Database Testing and Recommendations: On-Site vs Cloud and Relational vs Non Relational

Aiden Moncavage c0000279, Ali Malik c0009252, Benjamin Peachey b9035805

Introduction

We present to you our data and recommendations as to whether our company should continue to use our established on-premises databases or whether we should consider moving to a Cloud based database. In addition to this we have also tested new types of non relational databases that do not rely on SQL to compare to our current relational database, MySQL, to as if we are going to be transferring from our on-premise database to a Cloud based database we should also consider the possibility that there are more efficient and easier to manage alternative databases that might function better on a Cloud based platform. In this report we will be comparing MySQL, MongoDB, and Cassandra databases in order to determine what is the best database to use.

Please check the table of contents for an overview of what is contained within this report.

Table of Contents

Datasets 5
Identifying Different Types of Datasets 5
Relational vs Non Relational 5
Choosing a Dataset 6
Dataset 1: Employees Sample Database 6
Dataset details 6
Dataset 2: Citibike System Data (https://ride.citibikenyc.com/system-data) 6
Dataset Details 6
No SQL 6
Different types of NoSQL Databases 6
No SQL Databases in Relation to Datasets 7
Cassandra 7
MongoDB 7
Testing Results 8
Testing Plan 8
Test Results[a, b] 8
Test Case 1: Bulk Insert of Data[c1-results] 8
Test Case 2: Retrieving Data using Primary Index[c2-results] 9
Test Case 3: Retrieving Data using Secondary Index[c3-results] 9
Test Case 4: Retrieving Data using No Index[c4-results] 9
Test Case 5: Sorting data using Secondary Index[c5-results] 9
Test Case 6: Sorting Data using No index[c6-results] 10
Test Case 7: Update using Primary Index[c7-results] 10
Test Case 8: Bulk Update using Secondary Index[c8-results] 10
Test Case 9: Delete using Primary Index[c9-results] 10
Test Case 10: Bulk Delete using Secondary Index[c10-results] 10
Test Case 11: Bulk Delete using No Index[c11-results] 11
Test Case 12: Sorting Data in Descending Order using Primary Index[c12-results] 11
Security 11

MySQL 11
Cassandra 12
MongoDB 13

Recommendations 13

References 15

Appendix A: Detailed Testing Results 16
Case 1 16
Case 2 16
Case 3 16
Case 4 16
Case 5 16
Case 6 17
Case 7 17
Case 8 17
Case 9 17
Case 10 17
Case 11 17
Case 12 17

Appendix B: Testing Screenshots 19
Test Case 1 Screenshots 19
MySQL 19
MongoDB 20
Cassandra 20
Test Case 2 Screenshots 21
MySQL 21
MongoDB 21
Cassandra 22
Test Case 3 Screenshots 22
MySQL 22
MongoDB 23
Cassandra 23

Test Case 4 Screenshots 24
MySQL 24
MongoDB 24
Cassandra 25
Test Case 5 Screenshots 25
MySQL 25
MongoDB 26
Cassandra 26
Test Case 6 Screenshots 27
MySQL 27
MongoDB 27
Cassandra 28
Test Case 7 Screenshots 28
MySQL 28
MongoDB 28
Cassandra 28
Test Case 8 Screenshots 29
MySQL 29
MongoDB 29
Cassandra 29
Test Case 9 Screenshots 30
MySQL 30
MongoDB 30
Cassandra 30
Test Case 10 Screenshots 31
MySQL 31
MongoDB 31
Cassandra 31
Test Case 11 Screenshots 32
MySQL 32
MongoDB 32
Cassandra 32

Test Case 12 Screenshots 33
MySQL 33
MongoDB 34
Cassandra 34

Datasets

Identifying Different Types of Datasets

When it comes to choosing what datasets we were going to be using for testing our various databases, we had a bit of trouble initially searching for databases that were relative to what a "Software House" would actually be storing and/or using within their software or servers, since the term is rather vague. There is a wide variety of datasets which all serve different purposes and functions. Types of datasets can range from something that contains the transactions for a shop in a given day, to something much larger, complexer such as a country's world development indicator. Every different dataset requires thought into how it would be stored; be it for quick access when later required, or for archival purposes. In the end, we decided to go with more analytically focused sample datasets; sets where the bulk of the sample consists typically of general user information since we are selling software and most likely need the functionality to log customer and/or product details.

Relational vs Non Relational

Since our company's applications already successfully use MySQL, Oracle and Microsoft SQL Server as the back-ends, we're going to be testing our non relational solutions against MySQL, a relational database. We already know that we can reliably use our on-premises database so we are also going to be comparing the reliability of Cloud databases against this as well. In order to reliably do this we needed to identify a relational database that will be similar length to the database we've chosen for our NoSQL testing. As previously mentioned, it needs to be analytical and CMS focused and of a similar size to get comparable results.

Choosing a Dataset

Dataset 1: Employees Sample Database

Dataset details

Our relational dataset contains various employee data, including the departments in which they have worked, salaries, start and end dates, in addition to data for the managers of said departments.

Dataset 2: Citibike System Data (https://ride.citibikenyc.com/system-data)

Dataset Details

The non-relational dataset that we've chosen to test against is a sample dataset that provides a small sample of Citibike system data. This dataset will work for both of our NoSQL databases, MongoDB and Cassandra, as it's a .csv file with initially a little over 955K rows of data. This makes direct comparison of which NoSQL database will perform better much simpler and will also provide an equal creation comparison with our SQL database which has a little over 1 million records.

No SQL

Different types of NoSQL Databases

There are several well established NoSQL databases accepted within the NoSQL community as well as new databases that are being developed but aren't widely used. We will ignore the latter and focus on two separate approaches to data storage that NoSQL databases use: Document Based and Column Based. For our document based storage we will be using MongoDB and for our column based storage we will be using Cassandra; despite it not being the most popular column based approach, it is easy for beginners to learn and easily cooperates with the Cloud.

No SQL Databases in Relation to Datasets

Since each of our databases have different functionalities we needed to secure separate datasets with similar sizes that will easily integrate with the database we are testing against so that we can produce valid and comparable results. Our first dataset was chosen to be used with MySQL as it's a relational database. The second dataset is a column based dataset primarily for Cassandra's sake; this dataset will also be able to be used with MongoDB as the document based approach is an improved version of the column based approach that Cassandra takes.

Cassandra

The Cassandra database takes a Column based approach to data storage. Instead of organizing data into rows, it organizes them into columns. This essentially has the same functionality as relational databases, which limits the schemaless view that NoSQL typically has. That said, it is much more flexible when compared to a relational database. There's numerous benefits of a column based approach. Aggregation queries are rather fast as the majority of the information is stored in a column; this is important when large amounts of queries are performed in a small amount of time. Column based databases are also incredibly scalable, near infinitely, and are often spread across multiple clusters of machines, potentially numbering in the thousands; companies like Spotify and Netflix use them for this exact reason as their libraries of data are expanding daily. In general, column based databases are great for analytics and reporting.

