECT count(*) from InternetTrafficInternal;

Expected Output: 2704839

SELECT count(*) from Host;

Expected Output: 1903664

SELECT count(*) from Connection;

Expected Output: 2344392

SELECT count(*) from Flow;

Expected Output: 2704608

SELECT count(*) from ForwardFlowStatistics;

Expected Output: 2704608

SELECT count(*) from BackwardFlowStatistics;

Expected Output: 2704608

SELECT count(*) from TotalFlowStatistics;

Expected Output: 2704608

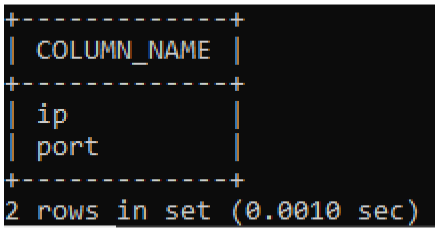
The number of rows in InternetTrafficInternal, Host and Connection are different from the rest because we do not store duplicate values in Connection and Host. InternetTrafficInternal stores all the rows found in our csv.

Furthermore, we test if our tables had the primary keys, foreign keys and indexes necessary for our data set:

For Primary Keys:

SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA="ece356_project" AND TABLE_NAME="Host" AND COLUMN_KEY="PRI";

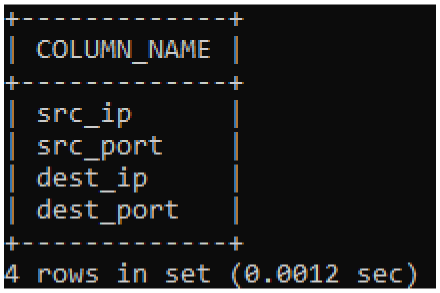
Expected Output:



```
+-------------+
| COLUMN_NAME |
+-------------+
| ip          |
| port        |
+-------------+
2 rows in set (0.0010 sec)
```

SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA = "ece356_project" AND TABLE_NAME = "Connection" AND COLUMN_KEY = "PRI";

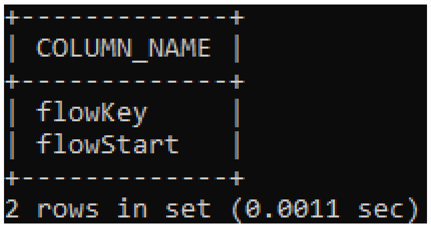
Expected Output:



```
+-------------+
| COLUMN_NAME |
+-------------+
| src_ip      |
| src_port    |
| dest_ip     |
| dest_port   |
+-------------+
4 rows in set (0.0012 sec)
```

SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA = "ece356_project" AND TABLE_NAME = "Flow" AND COLUMN_KEY = "PRI";

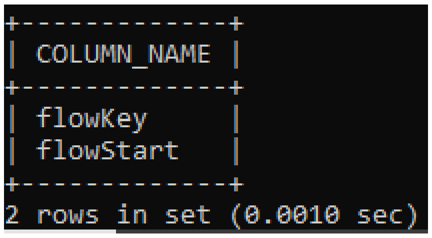
Expected Output:



```
+-------------+
| COLUMN_NAME |
+-------------+
| flowKey     |
| flowStart   |
+-------------+
2 rows in set (0.0011 sec)
```

SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA = "ece356_project" AND TABLE_NAME = "FlowStatistics" AND COLUMN_KEY = "PRI";

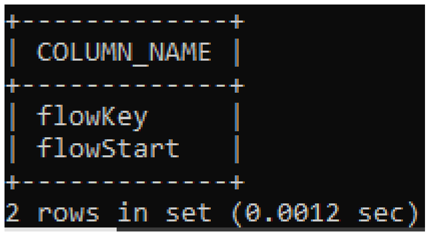
Expected Output:



```
+-------------+
| COLUMN_NAME |
+-------------+
| flowKey     |
| flowStart   |
+-------------+
2 rows in set (0.0010 sec)
```

SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA = "ece356_project" AND TABLE_NAME = "ForwardFlowStatistics" AND COLUMN_KEY = "PRI";

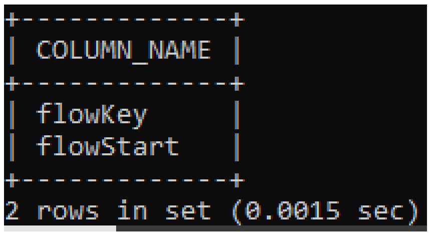
Expected Output:



```
+-------------+
| COLUMN_NAME |
+-------------+
| flowKey     |
| flowStart   |
+-------------+
2 rows in set (0.0012 sec)
```

SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA = "ece356_project" AND TABLE_NAME = "BackwardFlowStatistics" AND COLUMN_KEY = "PRI";

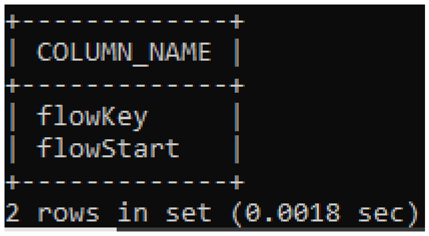
Expected Output:



```
+-------------+
| COLUMN_NAME |
+-------------+
| flowKey     |
| flowStart   |
+-------------+
2 rows in set (0.0015 sec)
```

SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA = "ece356_project" AND TABLE_NAME = "TotalFlowStatistics" AND COLUMN_KEY = "PRI";

Expected Output:

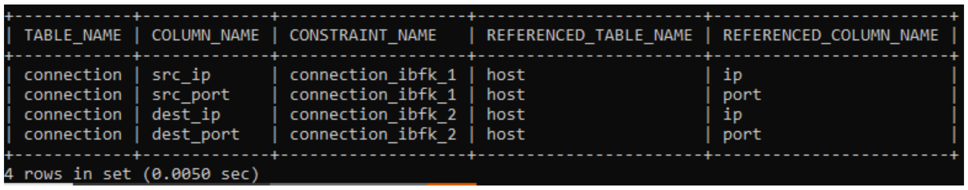


```
+-------------+
| COLUMN_NAME |
+-------------+
| flowKey     |
| flowStart   |
+-------------+
2 rows in set (0.0018 sec)
```

For Foreign Keys:

SELECT TABLE_NAME,COLUMN_NAME,CONSTRAINT_NAME, REFERENCED_TABLE_NAME,REFERENCED_COLUMN_NAME FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE WHERE REFERENCED_TABLE_SCHEMA = "ece356_project" AND REFERENCED_TABLE_NAME = "Host";

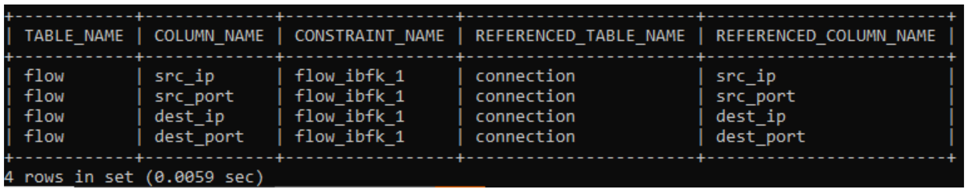
Expected Output:



```
+------------+-------------+-------------------+-----------------------+------------------------+
| TABLE_NAME | COLUMN_NAME | CONSTRAINT_NAME   | REFERENCED_TABLE_NAME | REFERENCED_COLUMN_NAME |
+------------+-------------+-------------------+-----------------------+------------------------+
| connection | src_ip      | connection_ibfk_1 | host                  | ip                     |
| connection | src_port    | connection_ibfk_1 | host                  | port                   |
| connection | dest_ip     | connection_ibfk_2 | host                  | ip                     |
| connection | dest_port   | connection_ibfk_2 | host                  | port                   |
+------------+-------------+-------------------+-----------------------+------------------------+
4 rows in set (0.0050 sec)
```

SELECT TABLE_NAME,COLUMN_NAME,CONSTRAINT_NAME, REFERENCED_TABLE_NAME,REFERENCED_COLUMN_NAME FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE WHERE REFERENCED_TABLE_SCHEMA = "ece356_project" AND REFERENCED_TABLE_NAME = "Connection";

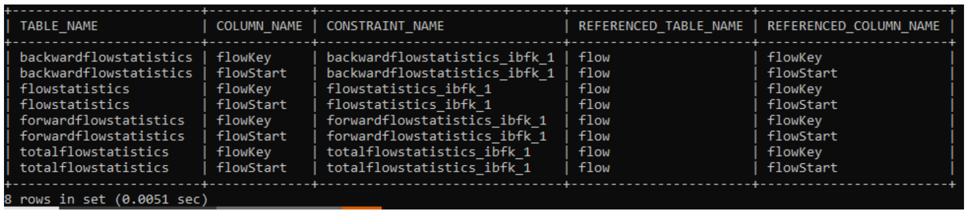
Expected Output:



```
+------------+-------------+-----------------+-----------------------+------------------------+
| TABLE_NAME | COLUMN_NAME | CONSTRAINT_NAME | REFERENCED_TABLE_NAME | REFERENCED_COLUMN_NAME |
+------------+-------------+-----------------+-----------------------+------------------------+
| flow       | src_ip      | flow_ibfk_1     | connection            | src_ip                 |
| flow       | src_port    | flow_ibfk_1     | connection            | src_port               |
| flow       | dest_ip     | flow_ibfk_1     | connection            | dest_ip                |
| flow       | dest_port   | flow_ibfk_1     | connection            | dest_port              |
+------------+-------------+-----------------+-----------------------+------------------------+
4 rows in set (0.0059 sec)
```

