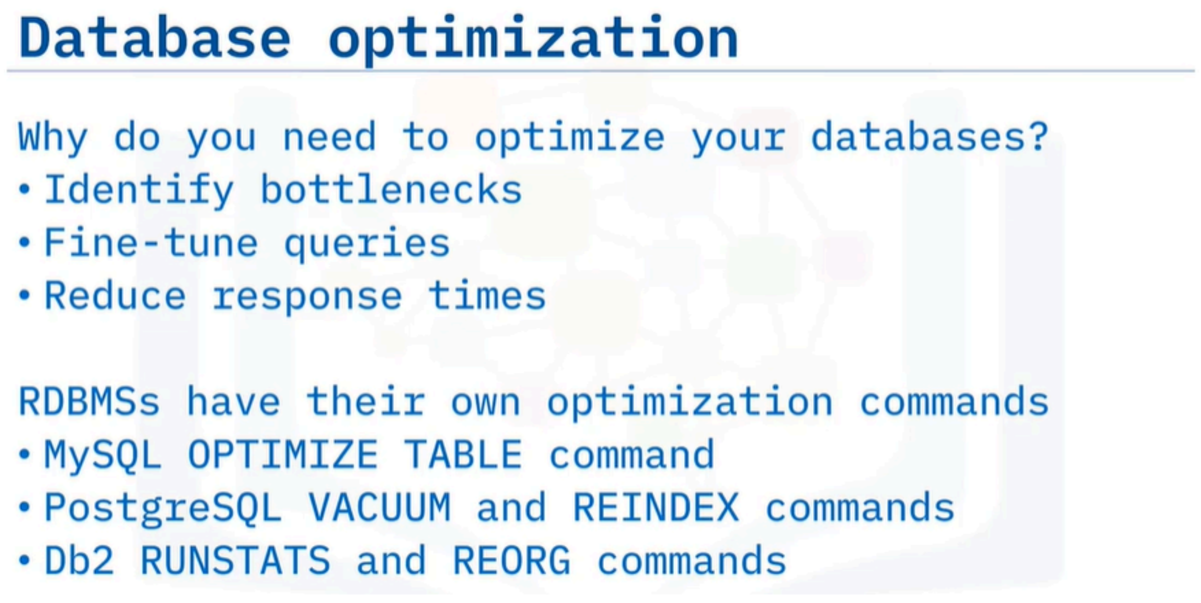
Optimizing Databases

Sure! Let's talk about the importance of optimizing databases.

Optimizing databases is like cleaning and organizing a messy room. Over time, as you add more items (or data) to your room (or database), things can get cluttered and disorganized. This clutter can make it hard to find what you need quickly. Similarly, when databases become fragmented or filled with unused space, they can slow down and make it harder for users to access the information they need. By optimizing, you can tidy up the database, making it faster and more efficient, just like a clean room allows you to find your belongings easily.

For example, in MySQL, you can use a command called `OPTIMIZE TABLE` to reorganize the data and free up space. Think of it as rearranging your furniture to create more space and improve the flow of the room. In PostgreSQL, you have commands like `VACUUM` that help clean up the unused space left behind after deleting items. This way, your database runs smoothly, just like a well-organized room makes it easier to live in.