MongoDB

MongoDB takes a document based approach to storing data. This means that data is stored in a document in the database similar to the format of JSON objects; it accepts formats like BSON and XML as well. Typically documents that contain similar data are gathered into collections. Because not all documents in a collection are required to have the same fields this provides a very flexible schema, unlike relational databases; not every document needs to contain the same data as the other documents and can even include completely different data. That said, there are document databases that can provide schema validation and MongoDB is one of

them. Documents can be mapped to objects which in turn can be used in most programming languages and makes for easy database integration into programs. In addition to this, you can query through an API or query language. With the document approach MongoDB takes, it makes it infinitely scalable similar to Cassandra, however it is scalable horizontally which is typically cheaper than vertical scalability which is how SQL and Cassandra scale. Overall MongoDB would be easy to integrate with programs because of the api functionality and object mapping.

Testing Results

Testing Plan

Using the previously selected datasets, for MongoDB, Cassandra, and MySQL the tests, as listed within the following section, will be performed for each of the databases. The performance times, and quantity of data affected will be compared to one another. This will shed light on which database performs better than the others in each function or process that is being undertaken, such as bulk updating.

Test Results[a, b]

Test Case 1: Bulk Insert of Data[c1-results]

Our first test is bulk insertion of data to see which database performs quicker and theoretically more efficiently when adding data. As to be expected, MySQL copied/created its sample data the fastest with a little under 1.075 million records in 16.579 seconds with an average rate of 64,838 rows a second. MongoDB was the more efficient NoSQL database with a bit over 955 thousand being copied/created in 47.962 seconds with an average rate of 19,916 rows a second. Cassandra by far performed the worst with an astonishing 2 minute and 0.292 seconds creation/copy time of the same 955,210 rows with an average rate of 7941 rows a second.

Test Case 2: Retrieving Data using Primary Index[c2-results]

MongoDB performed the fastest with primary index retrieval. It retrieved all 955210 records in 0.999 seconds, which averaged out to be 10,000 records every 0.01046 seconds. MySQL retrieved 10,001 records in 0.016 seconds and cassandra retrieved 13,816 records in 0.213495 seconds. It's becoming clear only after the second test that Cassandra is not quick when handling large volumes of data.

Test Case 3: Retrieving Data using Secondary Index[c3-results]

Once again, MongoDB retrieved data the quickest. Getting all of its records in 1.597 seconds, which averages to 0.19561 seconds per 117,000 records. MySQL performed very similarly to MongoDB, retrieving 117138 records in 0.203 seconds. Cassandra retrieved 799 records in 0.24456 seconds, much slower than both MongoDB and MySQL with only a fraction of the data.

Test Case 4: Retrieving Data using No Index[c4-results]

MySQL blows the competition out of the water, retrieving 120,051 records in 0.0 seconds; we recognize that there may be milliseconds that were not recorded on the software however. Regardless, it's still faster than both of our NoSQL databases. Mongo comes in at 24,731 records in 1.579 seconds and Cassandra with 9156 records in 0.18521 seconds. Cassandra actually beats Mongo in this test; were cassandra to retrieve the same number of records, it would've performed so in 0.50026 seconds, a third of the time.

Test Case 5: Sorting data using Secondary Index[c5-results]

MongoDb performs the best again with an all 955 thousand rows being sorted in 0.677 seconds; however a memory limit was exceeded. MySQL retrieved 117,075 in 0.203 seconds, which if put in the same scale as Mongo would be roughly around 1.624 seconds (117,075 and 0.203 multiplied by 8 to get you that answer). Cassandra fell behind with less than a tenth of MySQL's total record sort (9156 records) in 0.244178 seconds; when compared to the scale of MySQL's sort, the time would be well over 2 and a half seconds.

Test Case 6: Sorting Data using No index[c6-results]

MongoDB sorted its 955,210 in 0.478 seconds (once again exceeding the memory limit however) where MySQL sorted 300,024 records in 0.438 seconds; nearly a third of the data in the same amount of time. In Cassandra it's not possible to sort or retrieve data without using a primary or secondary index.[b-6-cassandra]

Test Case 7: Update using Primary Index[c7-results]

Both MongoDB and MySQL updated 1 record practically instantaneously. Cassandra updated 123 records in 0.008701 seconds. Overall the results for this were pretty similar.

Test Case 8: Bulk Update using Secondary Index[c8-results]

MySQL updated/changed 117202 records in 8.218 seconds, by far the longest test result we've come across. That said, MongoDB updated/changed all of its records, however the time was not able to be recorded, either because it was too fast or because the program was capable of recording this type of test. In Cassandra, you cannot update any results based on a secondary index,[b-8-cassandra] you need to use the primary index.

Test Case 9: Delete using Primary Index[c9-results]

Our comparison of results for this test are somewhat inapplicable. MySQL did indeed delete its 1 record in near instant timing. Cassandra also deleted 1 record in a little over half a second. MongoDB on the other hand deleted no records and the time was not able to be recorded; this could potentially be because of an error or something else. We will not consider Test Case 9 when making a final recommendation because of this fluke.

Test Case 10: Bulk Delete using Secondary Index[c10-results]

MySQL was able to delete 300,024 records in 6.75 seconds which is about on par for deletion timing with relational databases. MongoDB deleted nearly half of its records, 443,102, however the program was not able to record a time for this. We believe it to be longer than 3 seconds but faster than 8 seconds based on previous

tests, which is admittedly a wide margin of error. Once again, it is not possible to delete without a primary index in Cassandra.[b-10-cassandra]

Test Case 11: Bulk Delete using No Index[c11-results]

MySQL once again deleted 300,024 records in 6.672 seconds. MongoDB deleted 159,338, but the program is not able to record deletion times. Our guess remains similar to what was previously stated. Cassandra cannot delete, sort, or update without including the primary index so this practically eliminates Cassandra from being considered.

Test Case 12: Sorting Data in Descending Order using Primary Index[c12-results]

Our final test shows reports that MySQL was the fastest with sorting in descending order, having sorted 300,024 records in 0.344 seconds. MongoDB sorted all of its records in descending order in 3.009 seconds. Cassandra sorted 17 records in 0.2844 seconds, not even 0.5% of MySQL's records in a slower time. In comparison, MySQL would sort a little over 900 thousand records in 1.032 seconds, still faster than MongoDB.