SELECT TABLE_NAME,COLUMN_NAME,CONSTRAINT_NAME, REFERENCED_TABLE_NAME,REFERENCED_COLUMN_NAME FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE WHERE REFERENCED_TABLE_SCHEMA = "ece356_project" AND REFERENCED_TABLE_NAME = "Flow";

Expected Output:

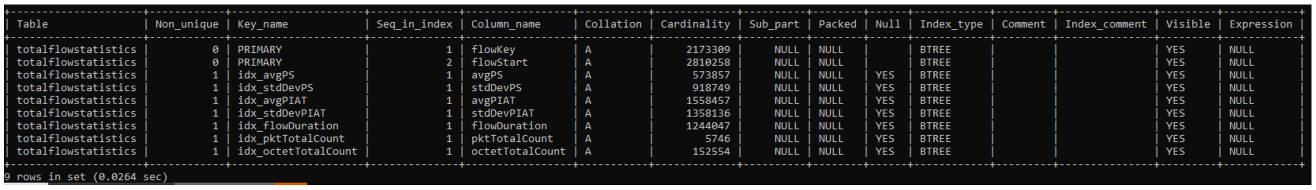


```
+------------------------+-------------+------------------------------+-----------------------+------------------------+
| TABLE_NAME             | COLUMN_NAME | CONSTRAINT_NAME              | REFERENCED_TABLE_NAME | REFERENCED_COLUMN_NAME |
+------------------------+-------------+------------------------------+-----------------------+------------------------+
| backwardflowstatistics | flowKey     | backwardflowstatistics_ibfk_1 | flow                 | flowKey                |
| backwardflowstatistics | flowStart   | backwardflowstatistics_ibfk_1 | flow                 | flowStart              |
| flowstatistics         | flowKey     | flowstatistics_ibfk_1        | flow                  | flowKey                |
| flowstatistics         | flowStart   | flowstatistics_ibfk_1        | flow                  | flowStart              |
| forwardflowstatistics  | flowKey     | forwardflowstatistics_ibfk_1 | flow                  | flowKey                |
| forwardflowstatistics  | flowStart   | forwardflowstatistics_ibfk_1 | flow                  | flowStart              |
| totalflowstatistics    | flowKey     | totalflowstatistics_ibfk_1   | flow                  | flowKey                |
| totalflowstatistics    | flowStart   | totalflowstatistics_ibfk_1   | flow                  | flowStart              |
+------------------------+-------------+------------------------------+-----------------------+------------------------+
8 rows in set (0.0051 sec)
```

For Indexes:

SHOW INDEXES from TotalFlowStatistics IN ece356_project;

Expected Output:



```
+---------------------+------------+---------------------+--------------+-----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table               | Non_unique | Key_name            | Seq_in_index | Column_name     | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+---------------------+------------+---------------------+--------------+-----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| totalflowstatistics |          0 | PRIMARY             |            1 | flowKey         | A         |     2173309 |     NULL | NULL   |      | BTREE      |         |               | YES     | NULL       |
| totalflowstatistics |          0 | PRIMARY             |            2 | flowStart       | A         |     2810258 |     NULL | NULL   |      | BTREE      |         |               | YES     | NULL       |
| totalflowstatistics |          1 | idx_avgPS           |            1 | avgPS           | A         |      573857 |     NULL | NULL   | YES  | BTREE      |         |               | YES     | NULL       |
| totalflowstatistics |          1 | idx_stdDevPS        |            1 | stdDevPS        | A         |      918749 |     NULL | NULL   | YES  | BTREE      |         |               | YES     | NULL       |
| totalflowstatistics |          1 | idx_avgPIAT         |            1 | avgPIAT         | A         |     1558457 |     NULL | NULL   | YES  | BTREE      |         |               | YES     | NULL       |
| totalflowstatistics |          1 | idx_stdDevPIAT      |            1 | stdDevPIAT      | A         |     1358136 |     NULL | NULL   | YES  | BTREE      |         |               | YES     | NULL       |
| totalflowstatistics |          1 | idx_flowDuration    |            1 | flowDuration    | A         |     1244047 |     NULL | NULL   | YES  | BTREE      |         |               | YES     | NULL       |
| totalflowstatistics |          1 | idx_pktTotalCount   |            1 | pktTotalCount   | A         |        5746 |     NULL | NULL   | YES  | BTREE      |         |               | YES     | NULL       |
| totalflowstatistics |          1 | idx_octetTotalCount |            1 | octetTotalCount | A         |      152554 |     NULL | NULL   | YES  | BTREE      |         |               | YES     | NULL       |
+---------------------+------------+---------------------+--------------+-----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
9 rows in set (0.0264 sec)
```

From the above expected outputs, it is shown that the results match what we created for our relational schema.

