MySQL Enhancement Proposal

EECS 4314: Advanced Software Engineering
Tabs vs. Spaces
December 4, 2017

Authors

Kevin Arindaeng

Anton Sitkovets

Glib Sitiugin

Nisha Sharma

Varsha Ragavendran

Ayman Abualsunun

Davood Anbarnam

Table of Contents

List of Figures

List of Tables

1.0 Abstract

This report presents an overview of a proposed enhancement to MySQL, which is the implementation of ranged data types. A detailed explanation of the enhancement, its motivation, and the benefits is presented. A comparison on several approaches to implement this feature is provided. The possible impact on the architectural subsystems, as well as the risks and limitations is shown. Lastly, a conclusion and lessons learned is presented based on the proposed enhancement.

2.0 Introduction and Overview

This report presents the proposal for enhancing MySQL 8.0.2 with a new feature called range types. Specifically, this report will outline the significance of this proposed feature, comparing the advantages and disadvantages of integrating this feature to the current MySQL. Range types allows the users to set a defined range for every column when creating a table. This simple features provides the solution to avoid having a table using multiple columns to define the start and end values of a range interval, thereby reducing the number of columns in a table.

The report begins with an analysis on the proposed feature, range types, explaining the motivation behind the feature. Followed by proposing a few approaches for realizing this new feature when integrating with the current MySQL. Each approach is examined further in detail, considering the pros and cons, and finally narrowing down to the most efficient approach. Next, at a high level, architectural changes that could occur when integrating this feature to the current MySQL are described, and potential impacts on existing subsystems due to these changes are identified. Plans for testing this new feature accounts for performance, scalability, and several other factors to be considered. As well, the benefits and values that arise from this enhancement are proposed. Finally, the potential risks and limitations of range types are examined in detail.

To close out this section of the report, a glossary for various terms/phrases used throughout the document as well as any naming conventions used are presented. The report concludes with potential advantages of integrating range types with MySQL, and lessons learned from putting together this document.

3.0 Proposed Enhancement and Motivation

3.1 Enhancement:

The proposed enhancement is to introduce a new data type to MySQL that represents a range of values, called a range type. This can be useful in many cases. One example application would be using range types as a way to make a room reservation system at a hotel. This would allow the staff to quickly find which rooms are unavailable at certain times. Another example application would be creating an employee scheduling system. This could be useful to managers to let them quickly see who is or who isn't working at certain shifts. One last application would be buying items within a certain range. For example, if you are buying a car to adhere to a budget, you can search for vehicles within that price range.

3.2 Motivation:

Whilst doing a comparison of different features between SQL implementations, it was found that range types was only implemented by PostgreSQL, and that MySQL and all other SQL implementations did not have this feature as seen in Figure 1.

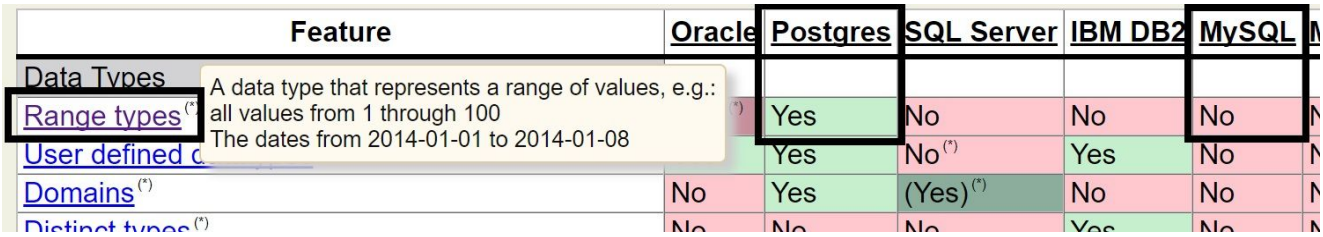

Figure 1. SQL Feature Comparison [1]

This intrigued the interest in proposing this enhancement, as it would be useful to know why PostgreSQL was the only database that implemented this feature, as well as seeing if it could be done on MySQL, and whether any problems would arise when approaching this enhancement.

3.3 Enhancement Usage:

Table 1 describes four examples of common numeric and date/time data types in MySQL, which are integers, fixed-point values, floating point values, and dates/times.