Security

MySQL

MySQL offers a range of measures for security purposes. Within a MySQL instance, users can be created for accessing the various objects within the instance, databases, functions, tables, and procedures. Users can be limited to connecting by specified IP addresses, which will therefore disallow the user account to be accessed from an unauthorised system. This is backed by (Dubois, 2013), who states that by choosing an appropriate address value for the incoming user connection, access can be limited to specified hosts, at the discretion of the user account creator.

Permissions on objects can be assigned to users. These include, but are not limited to; access to specified databases within the MySQL instance, and individual object permissions such as selecting, deleting, updating, and executing.

Additionally, it is possible that a user may need the ability to retrieve a selection of data from the results of a query where they should not have direct access to the source of the data. For this, MySQL offers objects that are known as Views. Views are like database tables; except they show the results of a query and cannot be directly edited - only users with appropriate permissions would be able to alter the underlying query. With Views, specific columns can be shown to the user, without allowing the user access to the tables from which the data was sourced.

Cassandra

Cassandra offers internal authentication and authorization within its "programming" but has no default permissions or security set up. Cassandra authentication is internally stored and role-based which means you can apply certain permissions to certain roles; for example, an administrator can create, alter, drop, or show the roles with a secure password. The base system is login based, so you will need to create users and passwords and apply the roles to said users. With the role system an individual user can be granted (or revoked) any number of roles; you can also grant a role another role. For example let's say the role "user" exists that allows only select functionality; you create a new user/role (via "CREATE ROLE sam WITH PASSWORD='1234'; ") and grant them the user role, you've then given one role another role. It's a very simple system which would be easy to learn. The downside to this is that it is a simple system and that if a hacker gets high enough access to the database it will be quick and simple to ruin the database.

You can additionally enable SSL encryption so that a client and the database cluster can communicate securely. With SSL encryption enabled it ensures that data is transferred securely and not compromised.However, node to node and client to node encryption are configured independently; you could forget to set it up on one or the other and the encryption will not be secured.

MongoDB

MongoDB has security to offer its users, for example network encryption is available with TLS/SSL this makes sure that data is received by the intended user, you can also limit the network exposure and allow the setting of role based user control. Furthermore, using network encryption and TLS/USS is not bulletproof as there are other paths a hacker could take to breach the database or network, for example SQL injections you think wouldn't work as MongoDB does not use SQL however this does not make it immune to an attack of this nature because queries still contain user supplied data and failing to sanitise the data may turn out hazardous

Limiting network exposure is a quite smart and effective move as it ensures that MongoDB runs in a trusted network environment and configure firewall or security groups to control inbound and outbound traffic for your MongoDB instances also Disable direct SSH root access and only Allow only trusted clients to access the network interfaces and ports on which MongoDB instances are available. Role based user control is great as it restricts system functionally to its authorised users only which is a security measure which is quite effective as other users simply don't have access to information that doesn't pertain to them. Overall mongo DB is considered a secure database and service by conventional means however it most certainly is not bulletproof and has been victim to many cyber attacks so this must be taken into consideration before using mongo.

Recommendations

With our complete analysis of each of our three databases, MySQL, MongoDB, and Cassandra, we've come to the conclusion that it would be best to stick with our MySQL database. The potential costs of migrating the entirety of our relational database to a non-relational, document based database (meaning converting our existing data into JSON format) would not make it worth switching. Had Cassandra not proven to be substantially slower and less efficient than the other databases, it wouldn't have been out of the realm of possibilities to migrate the

database to Cassandra as both are column based; however the testing showed that Cassandra is not the best NoSQL database, not to mention there is far less community support for it whereas MongoDB is well known and has a vast community, especially when regarding tutorials.

Despite not recommending a change in database, we still recommend changing our on-premise database to something more cloud based, whether it's the entire database or we are to create a hybrid system with essential data on the cloud and less essential data remaining stored on our pre-established onsite premise; that way the hardware does not go to waste. Overall cloud databases offer reliable uptime, access from anywhere, and vast scalability. Additionally, the migration of databases to the cloud will remove the responsibility of maintaining computer hardware, as this would be instead handled by the cloud service provider. Cloud-based solutions for backing up of data may also be provided, which would remove the responsibility of handling backups locally, and instead allow for geo redundancy of the backups, this is of course dependent on the service provider for the cloud based solution.

References

DuBois, Paul. (2013). MySQl. (5th Edition). Addison-Wesley Professional

Appendix A: Detailed Testing Results

Test Case	MySQL	Cassandra	MongoDB
Case 1	**Records Inserted:** 1,074,944. **Time Elapsed:** 16.579 seconds **Avg Rate:** 64,838 row/sec	**Rows Inserted:** 955,210 **Time Elapsed:** 2 mins & 0.292 sec **Avg Rate:** 7941 row/sec	**Records Inserted:** 955,210. **Time Elapsed:** 47.962 seconds **Avg Rate:** 19,916 row/sec
Case 2	**Primary Index:** emp_no **Time Elapsed:** 0.016 seconds **Records Retrieved:** 10,001	**Primary Index:** bikeID **Time Elapsed:** 0.213495 seconds **Records:** 13,816	**Primary Index:** _id **Time Elapsed:** 0.999 **Records Retrieved:** 955,210
Case 3	**Secondary Index:** birth_date **Time Elapsed:** 0.203 seconds **Records Retrieved:** 117,138	**Secondary Index:** startStationID **Time Elapsed:** 0.24456 seconds **Records:** 799	**Secondary Index:** bikeid **Time Elapsed:** 1.597 seconds **Records Retrieved:** 955,210
Case 4	**Time Elapsed:** 0.0 seconds **Records Retrieved:** 120,051	**Time:** 0.18521 **Records:** 9156	**Time Elapsed:** 1.579 **Records Retrieved:** 24,731
Case 5	**Secondary Index:** birth_date **Time Elapsed:** 0.266 **Records Sorted:** 117,075	**Secondary Index:** startStationID **Time:** 0.244178 **Records:** 9156	**Secondary Index:** bikeid **Time elapsed:** 0.677 **Records sorted:** 955,210 **Memory limit exceeded**