The test cases for our CLI check the overall functionality of the CLI while modifying and returning values found in the tables in our database correctly from the input given.

Test Cases for Create:

The following test cases are for the create functionality in our CLI where we are attempting to insert into tables for the given flags.

1. create -host -ip='127.0.0.1' -port=5050

This will insert an IP and port value into our Host table.

Expected Output:

```
INSERT IGNORE INTO Host VALUES('127.0.0.1',5050);

Successful Host Creation
```

2. create -host -port=2020 -ip=’127.0.0.2’

This will insert an IP and port value into our Host table. The order in which the flags are given in the input does not matter.

Expected Output:

```
INSERT IGNORE INTO Host VALUES('127.0.0.2',2020);

Successful Host Creation
```

3. create -connection -src_ip='127.0.0.1' -src_port=5050 -dest_ip='127.0.0.2' -dest_port=2020

This will insert a source and destination IP address and port into our Connection table. This will also try to insert new IP addresses and port for both of the source and destination given but the query will be ignored due to duplicates found in Host table.

Expected Output:

```
INSERT IGNORE INTO Host VALUES('127.0.0.1',5050);

Successful Host Creation

INSERT IGNORE INTO Host VALUES('127.0.0.2',2020);

Successful Host Creation

INSERT IGNORE INTO Connection VALUES('127.0.0.1',5050,'127.0.0.2',2020);

Successful Connection Creation
```

4. create -connection -src_ip='111.111.111.111' -src_port=1111 -dest_ip='111.111.111.112' -dest_port=1112

This will have the same kind of output as shown for 3 but we will create new rows that contains IP addresses of source and destination in our Host table since they are new values.

Expected Output:

```
INSERT IGNORE INTO Host VALUES('111.111.111.111',1111);

Successful Host Creation

INSERT IGNORE INTO Host VALUES('111.111.111.112',1112);

Successful Host Creation

INSERT IGNORE INTO Connection VALUES('111.111.111.111',1111,'111.111.111.112',1112);

Successful Connection Creation
```

5. create -flow -flowKey='1abc' -flowStart=1111 -src_ip='111.111.111.111' -src_port=1111 -dest_ip='111.111.111.112' -dest_port=1112 -protocolType=17

This command will insert a new row in Flow, FlowStatistics, ForwardFlowStatistics, BackwardFlowStatistics and TotalFlowStatistics table with the given values shown above. This should ignore Host and Connection creation of new rows because the IP addresses for both source and destination already exist in the table. The flags that match the attributes in the tables will have their values added while the rest of the values for the insert query will be NULL.

Expected Output:

```
INSERT IGNORE INTO Host VALUES('111.111.111.111',1111);

Successful Host Creation

INSERT IGNORE INTO Host VALUES('111.111.111.112',1112);
```

```
Successful Host Creation

INSERT IGNORE INTO Connection VALUES('111.111.111.111',1111,'111.111.111.112',1112);

Successful Connection Creation

INSERT INTO Flow(flowKey, flowStart, src_ip, src_port, dest_ip, dest_port, protocolType,
src_ip_numeric) VALUES ('1abc',1111,'111.111.111.111',1111,'111.111.111.112',1112,17,NULL);
INSERT INTO FlowStatistics(flowKey, flowStart, flowEndReason, category, app_protocol, web_service)
VALUES ('1abc',1111,NULL,NULL,NULL,NULL);
INSERT INTO ForwardFlowStatistics(flowKey, flowStart, f_minPS, f_maxPS, f_avgPS, f_stdDevPS,
f_minPIAT, f_maxPIAT, f_avgPIAT, f_stdDevPIAT, f_flowStart, f_flowEnd, f_flowDuration,
f_pktTotalCount, f_octetTotalCount) VALUES
('1abc',1111,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);
INSERT INTO BackwardFlowStatistics(flowKey, flowStart, b_minPS, b_maxPS, b_avgPS, b_stdDevPS,
b_minPIAT, b_maxPIAT, b_avgPIAT, b_stdDevPIAT, b_flowStart, b_flowEnd, b_flowDuration,
b_pktTotalCount, b_octetTotalCount) VALUES
('1abc',1111,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);
INSERT INTO TotalFlowStatistics(flowKey, flowStart, minPS, maxPS, avgPS, stdDevPS, minPIAT,
maxPIAT, avgPIAT, stdDevPIAT, flowEnd, flowDuration, pktTotalCount, octetTotalCount) VALUES
('1abc',1111,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);

Successful Flow Creation
```