Database optimization

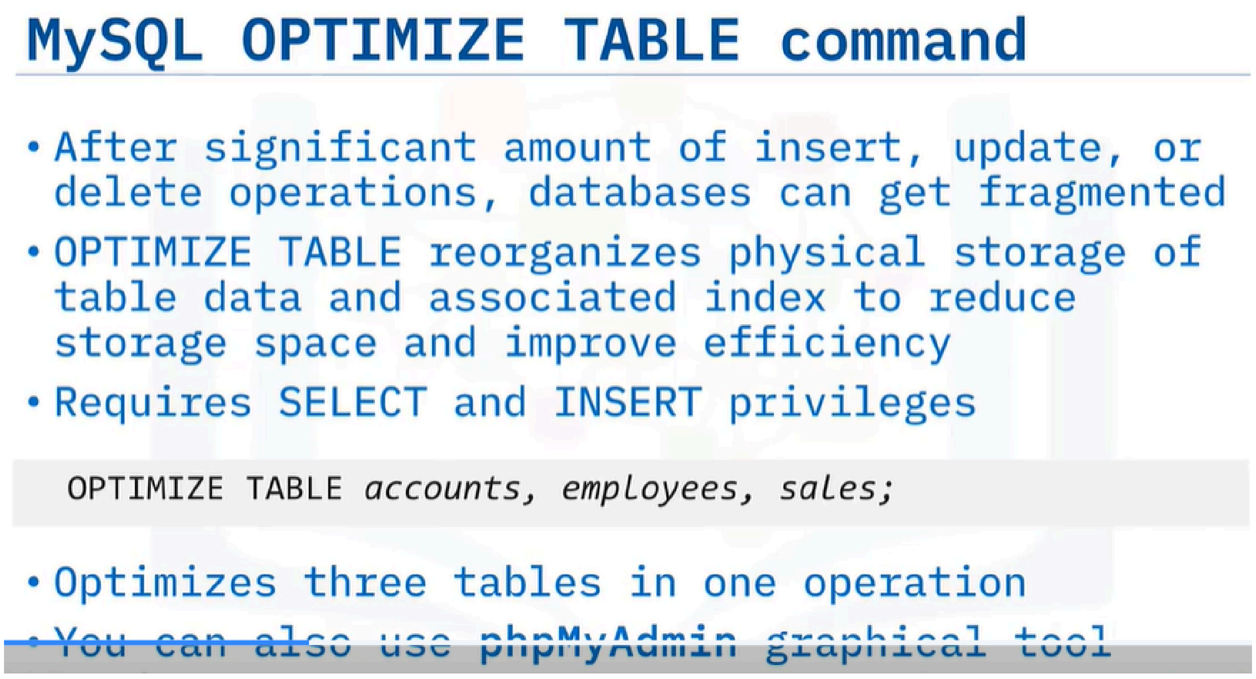
Why do you need to optimize your databases?

- Identify bottlenecks
- Fine-tune queries
- Reduce response times

RDBMSs have their own optimization commands

- MySQL OPTIMIZE TABLE command
- PostgreSQL VACUUM and REINDEX commands
- Db2 RUNSTATS and REORG commands

- First off, let’s ask the obvious question...why do you need to optimize your databases? Well, over time as the volume of data stored in your databases increases, and their operational workloads increase, data can become fragmented, tables can be left empty or partially empty, and database performance can suffer. By optimizing your databases, you can identify bottlenecks, fine-tune database queries, and ultimately reduce database response times for your users. Each relational database system has its own utilities or commands for optimizing its databases. For example, for MySQL databases you can use the OPTIMIZE TABLE command. For PostgreSQL databases you can use the VACUUM and REINDEX commands. And in Db2 you can use the RUNSTATS and REORG commands.



