Assignment 1: Demonstrate the creation of an index on a table and discuss how it improves query performance. Use a DROP INDEX statement to remove the index and analyze the impact on query execution.

Creation of an index on a table:

```
mysql> create index table_index ON users(email);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show indexes from users;
+-------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table | Non_unique | Key_name   | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| users |          0 | PRIMARY    |            1 | id          | A         |           6 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| users |          1 | table_index |           1 | email       | A         |           7 |     NULL |   NULL | YES  | BTREE      |         |               | YES     | NULL       |
+-------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
2 rows in set (0.01 sec)
```

How this index improves query performance:

1. Fast Email Searches: When I execute a query that involves searching for a specific email address or filtering by email, the database engine can use the `email_index` to quickly locate the relevant rows without scanning the entire table.

2. Efficient Email Joins: If I join the `users` table with another table using the `email` column, the index can speed up the join operation by allowing the database engine to efficiently locate matching rows based on email addresses.

3. Optimized Email Sorting: Sorting the results by email becomes faster with the index, as the database engine can use it to quickly retrieve the rows in the desired order without the need for costly sorting operations.

Removing the index:

```
mysql> drop index table_index on users;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> show indexes from users;
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| users |          0 | PRIMARY  |            1 | id          | A         |           6 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
1 row in set (0.00 sec)
```

After dropping the `email_index`, queries that rely on filtering, joining, or sorting by the `email` column may experience degraded performance. Without the index, the database engine will have to resort to full table scans or less efficient methods to fulfill these queries.

In summary, creating an index on the `email` column of the `users` table can significantly improve query performance for operations involving email searches, joins, and sorting. However, dropping the index can lead to slower query execution times for such operations.

Therefore, it's important to consider the trade-offs and the specific usage patterns of your database before creating or removing indexes.