6. create -flow -flowKey='1abcd' -flowStart=1111 -src_ip='111.111.111.113' -src_port=1113 -dest_ip='111.111.111.114' -dest_port=1114 -protocolType=6

This command will insert a new row in Flow, FlowStatistics, ForwardFlowStatistics, BackwardFlowStatistics and TotalFlowStatistics table with the given values shown above. This creates new Host and Connection rows in their respective tables because there are new source and destination IP addresses and port. The flags that match the attributes in the tables will have their values added while the rest of the values for the insert query will be NULL.

Expected Output:

```
INSERT IGNORE INTO Host VALUES('111.111.111.113',1113);

Successful Host Creation

INSERT IGNORE INTO Host VALUES('111.111.111.114',1114);

Successful Host Creation

INSERT IGNORE INTO Connection VALUES('111.111.111.113',1113,'111.111.111.114',1114);

Successful Connection Creation
```

```
INSERT INTO Flow(flowKey, flowStart, src_ip, src_port, dest_ip, dest_port, protocolType,
src_ip_numeric) VALUES ('1abcd',1111,'111.111.111.113',1113,'111.111.111.114',1114,6,NULL);
INSERT INTO FlowStatistics(flowKey, flowStart, flowEndReason, category, app_protocol, web_service)
VALUES ('1abcd',1111,NULL,NULL,NULL,NULL);
INSERT INTO ForwardFlowStatistics(flowKey, flowStart, f_minPS, f_maxPS, f_avgPS, f_stdDevPS,
f_minPIAT, f_maxPIAT, f_avgPIAT, f_stdDevPIAT, f_flowStart, f_flowEnd, f_flowDuration,
f_pktTotalCount, f_octetTotalCount) VALUES
('1abcd',1111,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);
INSERT INTO BackwardFlowStatistics(flowKey, flowStart, b_minPS, b_maxPS, b_avgPS, b_stdDevPS,
b_minPIAT, b_maxPIAT, b_avgPIAT, b_stdDevPIAT, b_flowStart, b_flowEnd, b_flowDuration,
b_pktTotalCount, b_octetTotalCount) VALUES
('1abcd',1111,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);
INSERT INTO TotalFlowStatistics(flowKey, flowStart, minPS, maxPS, avgPS, stdDevPS, minPIAT,
maxPIAT, avgPIAT, stdDevPIAT, flowEnd, flowDuration, pktTotalCount, octetTotalCount) VALUES
('1abcd',1111,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);

Successful Flow Creation
```

7. create -flow -flowKey='1abce' -flowStart=1111 -src_ip='111.111.111.113' -src_port=1113 -dest_ip='111.111.111.114' -dest_port=1114 -protocolType=17 -app_protocol='HTML' -minPS=1 -maxPS=10 -f_avgPS=1212 -f_minPIAT=12013

This command will insert a new row in Flow, FlowStatistics, ForwardFlowStatistics, BackwardFlowStatistics and TotalFlowStatistics table with the given values shown above. The flags that match the attributes in the tables will have their values added while the rest of the values for the insert query will be NULL.

Expected Output:

```
INSERT IGNORE INTO Host VALUES('111.111.111.113',1113);

Successful Host Creation

INSERT IGNORE INTO Host VALUES('111.111.111.114',1114);

Successful Host Creation

INSERT IGNORE INTO Connection VALUES('111.111.111.113',1113,'111.111.111.114',1114);

Successful Connection Creation

INSERT INTO Flow(flowKey, flowStart, src_ip, src_port, dest_ip, dest_port, protocolType,
src_ip_numeric) VALUES ('1abce',1111,'111.111.111.113',1113,'111.111.111.114',1114,17,NULL);
INSERT INTO FlowStatistics(flowKey, flowStart, flowEndReason, category, app_protocol, web_service)
VALUES ('1abce',1111,NULL,NULL,HTML,NULL);
Sql Error
INSERT INTO ForwardFlowStatistics(flowKey, flowStart, f_minPS, f_maxPS, f_avgPS, f_stdDevPS,
f_minPIAT, f_maxPIAT, f_avgPIAT, f_stdDevPIAT, f_flowStart, f_flowEnd, f_flowDuration,
f_pktTotalCount, f_octetTotalCount) VALUES
('1abce',1111,NULL,NULL,1212,NULL,12013,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);
INSERT INTO BackwardFlowStatistics(flowKey, flowStart, b_minPS, b_maxPS, b_avgPS, b_stdDevPS,
b_minPIAT, b_maxPIAT, b_avgPIAT, b_stdDevPIAT, b_flowStart, b_flowEnd, b_flowDuration,
b_pktTotalCount, b_octetTotalCount) VALUES
('1abce',1111,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);
INSERT INTO TotalFlowStatistics(flowKey, flowStart, minPS, maxPS, avgPS, stdDevPS, minPIAT,
maxPIAT, avgPIAT, stdDevPIAT, flowEnd, flowDuration, pktTotalCount, octetTotalCount) VALUES
('1abce',1111,1,10,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);

Successful Flow Creation
```