MySQL OPTIMIZE TABLE command

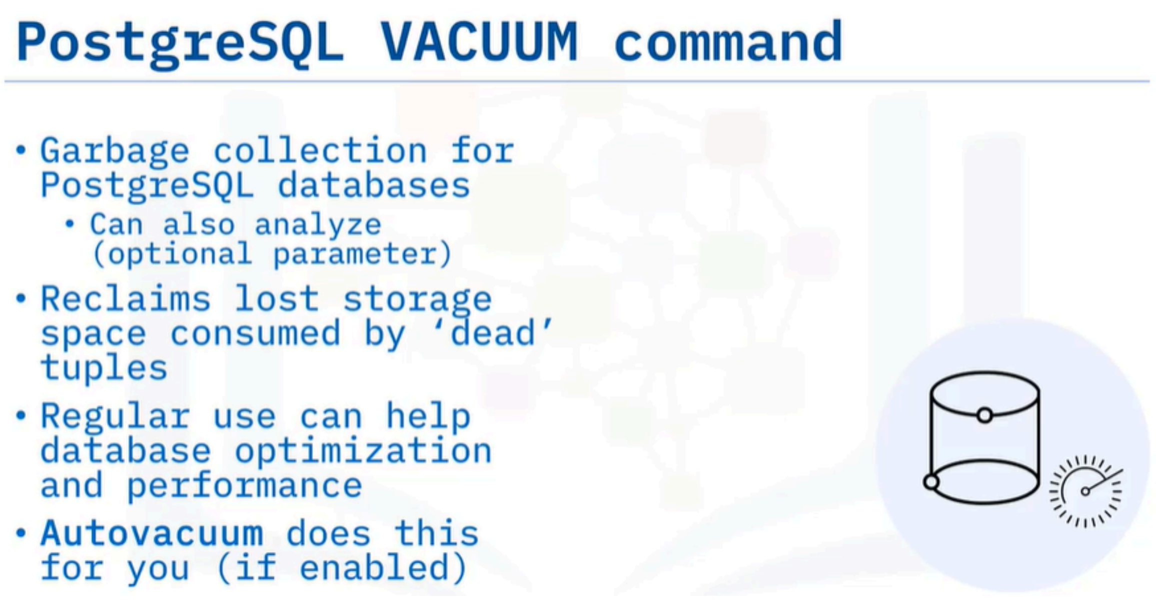
- After significant amount of insert, update, or delete operations, databases can get fragmented
- OPTIMIZE TABLE reorganizes physical storage of table data and associated index to reduce storage space and improve efficiency
- Requires SELECT and INSERT privileges

```
OPTIMIZE TABLE accounts, employees, sales;
```

- Optimizes three tables in one operation
- You can also use phpMyAdmin graphical tool

- If your MySQL database receives a significant number of insert, update, or delete operations, it can lead to fragmentation of your data files, meaning that a lot of unused space is wasted, which in turn can impact the database’s performance. Therefore, it is recommended that database admins defrag their MySQL tables on a regular basis. In MySQL, the OPTIMIZE TABLE command reorganizes the physical storage of table data and associated index data, to reduce storage space and improve input/output efficiency when accessing a table. To use the OPTIMIZE TABLE command, you need to have SELECT and INSERT privileges for the table you are working on. This example would optimize three different tables in one operation.

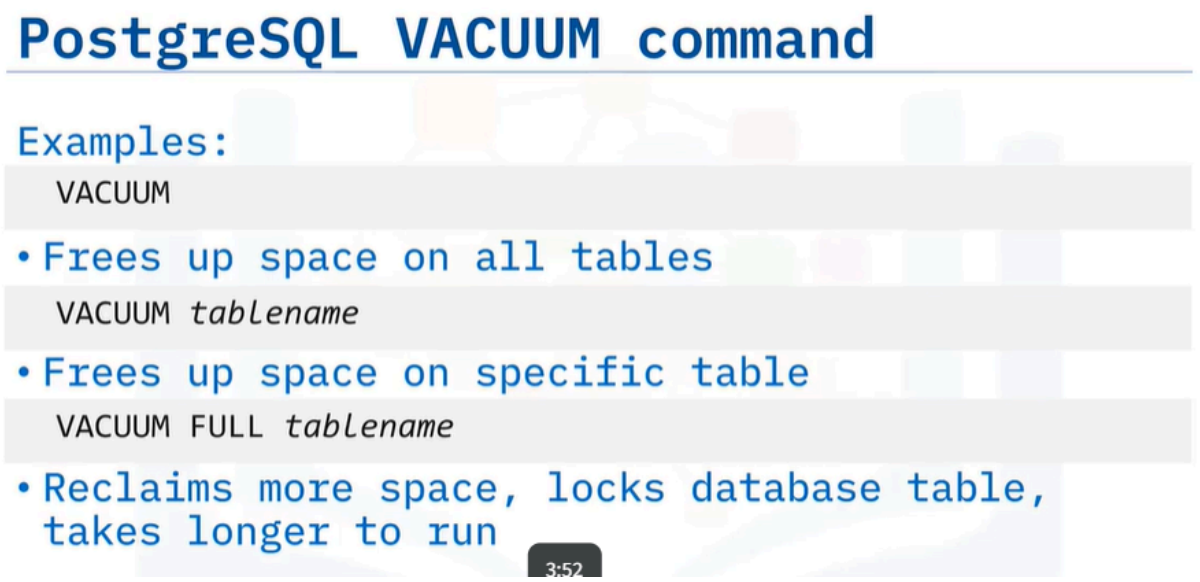
- You could also use a graphical tool such as phpMyAdmin to optimize tables in MySQL.



