



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

200
1822 - 2022

Enhancing Database Design

Man Geni Addysg Uwch yng Nghymru
Where Higher Education Began in Wales

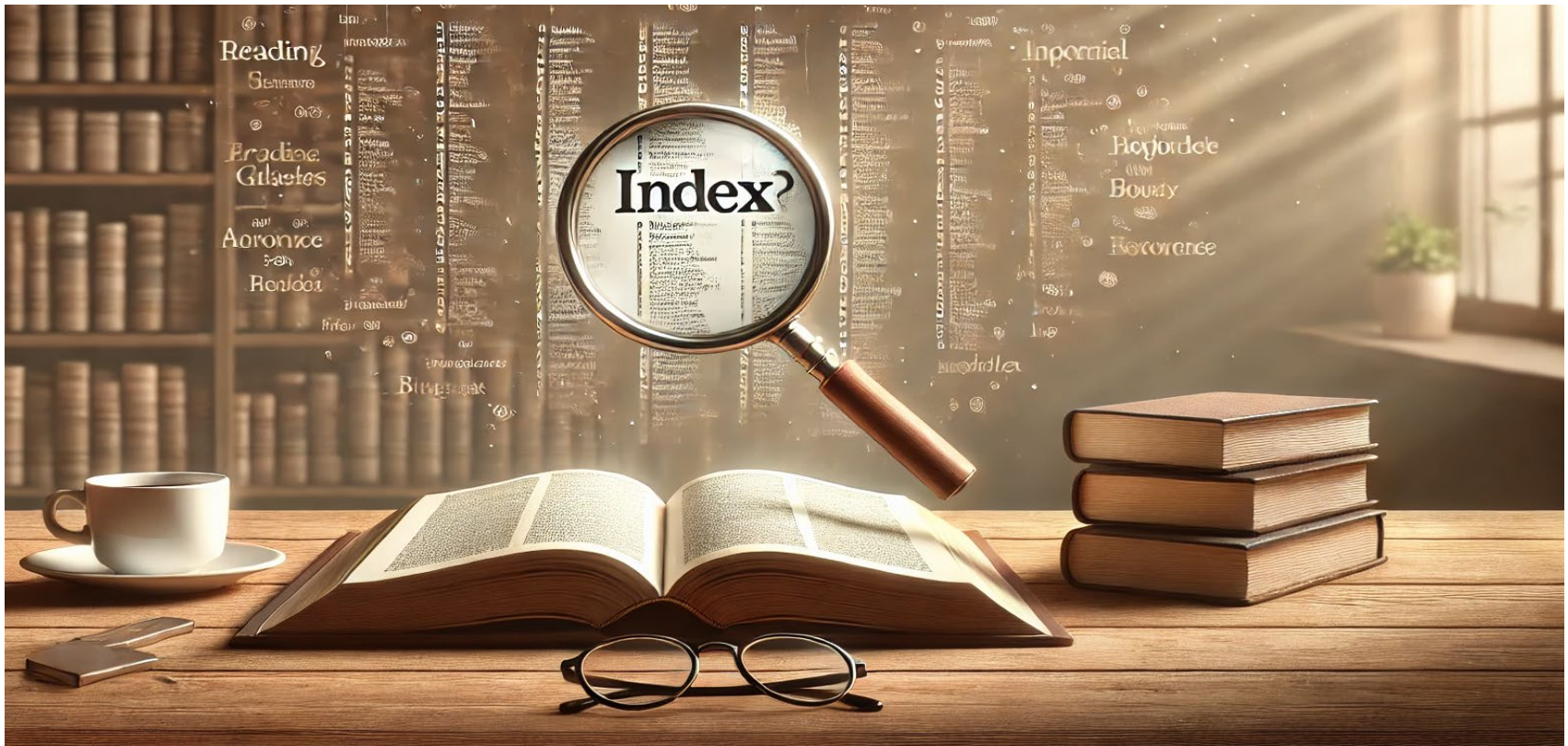
www.uwtsd.ac.uk

Objectives

- Indexing & Performance Tuning in Databases
- How to speed up queries and optimize operations
- Dropping and Managing Indexes

What is an Index?

- An index is a **data structure that improves the speed of data retrieval**



Why Indexing Matters

- Large tables → full table scans are slow
- Performance-critical queries benefit from indexes

```
SELECT * FROM friends WHERE name = 'Zack';
```

friends		
id	name	city
1	Matt	San Francisco
2	Dave	Oakland
3	Andrew	Blacksburg
4	Todd	Chicago
5	Blake	Atlanta
6	Evan	Detroit
7	Nick	New York City
8	Zack	Seattle

Creating an Index for Products table

```
CREATE TABLE products (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(100),  
    category VARCHAR(50),  
    price DECIMAL(10, 2)  
);
```

```
INSERT INTO products (name, category, price)
VALUES
('Laptop', 'Electronics', 999.99),
('Smartphone', 'Electronics', 499.99),
('Desk Chair', 'Furniture', 150.00),
('Coffee Table', 'Furniture', 200.00),
('Headphones', 'Electronics', 89.99),
('Monitor', 'Electronics', 299.99),
('Bookshelf', 'Furniture', 120.00);
```

Creating Indexes

Index Name

```
CREATE INDEX idx_category ON Products(category);
```

Multi-column example:

```
CREATE INDEX idx_category_price ON  
Products(category, price);
```

This index helps
queries filtering by
both category and price

Dropping & Managing Indexes

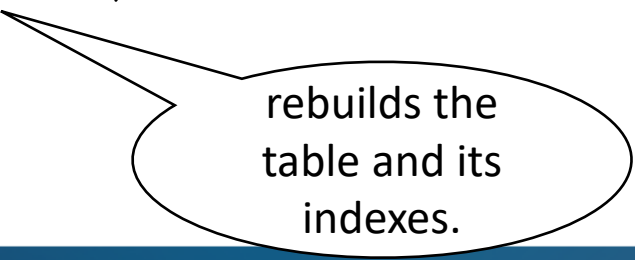
```
DROP INDEX idx_category ON Products;
```

check existing indexes using:

```
SHOW INDEXES FROM Products;
```

Maintenance of indexes:

```
OPTIMIZE TABLE Products1;
```



rebuilds the
table and its
indexes.

Using Indexes in Queries

- Indexes improve the performance of queries that use WHERE, JOIN, ORDER BY, and GROUP BY clauses

Example 1: Filter by category

```
SELECT * FROM Products WHERE  
category='Electronics';
```

Example 2: Sort by price

```
SELECT * FROM Products ORDER BY price DESC;
```

Example 3: Composite index filter

```
SELECT * FROM Products  
WHERE category='Electronics' AND price>100;
```

Performance Considerations

They consume storage and slightly slow down inserts, updates, and deletes because the index must also be updated. They are most useful on:

- Large tables
- Frequently searched columns
- Columns with high selectivity

On small tables, a full scan may actually be faster than using an index.

Further Reading :

Chapter 8 Advanced SQL

Database Systems: A Practical Approach to Design, Implementation, and Management Sixth edition by • Thomas Connolly • Carolyn Begg