Case 6	**Time Elapsed:** 0.438 seconds **Records Sorted:** 300,024	**Not possible**	**Time Elapsed:** 0.478 seconds **Records Sorted:** 955,210 **Memory limit exceeded**
Case 7	**Primary Index:** emp_no **Time Elapsed:** 0.0 seconds **Records Updated:** 1	**Primary:** bikeID **Time Elapsed** 0.008701 sec **Records Updated:** 123	**Primary Index:** _id **Time Elapsed: N/A** **Records Updated:** 1
Case 8	**Secondary Index:** birth_date **Time Elapsed:** 8.218 seconds **Records Updated:** 117,202	**Not possible**	**Secondary Index:** bikeid **Time Elapsed: N/A** **Records Updated:** 955,210
Case 9	**Primary Index:** emp_no **Time Elapsed:** 0.0 seconds **Records Deleted:** 1	**Primary:** bikeID & startstationID **Time:** 0.54193 **Deleted:** 1	**Primary Index:** _id **Time Elapsed: N/A** **Records Deleted:** 0
Case 10	**Secondary Index:** birth_Date **Time Elapsed:** 6.750 seconds **Records Deleted:** 300,024	**Not possible**	**Secondary Index:** bikeid **Time Elapsed: N/A** **Records Deleted:** 443,102
Case 11	**Time Elapsed:** 6.672 seconds **Records Deleted:** 300,024	**Not possible**	**Time Elapsed: N/A** **Records Deleted:** 159,338
Case 12	**Primary Index:** emp_no **Time Elapsed:** 0.344 seconds	**Index:** bikeID **Time:** .2844 **Records:** 17	**Primary Index:** _id **Time Elapsed:** 3.009 seconds

	Records Sorted: 300,024		**Records Sorted:** 955,210

Appendix B: Testing Screenshots

Test Case 1 Screenshots

MySQL

#	Time	Action	Duration / Fetch	Message
1	09:46:17	INSERT INTO `departments` VALUES ...	0.016 sec	9 row(s) affected Records: 9 Duplicates: 0 Warnings: 0
2	09:46:53		0.219 sec	17944 row(s) affected Records: 17944 Duplicates: 0 Warnings: 0
3	09:46:53		0.187 sec	17938 row(s) affected Records: 17938 Duplicates: 0 Warnings: 0
4	09:46:53		0.203 sec	17953 row(s) affected Records: 17953 Duplicates: 0 Warnings: 0
5	09:46:54		0.203 sec	17947 row(s) affected Records: 17947 Duplicates: 0 Warnings: 0
6	09:46:54		0.204 sec	17948 row(s) affected Records: 17948 Duplicates: 0 Warnings: 0
7	09:46:54		0.172 sec	17648 row(s) affected Records: 17648 Duplicates: 0 Warnings: 0
8	09:46:54		0.219 sec	17642 row(s) affected Records: 17642 Duplicates: 0 Warnings: 0
9	09:46:54		0.203 sec	17638 row(s) affected Records: 17638 Duplicates: 0 Warnings: 0
10	09:46:55		0.203 sec	17653 row(s) affected Records: 17653 Duplicates: 0 Warnings: 0
11	09:46:55		0.188 sec	17650 row(s) affected Records: 17650 Duplicates: 0 Warnings: 0
12	09:46:55		0.204 sec	17636 row(s) affected Records: 17636 Duplicates: 0 Warnings: 0
13	09:46:55		0.234 sec	17642 row(s) affected Records: 17642 Duplicates: 0 Warnings: 0
14	09:46:56		0.172 sec	17646 row(s) affected Records: 17646 Duplicates: 0 Warnings: 0
15	09:46:56		0.171 sec	17642 row(s) affected Records: 17642 Duplicates: 0 Warnings: 0
16	09:46:56		0.187 sec	17651 row(s) affected Records: 17651 Duplicates: 0 Warnings: 0
17	09:46:56		0.172 sec	17637 row(s) affected Records: 17637 Duplicates: 0 Warnings: 0
18	09:46:56		0.187 sec	16209 row(s) affected Records: 16209 Duplicates: 0 Warnings: 0
19	09:48:41		0.344 sec	21707 row(s) affected Records: 21707 Duplicates: 0 Warnings: 0
20	09:48:41		0.344 sec	21700 row(s) affected Records: 21700 Duplicates: 0 Warnings: 0
21	09:48:42		0.344 sec	21687 row(s) affected Records: 21687 Duplicates: 0 Warnings: 0
22	09:48:42		0.359 sec	21694 row(s) affected Records: 21694 Duplicates: 0 Warnings: 0
23	09:48:43		0.360 sec	21695 row(s) affected Records: 21695 Duplicates: 0 Warnings: 0
24	09:48:43		0.391 sec	21700 row(s) affected Records: 21700 Duplicates: 0 Warnings: 0
25	09:48:43		0.328 sec	21326 row(s) affected Records: 21326 Duplicates: 0 Warnings: 0
26	09:48:44		0.328 sec	21257 row(s) affected Records: 21257 Duplicates: 0 Warnings: 0
27	09:48:44		0.390 sec	21245 row(s) affected Records: 21245 Duplicates: 0 Warnings: 0
28	09:48:44		0.296 sec	21266 row(s) affected Records: 21266 Duplicates: 0 Warnings: 0
29	09:48:45		0.344 sec	21279 row(s) affected Records: 21279 Duplicates: 0 Warnings: 0
30	09:48:45		0.422 sec	21251 row(s) affected Records: 21251 Duplicates: 0 Warnings: 0
31	09:48:46		0.344 sec	21270 row(s) affected Records: 21270 Duplicates: 0 Warnings: 0
32	09:48:46		0.344 sec	21261 row(s) affected Records: 21261 Duplicates: 0 Warnings: 0
33	09:48:46		0.329 sec	21251 row(s) affected Records: 21251 Duplicates: 0 Warnings: 0
34	09:48:47		0.344 sec	21272 row(s) affected Records: 21272 Duplicates: 0 Warnings: 0
35	09:48:47		0.375 sec	21258 row(s) affected Records: 21258 Duplicates: 0 Warnings: 0
36	09:48:48		0.344 sec	21252 row(s) affected Records: 21252 Duplicates: 0 Warnings: 0
37	09:48:48		0.360 sec	21260 row(s) affected Records: 21260 Duplicates: 0 Warnings: 0
38	09:48:48		0.375 sec	21276 row(s) affected Records: 21276 Duplicates: 0 Warnings: 0
39	09:48:49		0.218 sec	15401 row(s) affected Records: 15401 Duplicates: 0 Warnings: 0