8. create -flow -flowKey='2abc' -flowStart=2222 -src_ip='111.111.111.115' -src_port=1115 -dest_ip='111.111.111.116' -dest_port=1116 -protocolType=6 -app_protocol='HTML' -minPS=2 -maxPS=20 -f_avgPS=1313 -f_minPIAT=12014

This command is to test and check whether each attribute value is added onto their respective table for the insert query. It will insert a new row in Flow, FlowStatistics, ForwardFlowStatistics, BackwardFlowStatistics and TotalFlowStatistics table with the given values shown above. The flags that match the attributes in the tables will have their values added while the rest of the values for the insert query will be NULL.

Expected Output:

```
INSERT IGNORE INTO Host VALUES('111.111.111.115',1115);

Successful Host Creation

INSERT IGNORE INTO Host VALUES('111.111.111.116',1116);

Successful Host Creation

INSERT IGNORE INTO Connection VALUES('111.111.111.115',1115,'111.111.111.116',1116);

Successful Connection Creation

INSERT INTO Flow(flowKey, flowStart, src_ip, src_port, dest_ip, dest_port, protocolType,
src_ip_numeric) VALUES ('2abc',2222,'111.111.111.115',1115,'111.111.111.116',1116,6,NULL);
INSERT INTO FlowStatistics(flowKey, flowStart, flowEndReason, category, app_protocol, web_service)
VALUES ('2abc',2222,NULL,NULL,'HTML',NULL);
INSERT INTO ForwardFlowStatistics(flowKey, flowStart, f_minPS, f_maxPS, f_avgPS, f_stdDevPS,
f_minPIAT, f_maxPIAT, f_avgPIAT, f_stdDevPIAT, f_flowStart, f_flowEnd, f_flowDuration,
f_pktTotalCount, f_octetTotalCount) VALUES
('2abc',2222,NULL,NULL,1313,NULL,12014,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);
INSERT INTO BackwardFlowStatistics(flowKey, flowStart, b_minPS, b_maxPS, b_avgPS, b_stdDevPS,
b_minPIAT, b_maxPIAT, b_avgPIAT, b_stdDevPIAT, b_flowStart, b_flowEnd, b_flowDuration,
b_pktTotalCount, b_octetTotalCount) VALUES
('2abc',2222,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);
```

```
INSERT INTO TotalFlowStatistics(flowKey, flowStart, minPS, maxPS, avgPS, stdDevPS, minPIAT,
maxPIAT, avgPIAT, stdDevPIAT, flowEnd, flowDuration, pktTotalCount, octetTotalCount) VALUES
('2abc',2222,2,20,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);

Successful Flow Creation
```

Test Cases for Read:

The following test cases are for the read functionality in our CLI where we are attempting to read column values for the given flags.

1. read -s -all

This will output all of the values found in each attribute for Flow, FlowStatistic, ForwardFlowStatistic, BackwardFlowStatistic and TotalFlowStatistic after inner joining all of the tables.

Expected Output:

```
The expected output is the value of every column in every table.
```

2. read –s -all -limit=1

This will output 1 row with values found in each attribute for Flow, FlowStatistic, ForwardFlowStatistic, BackwardFlowStatistic and TotalFlowStatistic after inner joining all of the tables.