	Integer	Fixed-Point	Floating Point	Date and Time
	`INT`	`DECIMAL(5,2)`	`FLOAT(5,3)`	`DATETIME`
Example	`123456`	`999.99`	`-999.999`	`1000-01-01 00:00:00`

Table 1. MySQL Numeric and Date/Time Datatypes [2]

The proposed range type syntax in MySQL for the four examples discussed above can be seen in Table 2. The range type would extend the already existing numeric and date/time data types in MySQL. It would simply append “RANGE” to the end of the original values, and insert values within a range of that specific data type. The square brackets would be defined as inclusivity (contains the bounded value), and the regular brackets would mean exclusivity (does not contain the bounded value).

	Integer	Fixed-Point	Floating Point	Date and Time
	`INT`**`RANGE`**	`DECIMAL`**`RANGE`** `(5,2)`	`FLOAT`**`RANGE`** `(5,3)`	`DATETIME`**`RANGE`**
Example	`[100, 200]`	`(500.01, 900.99)`	`[-123.45, 678.91)`	`[1000-01-01 00:00:00, 2017-11-29 13:00:00)`

Table 2. MySQL Equivalent Range Types from Table 1

These range types can then be used similar to regular datatypes on DDL, DML, or DQL statements. A sample DDL and DML statement in MySQL using the range type can be seen in Table 3. A sample table is created to track employee schedules on the left. On the right, a sample employee with an id of “1” is added where they work between on November 29th, 2017 from 9am to 5pm, utilizing a DATETIMERANGE type.

```
CREATE TABLE work_sched
(
      employeeId INT
     ,during DATETIMERANGE
);
```

```
INSERT INTO work_sched VALUES
(
      1,

     [2017-11-29 09:00:00,
     2017-11-29,17:00:00]
);
```

Table 3. Sample CREATE and INSERT statements with a range type

An example DQL statement can be seen in Table 4. The left query would find all employees working exactly on November 29th, 2017 from 9am to 5pm. With the previous INSERT statement done in Table 3, it would output employee 1 because that employee works exactly between those times and date.

Along with the range type, the enhancement would also include new operators to help with the usage of the range type in queries. One example operator would be "<@" which means "range is contained by". Here, the right query would find all employees working on November 29th, 2017 from 8am to 8pm. With the previous INSERT statement done in Table 3, it would output employee 1 because that employee is working from 9am to 5pm, which is between 8am to 8pm.

```
SELECT * from work_sched
where during =

'[2017-11-29 09:00:00,
2017-11-29,17:00:00]'::DATE
TIMERANGE
```

```
SELECT * from work_sched
where during <@

'[2017-11-29 08:00:00,
2017-11-29,20:00:00]'::DATETI
MERANGE
```

Table 4. Sample SELECT statements with a range type

4.0 Stakeholders

The parties affected by the proposed feature are presented below, with a rationale given regarding their inclusion.

End-User
Rationale: This feature will provide an alternative approach to defining ranges, where the current approach is to write a separate query, where this alternative will be beneficial to the user.

ORACLE
Rationale: ORACLE holds the intellectual property rights to MySQL. Any changes to MySQL must subsequently be approved.

Competitors
Rationale: As it stands, the only known DBMS to incorporate range types is Postgresql. Successful addition of this feature may force competitors, like IBM's DB2, to explore additional options of enhancement.

The Team
Rationale: The group proposing the enhancement must ultimately be responsible for its integration into MySQL.

5.0 Approaches for Realizing Range Types

5.1 First Approach - Mappings

The first approach centers around creating a mapping from a generated identifier (id) to an element in a specified range. Letting each "e*i*" being the ith element of a range and each "id*i*" being the ith element's identifier, the range can then be expressed as the set of mappings shown in Figure 2.

$$\{ id1 \mapsto e1, id2 \mapsto e2, id3 \mapsto e3, ..., idn \mapsto en \}$$

Figure 2: Approach 1: Set of Mappings

As an example, suppose specified a range containing all odd integers between 1 and 7 inclusive. The corresponding representation using the mapping approach would then be Figure 3.

$$\{ \text{“n1”} \mapsto 1, \text{“n2”} \mapsto 3, \text{“n3”} \mapsto 5, \text{“n4”} \mapsto 7 \}$$

Figure 3: Approach 1: Mapping Approach

5.2 Second Approach - Hybrid

The second approach looks to give end-users greater choice when defining a range. Assuming a specified range has a well-defined means of ordering its elements, then only a lower and upper bound need be specified. In this way, larger ranges can be more easily specified, since any element within the upper and lower bounds is defined to be an element of the range. Figure 4 shows the subsequent notation, highlighting the two necessary field that the user must provide.

$$[a, b] \triangleq \{ x \mid a <= x <= b \}$$

Figure 4: Approach 2: Hybrid Mapping

As an example, suppose a user wants to specify the range of all integers between 1 and 100000. The corresponding representation would then be [1, 100000], where the user would only have to provide the stated bounds.