42	09:52:19		0.469 sec	24940 row(s) affected Records: 24940 Duplicates: 0 Warnings: 0
43	09:52:19		0.437 sec	24940 row(s) affected Records: 24940 Duplicates: 0 Warnings: 0
44	09:52:20		0.468 sec	24940 row(s) affected Records: 24940 Duplicates: 0 Warnings: 0
45	09:52:20		0.468 sec	24934 row(s) affected Records: 24934 Duplicates: 0 Warnings: 0
46	09:52:21		0.390 sec	24360 row(s) affected Records: 24360 Duplicates: 0 Warnings: 0
47	09:52:21		0.438 sec	24360 row(s) affected Records: 24360 Duplicates: 0 Warnings: 0
48	09:52:22		0.406 sec	24360 row(s) affected Records: 24360 Duplicates: 0 Warnings: 0
49	09:52:22		0.453 sec	24360 row(s) affected Records: 24360 Duplicates: 0 Warnings: 0
50	09:52:23		0.438 sec	24360 row(s) affected Records: 24360 Duplicates: 0 Warnings: 0
51	09:52:23		0.453 sec	24360 row(s) affected Records: 24360 Duplicates: 0 Warnings: 0
52	09:52:24		0.453 sec	24360 row(s) affected Records: 24360 Duplicates: 0 Warnings: 0
53	09:52:24		0.422 sec	24360 row(s) affected Records: 24360 Duplicates: 0 Warnings: 0
54	09:52:25		0.438 sec	24360 row(s) affected Records: 24360 Duplicates: 0 Warnings: 0
55	09:52:25		0.219 sec	12609 row(s) affected Records: 12609 Duplicates: 0 Warnings: 0

MongoDB

```
2022-04-05T09:01:08.627+0000    connected to: mongodb://172.17.0.2/
2022-04-05T09:01:11.628+0000    [#.......................] citibike.records    10.5MB/172MB (6.1%)
2022-04-05T09:01:14.627+0000    [##......................] citibike.records    21.4MB/172MB (12.4%)
2022-04-05T09:01:17.627+0000    [####....................] citibike.records    32.1MB/172MB (18.6%)
2022-04-05T09:01:20.628+0000    [#####...................] citibike.records    42.8MB/172MB (24.8%)
2022-04-05T09:01:23.628+0000    [#######.................] citibike.records     53.3MB/172MB (30.9%)
2022-04-05T09:01:26.628+0000    [########................] citibike.records     63.9MB/172MB (37.1%)
2022-04-05T09:01:29.628+0000    [##########..............] citibike.records     74.5MB/172MB (43.2%)
2022-04-05T09:01:32.628+0000    [###########.............] citibike.records     84.7MB/172MB (49.1%)
2022-04-05T09:01:35.629+0000    [#############...........] citibike.records     95.3MB/172MB (55.3%)
2022-04-05T09:01:38.627+0000    [##############..........] citibike.records     106MB/172MB (61.5%)
2022-04-05T09:01:41.627+0000    [################........] citibike.records     117MB/172MB (67.6%)
2022-04-05T09:01:44.628+0000    [################........] citibike.records     122MB/172MB (70.6%)
2022-04-05T09:01:47.627+0000    [##################......] citibike.records     133MB/172MB (76.8%)
2022-04-05T09:01:50.628+0000    [###################.....] citibike.records     143MB/172MB (83.0%)
2022-04-05T09:01:53.627+0000    [#####################...] citibike.records     153MB/172MB (89.0%)
2022-04-05T09:01:56.627+0000    [######################..] citibike.records     162MB/172MB (93.8%)
2022-04-05T09:01:59.590+0000    [########################] citibike.records     172MB/172MB (100.0%)
2022-04-05T09:01:59.590+0000    955210 document(s) imported successfully. 0 document(s) failed to import.
```

Cassandra

```
cqlsh:crud_ks> copy citi (tripduration, starttime, stoptime, startStationID, startStationName
, startStationLatitude, statStationLongitude, endStationID, endStationName, endStationLatitud
e, endStationLongitude, bikeID, userType, birthYear, gender) from 'copy.csv' with delimiter='
,' and header=true;
Using 3 child processes

Starting copy of crud_ks.citi with columns [tripduration, starttime, stoptime, startstationid
, startstationname, startstationlatitude, statstationlongitude, endstationid, endstationname,
 endstationlatitude, endstationlongitude, bikeid, usertype, birthyear, gender].
Processed: 955210 rows; Rate:    4425 rows/s; Avg. rate:    7941 rows/s
955210 rows imported from 1 files in 0 day, 0 hour, 2 minutes, and 0.292 seconds (0 skipped).
cqlsh:crud_ks>
```

Test Case 2 Screenshots

MySQL

```
1 •   SELECT * FROM employees WHERE emp_no >= 20000 AND emp_no <= 30000;
2
```

Tabular Explain

d	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	employees		range	PRIMARY	PRIMARY	4		19930	100.00	Using where

employees 13 ×

Output

Action Output

#	Time	Duration / Fetch	Action	Message
1	11:03:52	0.000 sec / 0.016 sec	SELECT * FROM employees WHERE emp_no >= 20000 AND emp_no <= 30000	10001 row(s) returned
2	11:04:34	0.000 sec	EXPLAIN SELECT * FROM employees WHERE emp_no >= 20000 AND emp_no <= 3...	OK

MongoDB


```
> db.records.find().explain(true);
{
        "explainVersion" : "1",
        "queryPlanner" : {
                "namespace" : "citibike.records",
                "indexFilterSet" : false,
                "parsedQuery" : {

                },
                "maxIndexedOrSolutionsReached" : false,
                "maxIndexedAndSolutionsReached" : false,
                "maxScansToExplodeReached" : false,
                "winningPlan" : {
                        "stage" : "COLLSCAN",
                        "direction" : "forward"
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 955210,
                "executionTimeMillis" : 999,
                "totalKeysExamined" : 0,
                "totalDocsExamined" : 955210,
                "executionStages" : {
                        "stage" : "COLLSCAN",
                        "nReturned" : 955210,
                        "executionTimeMillisEstimate" : 176,
                        "works" : 955212,
                        "advanced" : 955210,
                        "needTime" : 1,
                        "needYield" : 0,
                        "saveState" : 955,
                        "restoreState" : 955,
                        "isEOF" : 1,
                        "direction" : "forward",
                        "docsExamined" : 955210
                },
                "allPlansExecution" : [ ]
        },
```

Cassandra

```
cqlsh:crud_ks>
cqlsh:crud_ks> select * from citi where bikeid>300 ALLOW FILTERING;
```

```
| 2019-12-31 16:
 Subscriber

(13816 rows)
```

Test Case 3 Screenshots

MySQL

```
1 •   SELECT * FROM employees WHERE birth_date >= '1960-01-01' AND birth_date <= '1989-12-31'
2
```

Tabular Explain

select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
SIMPLE	employees		ALL	I_BIRTH_DATE				299778	50.00	Using where

nployees 6 ×

utput

Action Output

#	Time	Duration / Fetch	Action	Message
1	11:10:05	0.000 sec / 0.203 sec	SELECT * FROM employees WHERE birth_date >= '1960-01-01' AND birth_date <= '1989-12-31'	117138 row(s) returned
2	11:10:09	0.000 sec	EXPLAIN SELECT * FROM employees WHERE birth_date >= '1960-01-01' AND birth_date <= '1989-12-31'	OK

