Database Theory Guide

A comprehensive guide to database concepts and technologies

Table of Contents

The difference between data and information

Data consists of raw facts, figures, or observations without any context or meaning, such as the numbers 23, 45, 67, 89, and 12. Information, on the other hand, is data that has been processed, organized, and given context to create meaning, like "Test scores: Average is 47, John got the highest score of 89." The key difference is that data is simply a collection of facts, while information transforms those facts into something meaningful that can be understood and used for decision-making.

Metadata is Data about data - information that describes other data

A portion of the metadata the following are the goals:

- Sort and classify data
- Locate data fast
- Comprehend data context
- Manage data effectively

Typical metadata Applications:

- File attributes (size, creation date)
- Information on database schemas; search engine optimisation for websites
- Information about the photo (camera, location, timestamp)
- Monitoring of documents (author, version, keywords)

The following are the types of metadata:

Descriptive

Describes the content of the data

Structural

How the data is arranged

Administrative

When, who made it, and what permission

Database Management System (DBMS)

Software used to store, arrange, and administer data in databases is known as a database management system. Envision it as a smart assistant that helps you save, locate, and arrange your files in a digital filing cabinet.

This works because:

- holds a lot of data and arranges it in tables and relationships
- prevents unwanted access to data
- permits several users to access data at once
- automatically creates a backup of the data

The benefits of DBMS are:

3.1. Organisation of Data

- maintains order and neatness
- Finding what you need is simple.

3.2. Security of Data

- Password protection limits access to certain data
- Several Users
- Data can be accessed simultaneously by numerous users
- No disputes or corrupted data

3.3. Backup of Data

- Data loss is avoided with automatic backups
- Options for recovery in case something goes wrong

3.4. Less Redundancy

- Data is stored once and used everywhere
- No redundant data

3.5. Integrity of Data

- guarantees the accuracy and consistency of the data
- stops incorrect data entry

Typical Instances:

MySQL

widely used for websites

Oracle

utilised by big businesses

Microsoft SQL Server

Applications for businesses

PostgreSQL

A robust and free solution

In summary, DBMS functions similarly to a dependable, safe, and well-organised digital assistant that effectively handles all of your critical data.

Operational vs Analytical Databases

Operational Databases (OLTP)

Handle day-to-day business operations and transactions.

Characteristics:

- Real-time processing
- High volume of small transactions
- Current data focus
- Fast inserts/updates/deletes
- Detailed records
- Normalized structure

Example: E-commerce Order System

```
-- Processing customer orders in real-time
INSERT INTO orders (customer_id, product_id, quantity, order_date)
VALUES (12345, 'LAPTOP001', 1, '2025-09-16 14:30:00');

UPDATE inventory
SET stock_quantity = stock_quantity - 1
WHERE product_id = 'LAPTOP001';
```

Analytical Databases (OLAP)

Support business intelligence, reporting, and data analysis.

Characteristics:

- Batch processing
- Large queries on historical data
- Historical data focus
- Fast complex queries

- Summarized data
- Denormalized structure

Example: Sales Analytics Data Warehouse

```
-- Analyzing sales trends over time
SELECT
  product_category,
  YEAR(order_date) as year,
  MONTH(order_date) as month,
  SUM(total_amount) as monthly_revenue,
  COUNT(*) as total_orders
FROM sales_fact_table
WHERE order_date BETWEEN '2023-01-01' AND '2025-12-31'
GROUP BY product_category, YEAR(order_date), MONTH(order_date)
ORDER BY year, month;
```

Key Differences

Aspect	Operational (OLTP)	Analytical (OLAP)
Purpose	Daily operations	Business analysis
Data Type	Current, detailed	Historical, summarized
Users	Many concurrent users	Few analytical users
Queries	Simple, fast	Complex, time-consuming
Updates	Frequent	Rare (batch loads)
Structure	Normalized	Denormalized
Response Time	Milliseconds	Minutes/Hours
Data Volume	Moderate	Very large

Real-World Examples - Operational Database

Banking: ATM transactions, account balances

Retail: Point-of-sale systems, inventory tracking

Healthcare: Patient records, appointment scheduling

Social Media: User posts, messages, friend requests

Real-World Examples - Analytical Database

Business Intelligence: Sales performance dashboards

Marketing: Customer behavior analysis

Finance: Quarterly revenue reports

Operations: Supply chain optimization analysis

Use Operational DB when:

- Processing real-time transactions
- Need immediate data updates
- Supporting daily business operations
- Serving many concurrent users

Use Analytical DB when:

- Creating reports and dashboards
- Analyzing historical trends
- Making strategic business decisions

- Processing large amounts of data for insights

Bottom Line: Operational databases keep your business running day-to-day, while analytical databases help you understand and improve your business through data insights.

What is NoSQL?

NoSQL (Not Only SQL) databases are designed to handle large volumes of unstructured, semi-structured, or rapidly changing data that doesn't fit well into traditional relational database tables.

