EXPLORE AI ACADEMY

SQL environments

SQL in the wild

Practical example

In order to understand the SQL development, testing, and production environments, we will **explore the SQL database creation** of a dynamic e-commerce platform named "ShopEase".

Overview

ShopEase is a prominent online marketplace offering a wide array of products to its customers. With its user-friendly interface and secure transactions, it has become a preferred destination for online shoppers. The platform allows users to browse products, add them to their carts, and seamlessly complete purchases.

Millions of users of this app conduct everyday transactions and all of this information must be kept in a safe location that is also conveniently accessible.

The company must employ databases to effectively organise and retrieve this data.

Practical example

In order to understand the SQL development, testing, and production environments, we will **explore the SQL database creation** of a dynamic e-commerce platform named "ShopEase".

The solution

01. Implement an efficient database schema

ShopEase can use SQL to manage, store, retrieve, and analyse data in databases, streamlining operations to improve user experiences and decision-making.

02. Integrate SQL in the application production environment

Programming languages like Python, Java, or PHP are used to integrate SQL queries within application code.

Solution: Implement an efficient database schema

In modern software development, **different environments play a crucial role** in ensuring the **quality and reliability** of a database.

These environments are **tailored to specific stages of the development lifecycle** and are designed to address various needs and challenges.

In SQL-based projects, such as **database-driven applications**, having **distinct SQL environments becomes essential** for efficient **development, testing, and deployment.**

Why do we use different SQL environments?

Development, testing, and deployment environments are essential in the lifecycle of a database in order to **ensure controlled progression, minimise risk, and maintain data integrity**.

Risk mitigation

Changes can be thoroughly tested before reaching the production environment, reducing the likelihood of critical issues.

Isolation

Different environments **prevent interference between development, testing, and production stages** of database creation, **ensuring that changes in one environment do not accidentally affect another**. It's like creating a **safe zone for each step**, ensuring that changes are contained until they're ready to be shared with the rest of the world.

Controlled changes

Environments provide **controlled channels for introducing new features or modifications,** ensuring that **changes made are well-thought-out, thoroughly tested, and won't unexpectedly affect the users** or the system's stability.

SQL environments: Development

The SQL development environment serves as the **foundation for designing**, **creating**, and **modifying** the database schema and SQL queries. This environment is also referred to as "DEV".

In this environment, developers have the **freedom to experiment**, **make changes**, and **iterate** without affecting other stages of the development lifecycle.

Example:

ShopEase developers create tables optimised for product information storage, customer data, and order histories. They write SQL scripts to create and modify tables, define relationships, and set indexes. They also implement features such as caching mechanisms to optimise frequent queries for product listings and customer carts.

Characteristics

Developers can work on their tasks **independently** without affecting the production system.

Frequent **changes** and **experimentation** are encouraged to refine database structures and query performance.

Rapid **iteration** and **prototyping** of SQL code for upcoming features.

SQL environments: Testing

The SQL test environment is an **exact or representative copy of the production environment**, set up specifically for rigorous testing and validation of database changes. This environment is also referred to as the QA environment.

This environment allows development teams to **catch** and **resolve issues** before changes are deployed to the live system.

Example:

With the new database schema constructed in the development environment, it's moved to the test environment for comprehensive testing. Testers verify the data integrity of the tables, ensuring that relationships and constraints are maintained. They generate extensive test scenarios, simulating high-traffic and complex queries to gauge performance. Developers monitor query execution plans, optimising indexes and refining SQL queries for efficiency.

Characteristics

This is a **near-identical replica** of the **production environment** to mimic real-world scenarios.

Rigorous testing of the new database schema, query performance, and data integrity is done.

Identification and **resolution of issues** that might not be apparent in the development environment.

SQL environments: Production

The SQL production environment is the **live environment** that serves users, applications, and critical business processes.

This environment is where the real action happens! **High availability**, **performance**, and **data integrity** are paramount.

Example:

Following thorough testing and validation, the optimised database schema is ready for deployment in the production environment. Developers closely monitor the database's performance, checking for any anomalies or bottlenecks. They may further optimise the database based on real-world usage patterns, ensuring that the platform's users continue to enjoy a seamless shopping experience.

Characteristics

This environment can **actively handle user interactions**, **transactions**, and application **requests**.

It ensures **accurate** and **reliable data storage**, **retrieval**, and **updates**.

It is an optimised environment that delivers **fast response times** and **minimal downtime**.

Due to **data integrity and maintainability rules** there are some **limitations** in this environment. We may not have the same **access to the database** and the **types of queries we can execute** are limited. The database must be **end-user-ready** at all times, therefore requiring very **limited tampering**.

SQL environments

Solution: Integrate SQL in the production environment

The SQL code is **integrated** within the application code in order to **create** and **maintain** a **reliable data-driven system**.

The integration of SQL with application code:

01. The application code will connect to the server that is responsible for hosting and managing the production database in SQL.

02. Code will be written to run the application using the application’s preferred language (such as Python, Java, or PHP).

03. SQL queries will be included within the code to extract or manage the data in SQL.

```
# Establish a connection to the database
conn = sqlite3.connect(“shopease.db”)                01.
cursor = conn.cursor()

# Retrieve product information                              02.
def get_product_info (product_id):
        query = f”SELECT name, price,stock FROM
                products WHERE id = {product_id}”
        cursor.execute(query)
        product_data - cursor.fetchone()
        return product_data

# Process an order
def process_order (order_id, product_id, quantity):
        #check product availability
        product_data = get_product_info(product_id)
        if product_data and product_data[2] >= quantity:
                # Update stock
                new_stock = product_data [2] - quantity
    03.         Update_query = f”UPDATE products SET stock
                        = {new_stock} WHERE id = {product_id}”
                cursor.execute(update_query)
```

Best practices for production-level data management

Deploying SQL databases in production environments **requires careful planning** and **adherence to best practices** to ensure performance, security, and maintainability. Here are some SQL best practices for production deployments:

01. Database security

To ensure database security, use **strong passwords**, **regularly update database management systems** to address security vulnerabilities, use **encryptions**, and implement **role-based access controls**.

02. High availability and scalability

To ensure high and constant availability of the database, implement **redundancy and failover mechanisms**, use load balancers to **distribute database traffic**, and scale the database horizontally by **adding more servers when needed** through constant **performance monitoring**.

03. Performance tuning

Optimise database performance by **monitoring schema design and performance**, making queries and procedures **run as fast as possible**, and **regularly analysing and optimising slow-running queries**.

04. Disaster recovery

Maintain regular **automatic backups** of the database and **test backup and restore procedures** to ensure data recoverability in the event of a disaster.

End-user access to production-level data

SQL is a powerful tool **used in a wide range of industries** to manage and analyse data. Here are some real-world examples of how end users interact with production-level SQL data.

Financial institutions

Financial institutions utilise SQL to manage account data, transactions, and risk assessments, enabling decision-making on things like loan approvals, interest rates, and fraud detection.

Healthcare

For better decision-making in treatment plans, resource allocation, and research priorities, hospitals use SQL to store patient records, treatment histories, and medical imaging data.

Manufacturing

A manufacturing company uses SQL to handle supply chain data, track equipment maintenance, and monitor production lines, enabling well-informed decisions and optimisation.

Retail

Retail chains use SQL to handle store inventories, customer loyalty programs, and sales data, facilitating strategic decision-making and improving consumer experiences.