MongoDB

```
> db.records.find({ bikeid: { $gt: 1000 } }).explain(true);
{
        "explainVersion" : "1",
        "queryPlanner" : {
                "namespace" : "citibike.records",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "bikeid" : {
                                "$gt" : 1000
                        }
                },
                "maxIndexedOrSolutionsReached" : false,
                "maxIndexedAndSolutionsReached" : false,
                "maxScansToExplodeReached" : false,
                "winningPlan" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "bikeid" : {
                                        "$gt" : 1000
                                }
                        },
                        "direction" : "forward"
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 955210,
                "executionTimeMillis" : 1597,
                "totalKeysExamined" : 0,
                "totalDocsExamined" : 955210,
                "executionStages" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "bikeid" : {
                                        "$gt" : 1000
                                }
                        },
                        "nReturned" : 955210,
                        "executionTimeMillisEstimate" : 540,
                        "works" : 955212,
                        "advanced" : 955210,
                        "needTime" : 1,
                        "needYield" : 0,
                        "saveState" : 955,
                        "restoreState" : 955,
                        "isEOF" : 1,
                        "direction" : "forward",
                        "docsExamined" : 955210
                },
```

Cassandra

```
 & Smith St

(799 rows)
```

24456

```
cqlsh:crud_ks>
cqlsh:crud_ks> select * from citi where startstationid=3382;
```

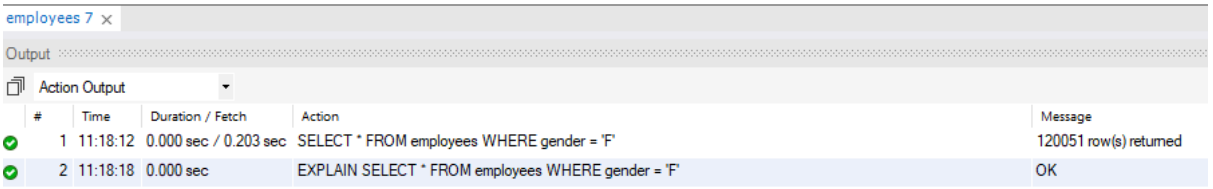
Test Case 4 Screenshots

MySQL

```
1 •   SELECT * FROM employees WHERE gender = 'F';
2
```

Tabular Explain

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	employees		ALL					299778	50.00	Using where



employees 7

Output

Action Output

#	Time	Duration / Fetch	Action	Message
1	11:18:12	0.000 sec / 0.203 sec	SELECT * FROM employees WHERE gender = 'F'	120051 row(s) returned
2	11:18:18	0.000 sec	EXPLAIN SELECT * FROM employees WHERE gender = 'F'	OK

MongoDB


```
> db.records.find({ "birth year": {$eq: 1995} }).explain(true);
{
        "explainVersion" : "1",
        "queryPlanner" : {
                "namespace" : "citibike.records",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "birth year" : {
                                "$eq" : 1995
                        }
                },
                "maxIndexedOrSolutionsReached" : false,
                "maxIndexedAndSolutionsReached" : false,
                "maxScansToExplodeReached" : false,
                "winningPlan" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "birth year" : {
                                        "$eq" : 1995
                                }
                        },
                        "direction" : "forward"
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 24731,
                "executionTimeMillis" : 1579,
                "totalKeysExamined" : 0,
                "totalDocsExamined" : 955210,
                "executionStages" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "birth year" : {
                                        "$eq" : 1995
                                }
                        },
```

Cassandra

```
cqlsh:crud_ks>
cqlsh:crud_ks> select * from citi where gender=1 ALLOW FILTERING;
```

```
1 16:47:45.63

(9156 rows)
```

Test Case 5 Screenshots

MySQL

```
1 •   SELECT * FROM employees WHERE birth_date > '1960-01-01' ORDER BY birth_date;
2
```

Tabular Explain

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	employees		ALL	I_BIRTH_DATE				299778	50.00	Using where; Using filesort

employees 11 ×

Output

Action Output

#	Time	Duration / Fetch	Action	Message
1	11:43:13	0.172 sec / 0.094 sec	SELECT * FROM employees WHERE birth_date > '1960-01-01' ORDER BY birth_date	117075 row(s) returned
2	11:43:17	0.000 sec	EXPLAIN SELECT * FROM employees WHERE birth_date > '1960-01-01' ORDER BY birth_date	OK

MongoDB

```
> db.records.find().sort({ bikeid: 1 }).explain(true);
{
        "explainVersion" : "1",
        "queryPlanner" : {
                "namespace" : "citibike.records",
                "indexFilterSet" : false,
                "parsedQuery" : {

                },
                "maxIndexedOrSolutionsReached" : false,
                "maxIndexedAndSolutionsReached" : false,
                "maxScansToExplodeReached" : false,
                "winningPlan" : {
                        "stage" : "SORT",
                        "sortPattern" : {
                                "bikeid" : 1
                        },
                        "memLimit" : 104857600,
                        "type" : "simple",
                        "inputStage" : {
                                "stage" : "COLLSCAN",
                                "direction" : "forward"
                        }
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : false,
                "errorMessage" : "Sort exceeded memory limit of 104857600 bytes, but did not opt in to external sorting.",
                "errorCode" : 292,
                "nReturned" : 0,
                "executionTimeMillis" : 677,
                "totalKeysExamined" : 0,
                "totalDocsExamined" : 222933,
                "failed" : true,
                "executionStages" : {
                        "stage" : "SORT",
                        "nReturned" : 0,
                        "executionTimeMillisEstimate" : 360,
                        "works" : 222934,
                        "advanced" : 0,
                        "needTime" : 222933,
                        "needYield" : 0,
                        "saveState" : 222,
                        "restoreState" : 222,
                        "failed" : true,
                        "isEOF" : 0,
                        "sortPattern" : {
                                "bikeid" : 1
                        },
                        "memLimit" : 104857600,
```

Cassandra

```
cqlsh:crud_ks>
cqlsh:crud_ks> select * from citi where startStationID>500 order by startStationID ALLOW FILTERING;
```

```
(9156 rows)
```

Test Case 6 Screenshots

MySQL

```
1 •   SELECT * FROM employees ORDER BY gender;
2
```

Tabular Explain

d	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	employees		ALL					299778	100.00	Using filesort

employees 8 ×

Output

Action Output

#	Time	Duration / Fetch	Action	Message
1	11:35:15	0.203 sec / 0.235 sec	SELECT * FROM employees ORDER BY gender	300024 row(s) returned
2	11:35:35	0.000 sec	EXPLAIN SELECT * FROM employees ORDER BY gender	OK