In the case that a user wishes to manually specify their own range, however, this approach would allow them to do so using the techniques discussed in the first approach. The term hybrid stems from providing this alternative option to the end user.

5.3 Comparison of Approaches

The Mappings approach offers a general solution to defining range type in MySQL. By allowing users to enter the elements wanted in a range, it is possible to create unique ranges, for instance the character range (‘*’, ‘?’. ‘!’). From an implementation standpoint, such an approach is relatively simple, however there are issues when it comes to scalability. The example that highlights such shortcomings would be specifying the range of integers from 1 to 1000. This approach would require the user having to enter each element.

The solution to that scalability issue is presented in the Hybrid approach, where only an interval with lower and upper bounds would have to be specified. Additionally, greater computation and storage efficiency is achieved should the interval portion of the Hybrid approach be selected, since:

1. Only the upper and lower bounds need to be stored.
2. Checking if an element is part of the range is done through two comparisons.

As stated earlier, the user is not restricted to using the interval option of the Hybrid approach. If they wished they could proceed with the Mappings approach, since the Hybrid approach provides both choices. It is for this reason that the Hybrid approach is ultimately selected, even though its implementation is relatively more complex. That is not to say, however, that this approach is perfect. There are some inherent limitations. These will be discussed later in the document.

6.0 Impacts on Subsystems

The introduction of ranged types has a noticeable impact on the Query Processor subsystem. In order to clarify those impacts, an introduction of a new subsystem was a good fit. The new subsystem “Ranged Types” as shown in Figure 5 identifies how other subsystems now depend on it. Starting from the Query Parser, since we introduced new types, the parser no longer uses the old syntax which made Parse tree, LEX, depend on the Ranged Types as well. The Execution Engine uses Ranged types indirectly through the DDL Compiler subsystem.

