Postgres Database Performance

A Downside of Indexes: Space

One of the downsides of creating an index in PostgreSQL is that indexes take up space. The index data structures can sometimes take up as much space as the database itself.

A Downside of Indexes: Data Entry and Updates

One of the downsides of creating an index in PostgreSQL is that indexes slow down data entry or modification. Whenever a new row is added that contains a column with an index, that index is modified as well. If you are adding a large amount of data to an existing table, it may be better to drop the index, add the data, and then recreate the index rather than having to update the index on each insertion.

Dropping an Index

In PostgreSQL, the `DROP INDEX` command can be used to drop an existing index. Indexes are dropped according to their name.

```
DROP INDEX IF EXISTS <index_name>;
```

EXPLAIN ANALYZE

In PostgreSQL, the keywords `EXPLAIN ANALYZE` can be used to get the query plan on for scripts. This can be used to see the runtime of a query.

```
EXPLAIN ANALYZE SELECT * FROM customers
WHERE first_name = 'David';
```

Benefits of an Index

In a relational database like PostgreSQL, indexes are used to improve the speed of searching and filtering at the cost of slower inserts, updates, and deletes.

Multicolumn Indexes

In PostgreSQL, multicolumn indexes allow for more than one column to be used in combination as an index on a table.
The syntax to do this is identical to adding a single-column index, except multiple columns can be given in a comma-separated list.

```
CREATE INDEX
customers_last_name_first_name_idx ON
customers(last_name, first_name);
```

Database Size

In PostgreSQL, to see the size of the database, you can use `pg_size_pretty` and `pg_total_relation_size` . This is a useful command to use before and after creating an index to see how much space the index is using.

```
SELECT pg_size_pretty
(pg_total_relation_size('<table_name>'));
```

The pg_index Table

In PostgreSQL, the `pg_indexes` table contains information about what indexes exist on a table. `pg_indexes` can be queried like any other table.

```
SELECT *
FROM pg_indexes
WHERE tablename = '<table_name>';
```

Indexes with WHERE

Indexes are used by the database server to increase the speed when searches for specific records are performed. This is often used in the `WHERE` clause(s) and when two tables are joined together on their `ON` clause(s).

```
SELECT * FROM customers WHERE last_name
= 'Jones';
```

Indexes and Primary Keys

In PostgreSQL, when a primary key is created on a table, the database server automatically creates a Unique Index on that table.

```
-- Assuming there is a customers table
with a customer_id field, this will create
an index on customer_id.
ALTER TABLE customers ADD PRIMARY KEY
(customer_id);
```

Clustered Indexes

A PostgreSQL database can have two types of indexes - clustered and non-clustered.
However, a table can only have one clustered index. This index physically changes the storage of the data in long term memory whereas a non-clustered index is a separate organization that references back to the original data.

Non-Clustered Indexes

In PostgreSQL, a table can have multiple non-clustered indexes. These indexes create a key(s) and a pointer back to the table where the rest of the information can be found.

The CLUSTER Keyword

In PostgreSQL, the `CLUSTER` keyword can be used to create a new clustered index on a table, or recluster a table already setup with an index.

```
-- Defining which existing index should be used as the clustered index for a given table
CLUSTER products USING products_product_name_idx;


-- Clustering a single table
CLUSTER products;

-- Clustering all tables in the database
CLUSTER;
```

Avoiding Secondary Lookup

In PostgreSQL, if all columns being used in a query are part of an index, no secondary lookup is done.

```
CREATE INDEX customers_last_name_first_name_email_address_idx
ON customers (last_name, first_name, email_address);

-- Because the three columns used in the query are in the index, no secondary lookup needs to happen.
SELECT first_name, last_name, email_address
FROM customers
WHERE last_name = 'Smith';
```

Partial Indexes

PostgreSQL allows for indexing on a subset of a table using the WHERE clause. These are called Partial Indexes.

```
CREATE INDEX <index_name>
ON <table_name> (<column>)
WHERE <condition>;
```

Ordered Indexes

PostgreSQL can use indexes to return results in order without a separate step to sort. This is done by specifying the order (`ASC` or `DESC`) you want the index to be in when you create the index.

```
-- Ascending order
CREATE INDEX <index_name> ON <table_name> (<column_name> ASC)
```

Combining Indexes

PostgreSQL can use multiple indexes together in a single query. This is done automatically by the system. A database engineer must consider whether to make multiple single indexes that are combined, a multicolumn index, or all combinations of single and multicolumn indexes.

```
-- Option one - one index on both columns
CREATE INDEX
customers_last_name_first_name_idx ON
customers (last_name, first_name);

-- Option two - two indexes. One on each
column
CREATE INDEX customers_last_name_idx ON
customers (last_name);
CREATE INDEX customers_first_name_idx ON
customers (first_name);
```

Indexes With Functions

A column Index is not limited to just a column reference, it can also be a function or scalar expression computed from one or more columns.

```
-- This uses the LOWER function to ensure
only one value of a string can be added to
a table, regardless of capitalization. i.e
'name@test.com' and 'NAME@test.com' will
be considered the same email address.
CREATE UNIQUE INDEX
customers_email_address_lower_unique_idx
ON customers(LOWER(email_address));
```