MongoDB

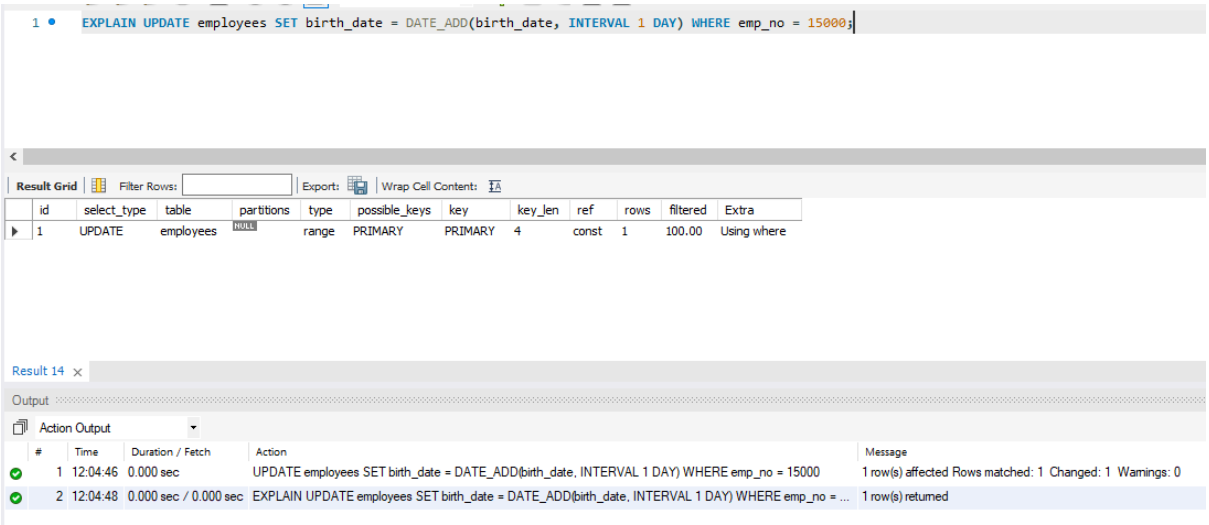
```
> db.records.find().sort({ "birth date": 1 }).explain(true);
{
        "explainVersion" : "1",
        "queryPlanner" : {
                "namespace" : "citibike.records",
                "indexFilterSet" : false,
                "parsedQuery" : {

                },
                "maxIndexedOrSolutionsReached" : false,
                "maxIndexedAndSolutionsReached" : false,
                "maxScansToExplodeReached" : false,
                "winningPlan" : {
                        "stage" : "SORT",
                        "sortPattern" : {
                                "birth date" : 1
                        },
                        "memLimit" : 104857600,
                        "type" : "simple",
                        "inputStage" : {
                                "stage" : "COLLSCAN",
                                "direction" : "forward"
                        }
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : false,
                "errorMessage" : "Sort exceeded memory limit of 104857600 bytes, but did not opt in to external sorting.",
                "errorCode" : 292,
                "nReturned" : 0,
                "executionTimeMillis" : 478,
                "totalKeysExamined" : 0,
                "totalDocsExamined" : 222933,
                "failed" : true,
                "executionStages" : {
                        "stage" : "SORT",
                        "nReturned" : 0,
                        "executionTimeMillisEstimate" : 135,
                        "works" : 222934,
                        "advanced" : 0,
                        "needTime" : 222933,
                        "needYield" : 0,
                        "saveState" : 222,
                        "restoreState" : 222,
                        "failed" : true,
                        "isEOF" : 0,
                        "sortPattern" : {
                                "birth date" : 1
                        },
```

Cassandra

```
InvalidRequest: Error from server: code=2200 [Invalid query] message="Some partitio
n key parts are missing: bikeid"
```

Test Case 7 Screenshots

MySQL



MongoDB

```
> db.records.updateOne({"_id": ObjectId("624c055499e258e8d08d96ad")}, {$set: {"birth year": 1996}});
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Cassandra

```
cqlsh:crud_ks> update citi SET birthyear = 1999 where bikeID=41932;

Tracing session: acfb1610-b45c-11ec-8d4c-d516455a14e0

 activity                                                                                  | timestamp                  | source      | source_elapsed | client
-------------------------------------------------------------------------------------------+----------------------------+-------------+----------------+-----------
                                                                Execute CQL3 query | 2022-04-04 21:17:47.377000 | 172.19.0.2 |              0 | 127.0.0.1
 Parsing update citi SET birthyear = 1999 where bikeID=41932; [Native-Transport-Requests-1] | 2022-04-04 21:17:47.377000 | 172.19.0.2 |            200 | 127.0.0.1
                                                 Preparing statement [Native-Transport-Requests-1] | 2022-04-04 21:17:47.378000 | 172.19.0.2 |            756 | 127.0.0.1
                                  Determining replicas for mutation [Native-Transport-Requests-1] | 2022-04-04 21:17:47.380000 | 172.19.0.2 |           2997 | 127.0.0.1
                                                      Appending to commitlog [MutationStage-2] | 2022-04-04 21:17:47.385000 | 172.19.0.2 |           7459 | 127.0.0.1
                                                      Adding to citi memtable [MutationStage-2] | 2022-04-04 21:17:47.385000 | 172.19.0.2 |           8143 | 127.0.0.1
                                                                   Request complete | 2022-04-04 21:17:47.385701 | 172.19.0.2 |           8701 | 127.0.0.1
```

Test Case 8 Screenshots

MySQL

```
1 •  EXPLAIN UPDATE employees SET birth_date = DATE_ADD(birth_date, INTERVAL 1 DAY) WHERE birth_date >= '1960-01-01';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	UPDATE	employees	NULL	index	I_BIRTH_DATE	PRIMARY	4	NULL	299335	100.00	Using where

Result 13

Output

Action Output

#	Time	Duration / Fetch	Action	Message
1	12:01:56	8.218 sec	UPDATE employees SET birth_date = DATE_ADD(birth_date, INTERVAL 1 DAY) WHERE birth_date >= '1960-0...	117202 row(s) affected Rows matched: 117202 Changed: 117202 Warnings: 0
2	12:02:09	0.000 sec / 0.000 sec	EXPLAIN UPDATE employees SET birth_date = DATE_ADD(birth_date, INTERVAL 1 DAY) WHERE birth_date ...	1 row(s) returned

MongoDB

```
> db.records.updateMany({ bikeid: {$gt: 1000 } }, { $set: { "usertype": "Subscriber" } });
{ "acknowledged" : true, "matchedCount" : 955210, "modifiedCount" : 955210 }
```

Cassandra

```
cqlsh:crud_ks> update citi set startstationname='geeric' where startstationid=362;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Some partitio
n key parts are missing: bikeid"
```