Expected Output:

```
SELECT * FROM Flow INNER JOIN FlowStatistics USING (flowKey, flowStart) INNER JOIN
ForwardFlowStatistics USING (flowKey, flowStart) INNER JOIN TotalFlowStatistics USING (flowKey,
flowStart) INNER JOIN BackwardFlowStatistics USING (flowKey, flowStart)  limit 1;


[('0000015199dd36a958566b7ec0c15d76', Decimal('1556124654.154530000000000000000000'),
3232267714, '192.168.125.194', 64080, '172.16.255.200', 53, 17, 2, 'Network', 'Unknown', 'DNS', 65,
65, Decimal('65.0000000000000000000000000'), Decimal('0E-25'), Decimal('0E-25'), Decimal('0E-25'),
Decimal('0E-25'), Decimal('0E-25'), Decimal('1556124654.154530000000000000000000'),
Decimal('1556124654.154530000000000000000000'), Decimal('0E-25'), 1, 65, 65, 132,
Decimal('98.5000000000000000000000000'), Decimal('33.5000000000000000000000000'),
Decimal('0.0005338191986083980000000'), Decimal('0.0005338191986083980000000'),
Decimal('0.0005338191986083980000000'), Decimal('0E-25'),
Decimal('1556124654.155070000000000000000000'), Decimal('0.0005338191986083980000000'),
2, 197, 132, 132, Decimal('132.0000000000000000000000000'), Decimal('0E-25'), Decimal('0E-25'),
Decimal('0E-25'), Decimal('0E-25'), Decimal('0E-25'),
Decimal('1556124654.155070000000000000000000'),
Decimal('1556124654.155070000000000000000000'), Decimal('0E-25'), 1, 132)]
```

3. read -s -all -w -na -flowKey='1abc'

This command checks all of the tables where flowKey is 1abc and then outputs the values of each attribute found in the tables for our schema for this particular flowKey.

Expected Output:

```
SELECT * FROM Flow INNER JOIN FlowStatistics USING (flowKey, flowStart) INNER JOIN
ForwardFlowStatistics USING (flowKey, flowStart) INNER JOIN TotalFlowStatistics USING (flowKey,
flowStart) INNER JOIN BackwardFlowStatistics USING (flowKey, flowStart)  WHERE flowKey='1abc';


[('1abc', Decimal('1111.0000000000000000000000000'), None, '111.111.111.111', 1111,
'111.111.111.112', 1112, 17, None, None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None, None, None, None, None, None, None,
None)]
```

4. read -s -minPS -limit=10

This command returns 10 values of minPS.

Expected Output:

```
SELECT minPS FROM Flow INNER JOIN FlowStatistics USING (flowKey, flowStart) INNER JOIN
ForwardFlowStatistics USING (flowKey, flowStart) INNER JOIN TotalFlowStatistics USING (flowKey,
flowStart) INNER JOIN BackwardFlowStatistics USING (flowKey, flowStart)  limit 10;


[(65,), (69,), (40,), (61,), (66,), (66,), (40,), (40,), (52,), (52,)]


Successfully read values from tables.
```

5. read -s -minPS -maxPS -w -and -minPS>10 -avgPS>100 -limit=10

This command will return values for minPS and maxPS where minPS>10 and avgPS>100 in the table and limit the number of values outputted for minPS and maxPS to be 10.

Expected Output:

```
SELECT minPS,maxPS FROM Flow INNER JOIN FlowStatistics USING (flowKey, flowStart) INNER JOIN
ForwardFlowStatistics USING (flowKey, flowStart) INNER JOIN TotalFlowStatistics USING (flowKey,
flowStart) INNER JOIN BackwardFlowStatistics USING (flowKey, flowStart)  WHERE minPS>10 AND
avgPS>100 limit 10;


[(69, 169), (40, 2195), (66, 149), (66, 149), (40, 625), (40, 3827), (52, 2515), (61, 246), (77, 137), (40,
2888)]
```

```
Successfully read values from tables.
```

6. read -s -pktTotalCount -octetTotalCount -w -or -pktTotalCount>100 -octetTotalCount>1000 -limit=5

This command will return values for pktTotalCount and octetTotalCount where pktTotalCount>100 or octetTotalCount>1000 in the table and limit the number of values outputted for pktTotalCount and octetTotalCount to be 5.

Expected Output:

```
SELECT pktTotalCount,octetTotalCount FROM Flow INNER JOIN FlowStatistics USING (flowKey,
flowStart) INNER JOIN ForwardFlowStatistics USING (flowKey, flowStart) INNER JOIN
TotalFlowStatistics USING (flowKey, flowStart) INNER JOIN BackwardFlowStatistics USING (flowKey,
flowStart)  WHERE pktTotalCount>100 OR  octetTotalCount>1000 limit 5;


[(47, 38596), (38, 4551), (100, 56826), (12, 3958), (29, 8215)]


Successfully read values from tables.
```

Test Cases for Update:

The following test cases are for the update functionality in our CLI where we are attempting to update column values for the given flags for the tables in our database.

1. update -flowKey='1abc' -flowStart=1111 -src_ip_numeric=1111111111

The above command will update src_ip_numeric with the given flowKey and flowStart in Flow table.