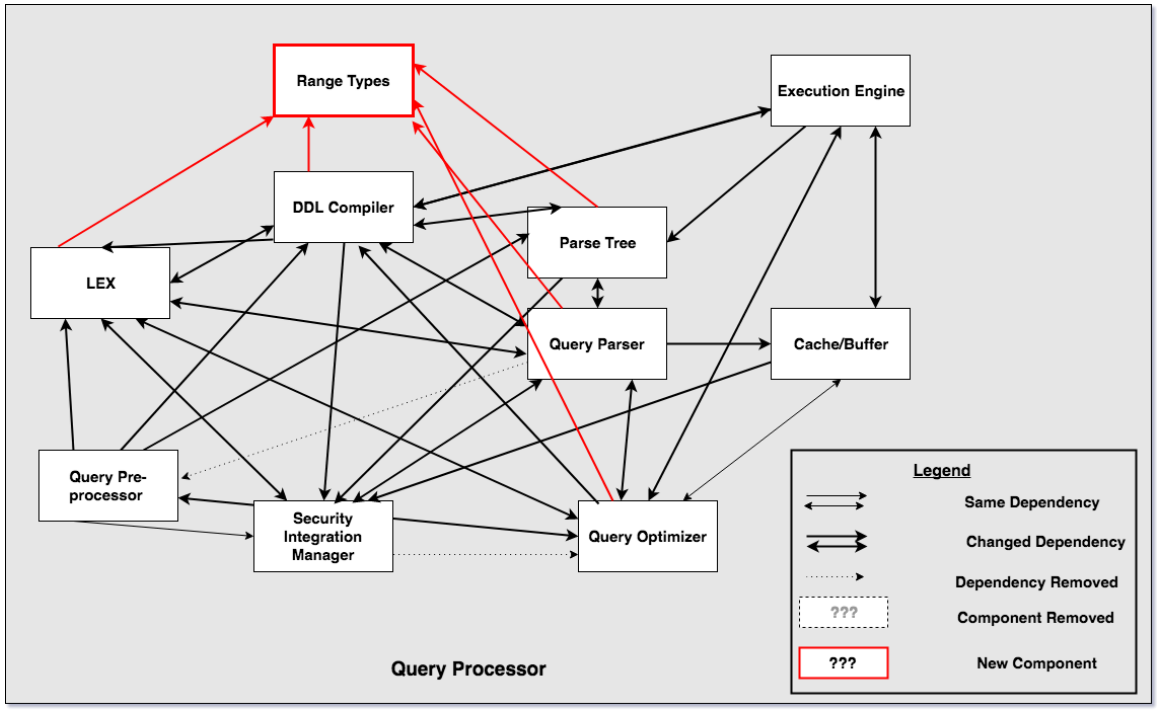

Figure 5: Impact on Architectural Subsystems

7.0 Plans for Testing

The following factors will be taken into consideration whilst testing range types integrated with MySQL:

- Performance
    - Verifying via throughput and response times to ensure these factors do not regress.[3]
- Scalability
    - Verify via Random Query Generator (RQG), increasing data loads, and increasing client connections to ensure scalability does not regress.[4]

Approaches for testing include comparing the functional and non-functional test coverages thoroughly and identifying any differences in the above-mentioned factors with range types integrated with MySQL vs. the current MySQL. This will allow to identify any impact in interactions between range types and other subsystems within MySQL, and ensure these factors do not regress.

For all testing approaches, we will be utilizing the Random Query Generator (RQG), which generates a large number of SQL queries. RQG is currently being used by the QA team in Oracle working on MySQL to perform concurrency tests.

8.0 Effects of Concurrency

The introduction of the range type feature is not expected to affect concurrency in any way. The lack of effect on concurrency can be explained by recalling concurrency strategy used by MySQL: MySQL offers a locking system with locks of different granularities (e.g. row locks, table locks). The locks act on a larger scale than individual cell and the mechanism of locking is not affected by the cell content. Thus, a range-type cell is not expected to change the locking behavior.

9.0 Value of the Enhancement

A range value can come up in many applications including price ranges, scheduling, and measurement data. They are useful in that they can represent many element values in a single range value. Often these types can be expressed in standard SQL within a “where” class for times or values in between

a range, but this requires two columns to represent a minimum and maximum value.[5]

One example where this feature is useful is when a car website has a feature to find all the cars that are within a price range. Since, cars themselves are a range since you can have the base model and then add to it to make it more expensive, this query will be for finding ranges that overlap the user's allocated price range.

Figure 6 below illustrates this example, where we need to find all the ranges that overlap the range of $13k - $15k. Looking at the SQL query below and the query using range types, it easy to see that range types make the life of the developer much easier. Without range types, it is very easy to make a mistake in how you write the query, where you might place a ">=" instead of a "<=", thus making the query incorrect. Hence, the problem with the standard SQL way of doing things, is that it is easy to make a mistake and hard to realize that this mistake has propagated into your product, while range types makes the developer's life easy.