Test Case 9 Screenshots

MySQL

```
1    EXPLAIN DELETE FROM employees WHERE emp_no = 10000;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	DELETE	employees	NULL	range	PRIMARY	PRIMARY	4	const	1	100.00	Using where

Result 16

Output

Action Output

#	Time	Duration / Fetch	Action	Message
1	12:09:58	0.000 sec	DELETE FROM employees WHERE emp_no = 200000	1 row(s) affected
2	12:11:43	0.000 sec / 0.000 sec	EXPLAIN DELETE FROM employees WHERE emp_no = 10000	1 row(s) returned

MongoDB

```
> db.records.deleteOne({"_id": ObjectId("624c055499e258e8d08d96ad")});
{ "acknowledged" : true, "deletedCount" : 0 }
```

Cassandra

```
Request complete | 2022-04-04 22:14:48.183193 | 172.19.0.2 |        54193 |

delete from citi where bikeid=15334 and startstationid=146 if exists;
```

Test Case 10 Screenshots

MySQL

```
7 •  EXPLAIN DELETE FROM employees WHERE birth_Date > '1950-01-01'
8
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	DELETE	employees	NULL	range	I_BIRTH_DATE	I_BIRTH_DATE	3	const	1	100.00	Using where

Result 20

Output

Action Output

#	Time	Duration / Fetch	Action	Message
1	16:43:08	6.750 sec	DELETE FROM employees WHERE birth_Date > '1950-01-01'	300024 row(s) affected
2	16:43:16	0.000 sec / 0.000 sec	EXPLAIN DELETE FROM employees WHERE birth_Date > '19...	1 row(s) returned

MongoDB

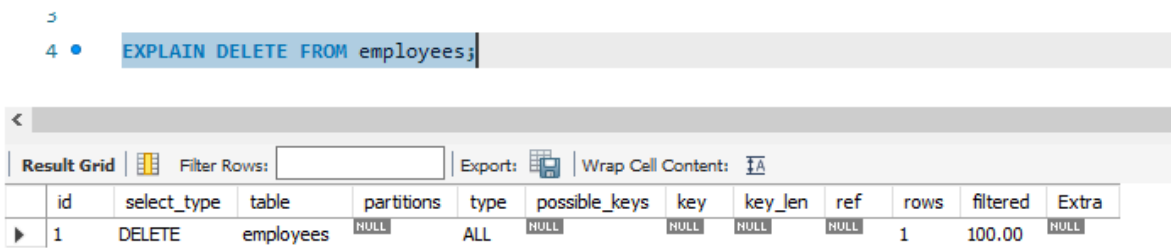
```
> db.records.deleteMany({ "bikeid": { $gt: 30000 } });
{ "acknowledged" : true, "deletedCount" : 443102 }
```

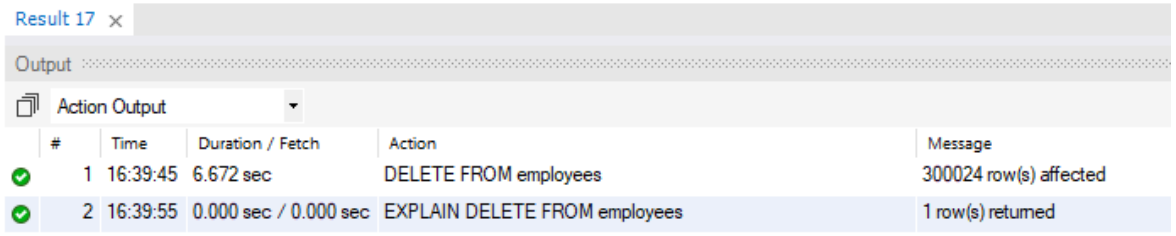
Cassandra

```
cqlsh:crud_ks> delete from citi where startstationid=146 if exists;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Some partition key parts are missing: bikeid"
```

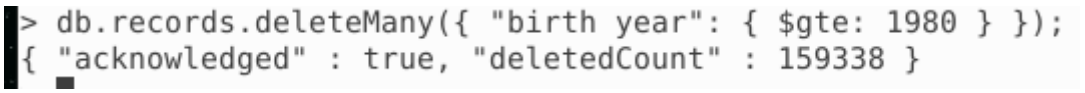
Test Case 11 Screenshots

MySQL

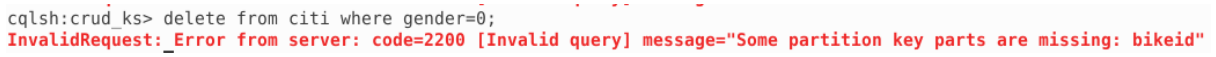




MongoDB



Cassandra



Test Case 12 Screenshots

MySQL

```
1 •  EXPLAIN SELECT * FROM employees ORDER BY emp_no DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	employees	NULL	index	NULL	PRIMARY	4	NULL	299157	100.00	Backward index scan

Result 4

Output

Action Output

#	Time	Duration / Fetch	Action	Message
1	16:27:21	0.000 sec / 0.344 sec	SELECT * FROM employees ORDER BY emp_no DESC	300024 row(s) returned
2	16:27:24	0.000 sec / 0.000 sec	EXPLAIN SELECT * FROM employees ORDER BY emp_no DE...	1 row(s) returned

MongoDB

```
> db.records.find().sort({ _id: -1 }).explain(true);
{
        "explainVersion" : "1",
        "queryPlanner" : {
                "namespace" : "citibike.records",
                "indexFilterSet" : false,
                "parsedQuery" : {

                },
                "maxIndexedOrSolutionsReached" : false,
                "maxIndexedAndSolutionsReached" : false,
                "maxScansToExplodeReached" : false,
                "winningPlan" : {
                        "stage" : "FETCH",
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "keyPattern" : {
                                        "_id" : 1
                                },
                                "indexName" : "_id_",
                                "isMultiKey" : false,
                                "multiKeyPaths" : {
                                        "_id" : [ ]
                                },
                                "isUnique" : true,
                                "isSparse" : false,
                                "isPartial" : false,
                                "indexVersion" : 2,
                                "direction" : "backward",
                                "indexBounds" : {
                                        "_id" : [
                                                "[MaxKey, MinKey]"
                                        ]
                                }
                        }
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 955210,
                "executionTimeMillis" : 3009,
                "totalKeysExamined" : 955210,
                "totalDocsExamined" : 955210,
                "executionStages" : {
                        "stage" : "FETCH",
                        "nReturned" : 955210,
                        "executionTimeMillisEstimate" : 890,
                        "works" : 955211,
                        "advanced" : 955210,
                        "needTime" : 0,
```

Cassandra

```
                                  Request complete | 2022-04-04 22:30:16.652844 | 172.19.0.2 |           2844 |

select * from citi where bikeid=15334;
```