PostgreSQL VACUUM command

- Garbage collection for PostgreSQL databases
  - Can also analyze (optional parameter)
- Reclaims lost storage space consumed by ‘dead’ tuples
- Regular use can help database optimization and performance
- **Autovacuum** does this for you (if enabled)

- The VACUUM command can be used on your PostgreSQL databases to perform garbage collection, and optionally, analysis tasks. VACUUM reclaims lost storage space consumed by ‘dead’ tuples, which are not physically removed from their database tables after being deleted or made obsolete by an update during routine PostgreSQL operations. Regularly reclaiming this lost storage space can help improve your databases’ overall performance in PostgreSQL. Please note that if the autovacuum feature is enabled, then PostgreSQL will automate the vacuum maintenance process for you.


PostgreSQL VACUUM command

Examples:

```
VACUUM
```

- Frees up space on all tables

```
VACUUM tablename
```

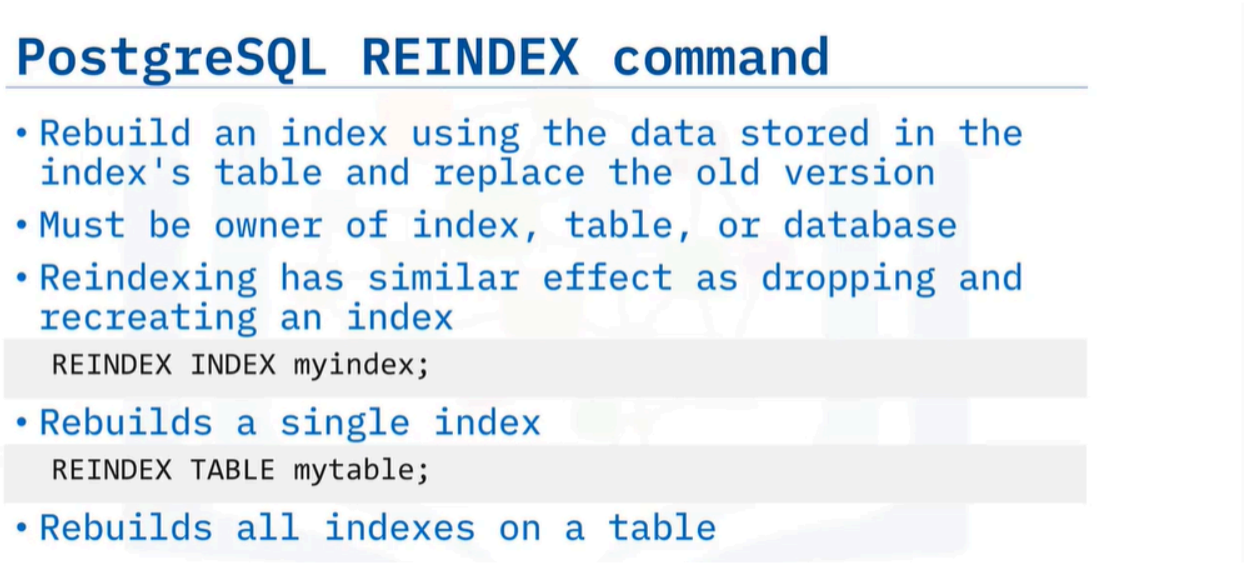
- Frees up space on specific table

```
VACUUM FULL tablename
```

- Reclaims more space, locks database table, takes longer to run

3:52

- In these examples, when you run VACUUM with no parameters the command will free up space on every table in every database that the current user has the rights to perform this command on. If you specify the name of a table, then the command will only process and free up space on that table. Without the FULL parameter, the VACUUM command can run alongside normal read and write table operations, as it does not require an exclusivity lock, and any reclaimed space is retained for future reuse within the table it was vacuumed from. However, if you specify the FULL parameter, the command will create a complete copy of the table contents and write it to disk with no unused space, which allows the space to be reclaimed by the operating system. So, using the FULL parameter can reclaim more space for you, but it takes much longer to run and requires an exclusivity lock on each table while it is being processed.



