Database Systems

Embedded SQL

Muhammed-Ugur Karagülle for Prof. Dr. Agnès Voisard

Freie Universität Berlin, Department of Mathematics and Computer Science,Institute of Computer Science,
Databases and Information Systems Group,
Prof. Dr. Agnès Voisard

2025

Outline

Recap of Relational Databases

- A relational database organizes data into tables (or ’relations’) where each table represents an entity (like a person, place, event, etc.).
- Each table has rows (also known as ’tuples’ or ’records’) and columns (’attributes’). Each row represents an instance of the entity.
- Tables can be linked, or ’related’, by a common attribute, allowing us to combine data across tables. This is where the term ’relational’ comes from.
- Key concepts include primary key (a unique identifier for records in the table), foreign key (a field in a table that uniquely identifies a row of another table), and indexing (a data structure to improve the speed of data retrieval operations).

Example of a Relational Database

Students Table:

ID	Name	Major
1	Alice	Computer Science
2	Bob	Bioinformatics
3	Charlie	Information Systems

Courses Table:

Course ID	Course Name	Student ID
19301501	Database Systems	1
19303811	Project Seminar Data Management	2
19318412	Software Project Data Management	3
19328211	New Trends in DB	3

Quick Recap of SQL

- SQL stands for Structured Query Language. It is used to communicate with and manipulate databases.
- SQL is used to insert, search, update, and delete database records. SQL can also do lots of other operations including optimizing and maintenance of databases.
- Common SQL commands include ”SELECT”, ”INSERT”, ”UPDATE”, ”DELETE”, ”CREATE”, and ”DROP”.
- SQL operations can be divided into two main categories: DDL (Data Definition Language) which deals with database schemas and descriptions, and DML (Data Manipulation Language) which deals with data manipulation, and includes most common SQL statements such SELECT, INSERT, etc.

Example of SQL commands


```
-- DDL command
CREATE TABLE Students (
    ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Major VARCHAR(100)
);

-- DML commands
INSERT INTO Students (ID, Name, Major) VALUES
            (1, 'Alice', 'Computer Science');
SELECT * FROM Students WHERE Major = 'Computer Science';
UPDATE Students SET Major = 'Mathematics' WHERE Name = 'Alice';
DELETE FROM Students WHERE ID = 1;
```

- Standalone SQL refers to SQL queries executed directly against a database, typically through a command-line interface or a dedicated SQL software.
- These queries are standalone as they are not embedded within a larger programming language.

Example

SQL queries executed in a database management system like MySQL or PostgreSQL.

Variations of SQL Usage - Static SQL

- Static SQL refers to SQL statements that are predefined and hard-coded in an application before it is compiled.
- These queries do not change each time the program is executed, leading to improved efficiency.

Example

A query that pulls the same type of report every time the program is executed.

- Dynamic SQL allows for SQL statements to be created and modified on the fly during runtime.
- This provides more flexibility, allowing the program to construct different queries based on user input or program context.

Example

An e-commerce site's search function that generates different SQL queries based on the user's input.

- Embedded SQL allows for SQL statements to be incorporated in a high-level programming language.
- This provides a bridge between the database and the programming language, allowing the language to directly manipulate the database within the context of a larger program.

Example

A Python program that uses SQL queries to interact with a PostgreSQL database.

- Embedded SQL is a method of combining the computing power of a high-level programming language and the database manipulation capabilities of SQL.
- Embedded SQL provides a bridge between a database and a programming language to manipulate the database directly.

Languages that can incorporate Embedded SQL

- Various programming languages can incorporate Embedded SQL. These include C, C++, Java, Python, Haskell, and many others. The specifics of embedding SQL can vary between these languages.
- There are also numerous libraries and frameworks, such as JDBC for Java or SQLAlchemy for Python, which can facilitate the use of embedded SQL.

Benefits of Embedded SQL

- **Unified development environment:** Same environment for application logic and data manipulation.
- **Control structures:** Integration in control structures of the host language.
- **Error handling:** Error handling mechanisms and utilities of the host language to manage SQL execution errors.
- **Variable integration:** Dynamic variables from the host language to use with the SQL statements.
- **Database connectivity:** Built-in or easily accessible libraries for handling database connection, SQL command execution, and retrieval of results.

SQL Injection

- SQL Injection is a type of security vulnerability that allows an attacker to inject malicious SQL code into a query.
- This could lead to unauthorized viewing of data, data manipulation, or even data loss.
- To prevent SQL injection, parameterized queries or prepared statements can be used.