Expected Output:

```
UPDATE Flow SET src_ip_numeric=1111111111 WHERE flowKey='1abc'AND flowStart=1111;


Successfully updated rows.
```

2. update -flowKey='1abc' -flowStart=1111 -maxPS=1000 -avgPS=10000

This will update maxPS to be 1000 and avgPS to be 10000 for the given flowKey and flowStart values inside TotalFlowStatistics table.

Expected Output:

```
UPDATE TotalFlowStatistics SET maxPS=1000,avgPS=10000 WHERE flowKey='1abc'AND
flowStart=1111;
```

```

Successfully updated rows.
```

3. update -flowKey='2abc' -flowStart=2222 -f_flowStart=20104 -b_stdDevPIAT=1.101010 -protocolType=17

The above command updates values in 3 tables. The attribute protocolType is in Flow, f_flowStart is in ForwardFlowStatistics and b_stdDevPIAT is in BackwardFlowStatistics. The three tables will have new values for those attributes where the flowKey is 2abc and flowStart is 2222.

Expected Output:

```
UPDATE Flow SET protocolType=17 WHERE flowKey='2abc'AND flowStart=2222;
UPDATE ForwardFlowStatistics SET f_flowStart=20104 WHERE flowKey='2abc'AND flowStart=2222;
UPDATE BackwardFlowStatistics SET b_stdDevPIAT=1.101010 WHERE flowKey='2abc'AND
flowStart=2222;


Successfully updated rows.
```

Test Cases for Delete:

The following test cases are for the delete functionality in our CLI where we are attempting to delete certain rows accoring to the column values of flowKey and flowStart.

1. delete -flowKey='1abc' -flowStart=1111

The above command deletes all rows found in Flow, FlowStatistics, ForwardFlowStatistics, BackwardFlowStatistics and TotalFlowStatistics where flowKey is 1abc and flowStart is 1111.

Expected Output:

```
DELETE FROM Flow WHERE flowKey='1abc' AND flowStart=1111;


Successfully deleted rows.
```

2. delete -flowKey='2abc' -flowStart=2222

The above command deletes all rows found in Flow, FlowStatistics, ForwardFlowStatistics, BackwardFlowStatistics and TotalFlowStatistics where flowKey is 2abc and flowStart is 2222.

Expected Output:

```
DELETE FROM Flow WHERE flowKey='2abc' AND flowStart=2222;


Successfully deleted rows.
```

3. read -s -minPS -maxPS -w -and -flowKey='1abc' -flowStart=1111

This test case is to check for the previous deletion if it was successful. It will return an empty row.

Expected Output:

```
SELECT minPS,maxPS FROM Flow INNER JOIN FlowStatistics USING (flowKey, flowStart) INNER JOIN
ForwardFlowStatistics USING (flowKey, flowStart) INNER JOIN TotalFlowStatistics USING (flowKey,
flowStart) INNER JOIN BackwardFlowStatistics USING (flowKey, flowStart)  WHERE flowKey='1abc'
AND flowStart=1111;


[]


Successfully read values from tables.
```

The expected output of all our CLI inputs is shown inside the boxes. For all of the test cases above, the number of rows that were taken from our csv is 2,704,839. However, the output may be different due to how many rows from the csv are loaded into the tables. This was noticed when our CLI was run in eceubuntu with marmoset04 as the mysql server. The values returned or modified from the queries are different due to the number of rows missing from marmoset04 server db356 database.

5. README

1. Install mysql-connector-python by running “pip install mysql-connector-python"
2. Run updated_schema.sql in database
    a. Modify number of rows inserted from csv on line 66 if you wish to do so. Total number of rows in csv is 2,704,839. Line 66 states how many rows to ignore when loading data into the table such as “IGNORE 2,700,000 ROWS” will create a table with 4,839 rows,
3. Open cli.py and modify the following lines 171, 342, 565 and 604 with correct values for host, user, passwd and database. If host requires port then add an additional property ‘ port=“yourPortNumber“ ’. If you want to connect to marmoset04 then change host to host=“marmoset04.shoshin.uwaterloo.ca”, user=“userID”, password=“yourPassword” and database=“databaseName”.
4. To run CLI, type ‘python3 cli.py’ in terminal
5. Program will ask for credentials. There are 2 users with authentication given to run CLI
    a. Admin: user=admin, password=admin (User is able to use delete command)
    b. Client: user=client, password=client (User is unable to use delete command)
6. Run commands or type ‘help’ inside CLI for further instructions (refer to testcases for valid commands).
7. Type ‘exit’ to exit out of CLI