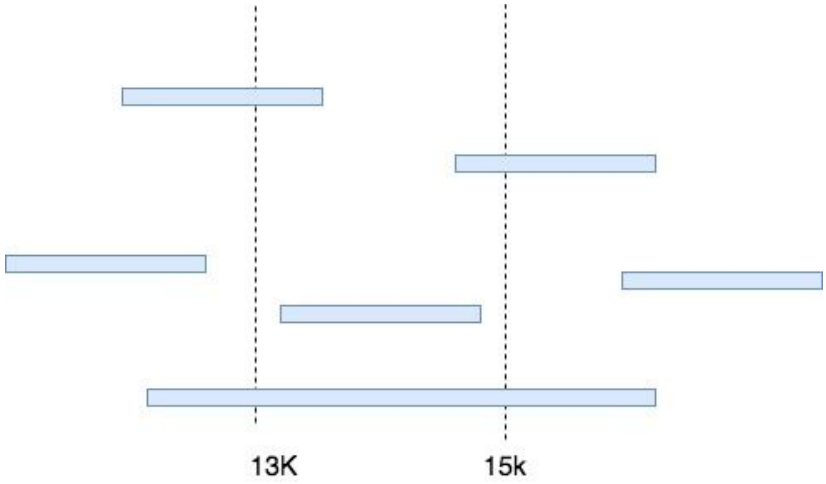

Figure 6: Overlapping Ranges

```
SELECT *
FROM cars
WHERE
  (
    cars.min_price ≤ 13000 AND
    cars.min_price ≤ 15000 AND
    cars.max_price ≥ 13000 AND
    cars.max_price ≤ 15000
  ) OR
  (
    cars.min_price ≤ 13000 AND
    cars.min_price ≤ 15000 AND
    cars.max_price ≥ 13000 AND
    cars.max_price ≥ 15000
  ) OR
  (
    cars.min_price ≥ 13000 AND
    cars.min_price ≤ 15000 AND
    cars.max_price ≥ 13000 AND
    cars.max_price ≤ 15000
  ) OR
  (
    cars.min_price ≥ 13000 AND
    cars.min_price ≤ 15000 AND
    cars.max_price ≥ 13000 AND
    cars.max_price ≥ 15000
  )
ORDER BY cars.min_price;
```

VS.

```
SELECT *
FROM cars
WHERE cars.price_range && int4range(13000, 15000, '[]')
ORDER BY lower(cars.price_range);
```

Figure 7: Query Differences With and Without Range Types

The most common use case of range types involves creating a table using a range type column. In the use case seen in Figure 8, the user has created a table called “reservation” with a column called “during”, which stores the timestamp range for when the reservation is made. The user can then query for all the rooms that are reserved within a time frame in order to better understand the amount of activity within the hotel.

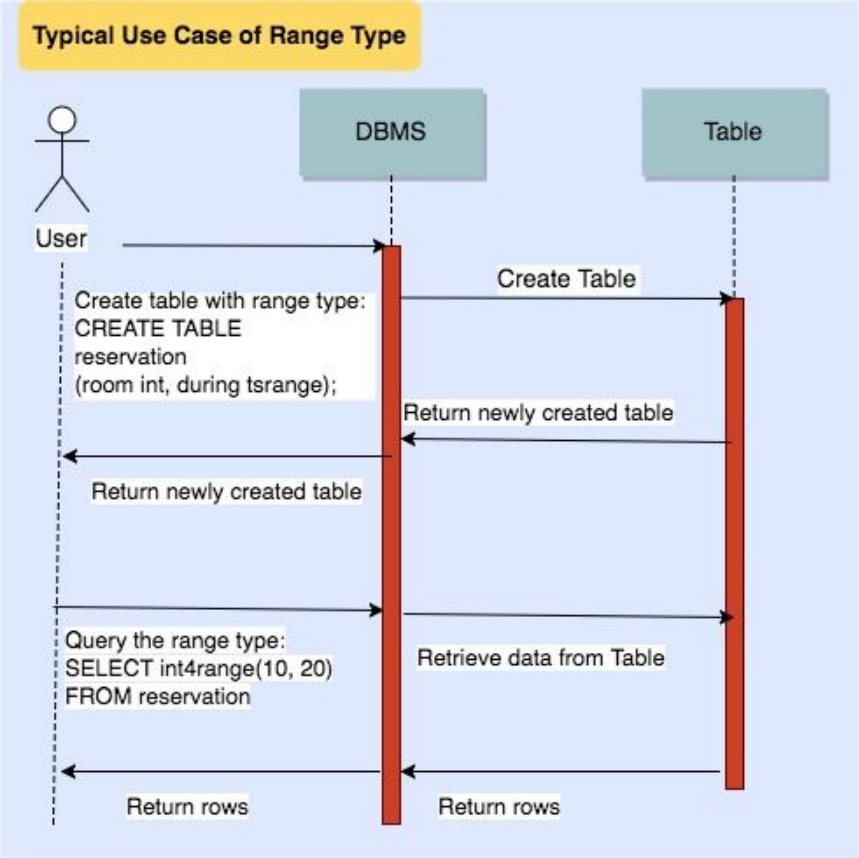

Figure 8: Typical Use Case of Range Type

10.0 Risks and Limitations

Even though we mentioned that there is no effect on concurrency; there are still potential risks to the proposed approach. Since a new subsystem is added to MySQL's architecture, some unexpected conflicts and overlappings with existing subsystems might be there. Details would be known only during testing phase.