Example of SQL Injection

- Consider a login form where a user enters a username and password.
- A common, yet insecure way to verify the credentials might be:

```
query = "SELECT * FROM Users WHERE
username='" + username + "' AND password='" + password + "';"
```

An attacker could use SQL injection in the username field by entering: `admin'; --`

This would modify the query to:

```
"SELECT * FROM Users WHERE
username='admin'; -- AND password='';"
```

Everything after - - is a comment, so it nullifies the password check.

- Parameterized queries or prepared statements are a common way to prevent SQL injection.
- They separate SQL code from data, preventing the attacker from manipulating the query structure.
- An example using Python's psycopg2 module:

```
1 query = "SELECT * FROM Users WHERE username=%s AND password
      =%s;"
2 params = (username, password)
3 cursor.execute(query, params)
```

Source Code 1: Preventing SQL Injection

Python Embedded SQL Example (1)


```
1   import psycopg2 #PostgreSQL database adapter for Python
2   # Establish connection with the database
3   conn = psycopg2.connect(
4       dbname="your_database_name",
5       user="your_username",
6       password="your_password",
7       host="your_host_name_or_ip_address",
8       port="your_port_number"
9   )
10  # Create a cursor object. This will be used to execute
    SQL commands.
11  c = conn.cursor()
12
```

Source Code 2: Python Embedded SQL Example (Part 1)

Python Embedded SQL Example (2)


```
13      # Create table
14      c.execute(''' CREATE TABLE EMPLOYEE
15      (FName text, LName text, SSN text, BDate text, Address
        text, Salary real, BossSSN text, NumDept integer)
16      ''')
17      # Insert a row of data
18      c.execute(""" INSERT INTO EMPLOYEE VALUES
19      ('John', 'Doe', '123-45-6789', '1980-01-01', '123 Elm St
        ', 55000.00, '987-65-4321', 1)
20      """)
21      conn.commit() # Save (commit) the changes
22      conn.close() # Close the connection
23
```

Source Code 3: Python Embedded SQL Example (Part 2)

```
1  import java.sql.*; //Import classes for the connection
2  public class Main {
3      public static void main(String[] args) {
4          // Define the database connection string
5          String url = "jdbc:postgresql://
6          your_host_name_or_ip_address:your_port_number/
7          your_database_name";
8          // Provide your database credentials
9          String user = "your_username";
10         String password = "your_password";
11
```

Source Code 4: Java Embedded SQL Example (Part 1)

Java Embedded SQL Example


```
12          try {
13              Connection conn = DriverManager.getConnection(url, user,
14              password);// Establish connection with the database
15              String query = "INSERT INTO DEPENDENT (Name, Gender, BDate,
16              Relation, ESSN) VALUES (?, ?, ?, ?, ?)"; // Prepare SQL
17              statement
18              PreparedStatement pstmt = conn.prepareStatement(query);
19              pstmt.setString(1, "Jane Doe");
20              pstmt.setString(2, "F");
21              pstmt.setDate(3, java.sql.Date.valueOf("2010-01-01"));
22              pstmt.setString(4, "Daughter");
23              pstmt.setString(5, "123-45-6789");
24              pstmt.executeUpdate();// Execute SQL statement
25          } catch (SQLException e) {
26              e.printStackTrace();
27          } } } //closing brackets catch, try, and main
28
```

Source Code 5: Java Embedded SQL Example (Part 2)

Advantages of Embedded SQL

- **Efficiency**: Embedded SQL can provide improved performance over standalone SQL, as the database system can optimize the static SQL statements during the precompilation process.
- **Usability**: Embedded SQL allows developers to use the SQL within a familiar programming language environment.
- **Flexibility**: Embedded SQL supports both dynamic and static SQL. Dynamic SQL can be generated at runtime based on the conditions, improving flexibility.
- **Type checking**: Errors can be detected at compile-time rather than at runtime.

Disadvantages of Embedded SQL

- **Complexity**: Embedded SQL can make the program more complex. Developers need to know both SQL and the host language.
- **Limited portability**: SQL code embedded in one host language may not be portable to another host language. However, SQL standards try to ensure some level of portability.
- **Precompilation required**: Embedded SQL requires a precompilation step, which can make the development process more complicated compared to standalone SQL.

Questions?

What will come next?

1 Welcome to Database Systems
2 Introduction to Database Systems
3 Entity Relationship Design Diagram (ERM)
4 Relational Model
5 Relational Algebra
6 Structured Query Language (SQL)
7 Relational Database Design - Functional Dependencies
8 Relational Database Design - Normalization
9 Online Analytical Processing + Embedded SQL
10 Physical Representation - Storage and File Structure
11 Physical Representation - Indexing and Hashing
12 Transactions
13 Concurrency Control Techniques
14 Recovery Techniques
15 Query Processing and Optimization
16 Data Mining