Types of Data Best Suited for NoSQL

5.1. Unstructured Data

- Text documents, emails, social media posts
- Images, videos, audio files
- Log files, sensor data
- Web content, articles, blogs

5.2. Semi-Structured Data

- JSON documents
- XML files
- CSV files with varying columns
- API responses

5.3. High-Volume, High-Velocity Data

- Real-time analytics data
- IoT sensor readings
- Social media feeds
- Gaming telemetry

5.4. Flexible Schema Data

- Product catalogs with varying attributes
- User profiles with different fields
- Content management systems
- Configuration data

Types of NoSQL Databases & Use Cases

1. Document Databases

Store data as documents (JSON, BSON)

Best For:

- Content management systems
- Product catalogs
- User profiles
- Real-time analytics

Examples:

MongoDB CouchDB Amazon DocumentDB

Use Case Example:

```
// E-commerce product catalog
{
  "product_id": "12345",
  "name": "Wireless Headphones",
  "category": "Electronics",
  "specifications": {
    "battery_life": "20 hours",
    "connectivity": ["Bluetooth 5.0", "USB-C"],
    "colors": ["Black", "White", "Blue"]
  },
  "reviews": [
    {"user": "john_doe", "rating": 5, "comment": "Great sound quality"}
  ]
}
```

2. Key-Value Stores

Simple key-value pairs

Best For:

- Caching systems
- Session management
- Shopping carts
- Configuration storage

Examples:

Redis Amazon DynamoDB Riak

Use Case Example:

```
// User session data
user:12345:session →
{"login_time": "2025-09-16T14:30:00", "cart_items": 3}

user:12345:preferences →
{"theme": "dark", "language": "en"}
```

3. Column-Family (Wide-Column)

Data stored in column families

Best For:

- Time-series data
- IoT applications
- Analytics workloads
- Large-scale logging

Examples:

Cassandra HBase Amazon SimpleDB

4. Graph Databases

Nodes and relationships

Best For:

- Social networks
- Recommendation engines
- Fraud detection
- Knowledge graphs

Examples:

Neo4j Amazon Neptune ArangoDB

Use Case Example:

```
// Social network relationships
(Person: John) -[FRIENDS_WITH]-> (Person: Mary)
(Person: Mary) -[LIKES]-> (Product: iPhone)
(Person: John) -[WORKS_AT]-> (Company: TechCorp)
```

Real-World Examples

Netflix

Uses Cassandra for streaming data and recommendations

Facebook

Uses MongoDB for user profiles and social graph

Uber

Uses Redis for real-time location tracking

LinkedIn

Uses graph databases for professional networks

Bottom Line: NoSQL databases excel when you need to handle large volumes of diverse, rapidly changing data with high performance requirements and flexible schemas. Choose NoSQL when your data doesn't fit neatly into tables or when you need massive scale and speed.

Serverless DBMS: SQLite

What is SQLite?

SQLite is a serverless, file-based database that doesn't require any server setup or configuration. The entire database is stored in a single file on your computer.

Key Characteristics:

No server needed

Just a file on your device

Zero configuration

Works immediately after installation

Self-contained

Everything in one database file

Lightweight

Small footprint and fast performance

Code Example

```
import sqlite3

# Create/connect to database (file created automatically)
conn = sqlite3.connect('my_app.db')
cursor = conn.cursor()

# Create table
cursor.execute('''
  CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY,
        name TEXT NOT NULL,
        email TEXT UNIQUE
    )
''')

# Insert data
cursor.execute("INSERT INTO users (name, email) VALUES (?, ?)",
               ("John Doe", "john@example.com"))

# Query data
cursor.execute("SELECT * FROM users")
results = cursor.fetchall()
print(results)

conn.commit()
conn.close()
```

Bottom Line: SQLite is ideal when you need a simple, fast, and maintenance-free database solution for applications that don't require complex server infrastructure.

Best choice for: Local storage, development, testing, small applications, and embedded systems.

ACID Properties in DBMS

What is ACID?

ACID is a set of properties that guarantee reliable processing of database transactions. These properties ensure data integrity and consistency even in the face of errors, power failures, or other system issues.

The Four ACID Properties

1. Atomicity

All operations in a transaction must complete successfully, or none at all.

Key Concept: "All or Nothing"

```
-- Bank transfer transaction
BEGIN TRANSACTION;
UPDATE accounts
SET balance = balance - 100
WHERE account_id = 'A001'; -- Debit

UPDATE accounts
SET balance = balance + 100
WHERE account_id = 'B002'; -- Credit
COMMIT;

-- If ANY step fails, the ENTIRE transaction is rolled back
```

2. Consistency

Database must remain in a valid state before and after any transaction.

Key Concept: "Data Integrity Rules"

```
-- Business rule: Account balance cannot be negative
CREATE TABLE accounts (
  account_id VARCHAR(10) PRIMARY KEY,
  balance DECIMAL(10,2) CHECK (balance >= 0)
);

-- This transaction will FAIL if it violates the rule
```

3. Isolation

Concurrent transactions should not interfere with each other.

Key Concept: "Transactions are Independent"

```
-- Two users trying to book the last concert ticket simultaneously
-- User A Transaction:
BEGIN TRANSACTION;
SELECT available_seats FROM concerts WHERE concert_id = 'C001';
-- Returns: 1
UPDATE concerts SET available_seats = 0 WHERE concert_id = 'C001';
COMMIT;
```

4. Durability

Once a transaction is committed, changes are permanent even if system crashes.

Key Concept: "Permanent Storage"

```
-- After this transaction completes successfully
BEGIN TRANSACTION;
INSERT INTO orders (order_id, customer_id, total)
VALUES ('ORD001', 'CUST123', 299.99);
COMMIT; -- Changes are now PERMANENT
```

ACID properties ensure that database transactions are reliable, consistent, and safe. They prevent data corruption, ensure business rules are followed, and guarantee that your data remains intact even during system failures or high-traffic situations.

Database Theory Guide - Comprehensive overview of database concepts and technologies