# 12-Indexing

*Author: Vincent Lau*

*Note: This material is intended for educational purposes only. All rights reserved. Any unauthorized sharing or copying of this material, in any form, to any individual or party, for any use without prior permission, is strictly prohibited.*

## Learning Objectives

- Understand the purpose of creating Index for tables
- Understand the ways of implementing indexes
- Understand the advantages and disadvantages of creating indexes

## Introduction

Indexing is a **database optimization technique** used to **improve the speed of data retrieval operations, such as `SELECT` queries**. Indexes are data structures that provide a quick way to look up rows based on the values in one or more columns. They work similar to the index in a book, allowing the database system to jump directly to the relevant data instead of scanning the entire table.

Here's how you can create and use indexes in MySQL and PostgreSQL:

## Create Index with Table

Suppose you have a table named "employees" with columns "employee_id", "dept_code", "first_name", "last_name" and "salary".

Indexes can be single-column or multi-column. The above example is a single-column index. To create a multi-column index, you can specify multiple column names within the parentheses.

PostgreSQL also supports different types of indexes, such as B-tree, Hash, and GiST indexes, each with its own use cases and benefits.

You can create an index on the "dept_code" column like this:

```sql
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    dept_code VARCHAR(5),
    salary DECIMAL(10, 2), -- same as NUMERIC(10, 2)
    INDEX idx_emp_dept (dept_code)
);
```

## Create Index seperately

Using the same "employees" table, here's how you would create an index on the "dept_code" column in PostgreSQL:

```sql
-- Create an index in MySQL
CREATE INDEX idx_emp_dept ON employees (dept_code);
```

## Using Indexes

Once the index is created, the database system will use it to speed up queries involving the indexed column. For example:

```sql
-- Query using the index in MySQL
SELECT * FROM employees WHERE dept_code = 'HR';
```

Creating indexes in database tables offers both advantages and disadvantages. Let's explore them:

## Why Indexes

### Advantages

#### Improved Query Performance

Indexes speed up data retrieval operations, especially for SELECT queries. Instead of scanning the entire table, the database can quickly locate rows that match the query conditions.

#### Faster Sorting and Grouping

Indexes help with sorting and grouping operations, making aggregate functions like SUM, COUNT, AVG, and ORDER BY more efficient.

#### Enhanced JOIN Operations

When joining multiple tables, indexes on join columns significantly reduce the time required to match and combine rows.

#### Efficient Unique Value Enforcement

Unique indexes ensure that a column or combination of columns has unique values, preventing duplicate data.

#### Primary Key and Foreign Key Constraints

**Primary keys and foreign keys are typically implemented using indexes**, enforcing data integrity and relationships between tables.

#### Reduction in Disk I/O

**Indexes minimize the amount of data read from disk**, as they act as a roadmap to the relevant data, reducing disk I/O operations.

### Drawbacks

#### Increased Storage

Indexes consume disk space, which can be significant for large tables. This may lead to increased storage costs and maintenance overhead.

#### Slower Write Operations

Indexes need to be updated when data is inserted, updated, or deleted. This can slow down write operations (INSERT, UPDATE, DELETE).

### Maintenance Overhead

As data changes, indexes need to be maintained to reflect the changes accurately. This can lead to additional maintenance overhead.

### Complexity

With more indexes, the query optimizer has more options to consider, making query optimization more complex and potentially affecting performance.

### Choosing the Right Columns

Creating indexes on the wrong columns or creating too many indexes can lead to inefficient queries and performance degradation.

### Outdated Statistics

The database's query optimizer relies on statistics about indexes to make informed decisions. If statistics are outdated, query performance may suffer.

## Summay

Given these pros and cons, creating indexes requires careful consideration. It's important to **analyze your database's usage patterns, query workload, and specific requirements before adding indexes**. Creating indexes for frequently queried columns can significantly improve performance, but over-indexing can lead to unnecessary overhead. **Regular monitoring, index maintenance, and periodically reviewing index usage are essential practices for maintaining a well-performing database**.

Keep in mind that while indexes greatly improve query performance, they also have some **downsides**.

Indexes consume disk space and can **slow down write operations (INSERT, UPDATE, DELETE)** because the indexes need to be updated along with the data.

Therefore, it's important to strike a balance between creating indexes for frequently queried columns and managing their impact on overall database performance.

Additionally, the choice of which columns to index and the type of indexes to use depends on the nature of your queries and your specific use case. It's a good practice to analyze query patterns and performance to determine where indexes are most beneficial.