Even though the query design becomes much more user friendly and optimal, there are also some limitations to the proposed enhancement. Our approach currently is only limited to 'Numeric' and 'DateTime' data types where is it considered to be optimal. But with other data types such as 'Strings' it might not be as optimal. Also, since the approach does not consider 'indexing' as of now, the performance speed of proposed query format would be approximately the same, if not better, as that of the existing nested loop queries. However, there is still potential to optimize the performance speed by incorporating indexing functionality in future iterations of proposed enhancement.

11.0 Data Dictionary/Glossary

Amazon RDS: A distributed relational database service by Amazon Web Services (AWS).
API: A set of subroutine definitions, protocols, and tools for building application software.
CRUD: Create, read, update, and delete, the four basic functions of persistent storage.
DB2: Database server products developed by IBM. These products all support the relational model, but in recent years some products have been extended to support object-relational features and non-relational structures like JSON and XML.
DBMS: Database-management system is a computer-software application that interacts with end-users, other applications, and the database itself to capture and analyze data. A general-purpose DBMS allows the definition, creation, querying, update, and administration of databases.
DDL: Data definition language or data description language, a syntax similar to a computer programming language for defining data structures, especially database schemas.
IDE: Integrated Development Environment, a software application that provides comprehensive facilities to computer programmers for software development.
RDBMS: Relational database management system, a database management system (DBMS) that is based on the relational model invented by Edgar F. Codd, of IBM's San Jose Research Laboratory. As of 2017, many of the databases in widespread use are based on the relational database model.
SQL: Structured Query Language, a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS).

12.0 Naming Conventions

Amazon RDS:	Amazon Relational Database Service
API:	Application Programming Interface
CRUD:	Create, Read, Update, Delete
DB2:	IBM Database 2
DBMS:	Database Management System
DDL:	Data Definition Language
DML:	Data Manipulation Language

DQL:	Data Query Language
IDE:	Interactive Development Environment
INT:	Integer
LEX:	Lexical Analyzer
ORACLE:	Oak Ridge Automatic Computer and Logical Engine
RDBMS:	Relational Database Management System
RQG:	Random Query Generator
SQL:	The Structured Query Language

13.0 Conclusions

In conclusion, the introduction of a new feature/enhancement is not an easy task, however; with the given second adaptive approach of implementing the ranged type, it was shown that it's not impossible to design and implement such feature. It might not always be feasible to enhance a legacy code or introduce a new feature, but in some applications it would make a dramatic difference as the case with ranged types and hotel booking systems or flight scheduling. With such enhancement of a legacy code, it's only normal for other subsystems to change and adapt to the new changes, as long as that doesn't affect the concurrency of the current system. Since this report is a suggested approach, there will always be risks and limits.

14.0 Lessons Learned

When dealing with legacy code, it's really important to look at the overall architecture before adding/enhancing a feature, and maintain the main functionality of the system. That way, the overall impact on the system can be minimum. When approaching a problem, the first solution might sound the right one at first, but after brainstorming and hard thinking, a better solution could arise thus, spending more time in the problem domain can be beneficial. Without implementing indexes, our approach might be still as fast as existing approach using nested queries. Extending our approach to incorporate indexes might improve the performance. finally, The current way to find overlapping range types using MySQL is highly prone to error and difficult to debug the issue. Thus adding range types, improves the user experience for MySQL.

15.0 References

[1] "SQL Feature Comparison", *Sql-workbench.net*, 2017. [Online]. Available: http://www.sql-workbench.net/dbms_comparison.html. [Accessed: 04- Dec- 2017].

[2] "MySQL: Data Types", *Techonthenet.com*, 2017. [Online]. Available: https://www.techonthenet.com/mysql/datatypes.php. [Accessed: 04- Dec- 2017].

[3] "MySQL :: MySQL Development Cycle :: 4 Performance Testing", *Dev.mysql.com*, 2017. [Online]. Available: http://dev.mysql.com/doc/mysql-development-cycle/en/performance-testing.html. [Accessed: 04- Dec- 2017].

[4] "MySQL :: MySQL Enterprise Scalability", *Mysql.com*, 2017. [Online]. Available: http://www.mysql.com/products/enterprise/scalability.html. [Accessed: 04- Dec- 2017].

[5]J. Katz, *Range Types: Your Life Will Never Be The Same*. VenueBook, 2017. [Online]. Available: https://wiki.postgresql.org/images/7/73/Range-types-pgopen-2012.pdf. [Accessed: 04- Dec- 2017].