PostgreSQL REINDEX command

- Rebuild an index using the data stored in the index's table and replace the old version
- Must be owner of index, table, or database
- Reindexing has similar effect as dropping and recreating an index

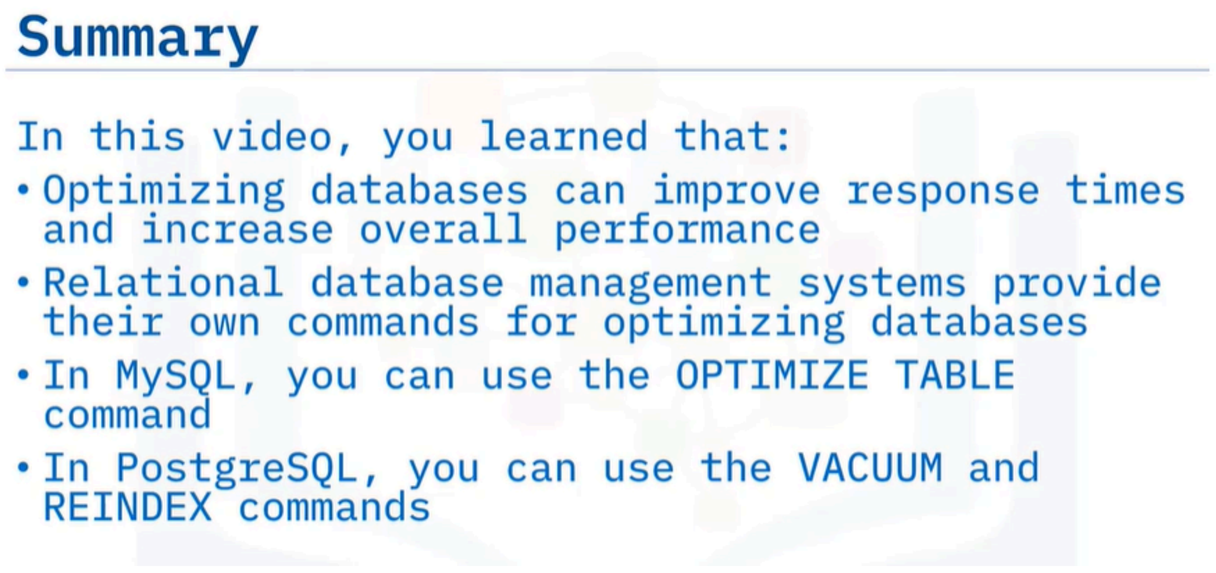
```
REINDEX INDEX myindex;
```

- Rebuilds a single index

```
REINDEX TABLE mytable;
```

- Rebuilds all indexes on a table

- In PostgreSQL, you can use the REINDEX command to rebuild one or more indexes. REINDEX rebuilds an index using the data stored in the index's table and replaces the previous, older, version of the index. It can be used when software bugs or hardware failures have corrupted an index, or when an index has become bloated and contains numerous empty, or nearly empty, pages. You need to be the owner of the index, or the table, or the database in order to reindex it. When you use REINDEX, it rebuilds the index contents from scratch, and therefore it has a very similar effect as dropping and recreating an index. However, the way that locks on reads and writes works on them is different. Here are some REINDEX usage examples. The first example will rebuild the index named myindex. The second example will rebuild all the indexes on the table named mytable



Summary

In this video, you learned that:

- Optimizing databases can improve response times and increase overall performance
- Relational database management systems provide their own commands for optimizing databases
- In MySQL, you can use the OPTIMIZE TABLE command
- In PostgreSQL, you can use the VACUUM and REINDEX commands