DBMS

Week 05

SQL Data Types and Schemas

Date and Time Types

- SQL supports date, time, and timestamp data types for handling date and time information.
- Date represents a calendar date with year, month, and day.
- Time represents the time of day with hours, minutes, and seconds, and can include fractional seconds.
- Timestamp combines date and time, including fractional seconds.
- SQL provides functions to extract individual components from date and time values.
- Time-zone information can also be stored and manipulated with these data types.
- SQL offers functions to get the current date and time.

Type Conversion and Formatting Functions

- SQL allows explicit type conversion using the cast function.
- Formatting functions allow customization of data display, such as number of digits or date format.
- Handling of null values in query results can be controlled using functions like coalesce.

Default Values

- Default values can be specified for attributes in a table definition using the default keyword.
- When a tuple is inserted without a value for an attribute, the default value is used.

Large-Object Types

- SQL provides large-object data types (LOBs) for storing large data items like images or videos.
- Large objects are often stored in chunks or locators, allowing efficient retrieval.
- Different database systems may implement LOBs differently.

User-Defined Types

- SQL supports user-defined types, including distinct types and structured data types.
- Distinct types define new types based on existing ones with optional constraints.
- Structured data types can include nested structures, arrays, and subtypes.
- User-defined types can enhance type checking and data integrity.

Generating Unique Key Values

- Database systems support automatic generation of unique key values using features like identity columns or sequences.
- These mechanisms ensure the uniqueness and manageability of primary key values.

Create Table Extensions

- SQL allows creating new tables with the same schema as existing tables using create table like.
- create table ... as creates new tables populated with query results, similar to creating views.

Schemas, Catalogs, and Environments

- SQL provides a three-level hierarchy for naming relations: catalogs, schemas, and objects.
- Schemas help organize and isolate objects within a database.
- Each user has a default catalog and schema, contributing to a user's SQL environment.
- Users can access relations by providing a three-part name or using defaults.

Index Definition in SQL

Indices in a Database

- Queries often access a small portion of the records in a relation.
- An index is a data structure that allows the database system to efficiently retrieve tuples with a specified value for a certain attribute without scanning the entire relation.
- Indices are not required for correctness; they are part of the physical schema and enhance query performance.
- They are especially important for efficient processing of transactions and queries.

Creating an Index

- Indices can be created using the CREATE INDEX command.
- Syntax: CREATE INDEX <index-name> ON <relation-name> (<attribute-list>);
- The attribute-list specifies the attributes forming the search key for the index.
- Example: CREATE INDEX dept_index ON instructor (dept_name);

Using an Index

- The SQL query processor automatically uses an index when it can benefit a query.
- For example, querying for an instructor tuple with dept_name "Music" will use the dept_index to efficiently find the required tuple.

Unique Index and Candidate Key

- Adding the UNIQUE keyword to the index definition enforces uniqueness of the indexed values.
- Example: CREATE UNIQUE INDEX dept_index ON instructor (dept_name);
- If dept_name is not already a candidate key, the system will enforce this uniqueness.
- Subsequent attempts to insert tuples violating the unique constraint will fail.

Dropping an Index

- Indices can be dropped using the DROP INDEX command.
- Syntax: DROP INDEX <index-name>;
- The index name is required to drop an index from the relation.

Index Types and Clustering

- Some database systems allow specifying the type of index to be used, like B+-tree or hash indices (covered in Chapter 14).
- Certain systems allow declaring one index as clustered, meaning the relation is stored sorted by the clustered index's search key.

Implementation and Decision on Indices

- Chapter 14 covers how indices are implemented and which ones are automatically created by databases.
- Deciding on additional indices to create involves considerations like query performance and update processing impact.

Authorization

Types of Authorizations

- Authorizations include privileges that grant users specific rights on parts of the database.
- Types of authorizations (privileges) include:
  - Read data authorization.
  - Insert new data authorization.
  - Update data authorization.
  - Delete data authorization.

Privilege Granting and Checking

- When a user submits a query or update, the system checks if the user is authorized based on the granted authorizations.
- If the user doesn't have the necessary authorization, the query or update is rejected.

Authorizations on Data and Schema

- Users can be authorized to perform actions on data as well as on the database schema.
- Actions like creating, modifying, or dropping relations can be authorized.
- Users can be granted certain authorizations and may have the ability to pass on or revoke those authorizations to others.

Roles

- Roles are predefined sets of authorizations that can be granted to users.
- Roles allow users to be granted a set of privileges without specifying each privilege individually.
- Roles can be granted to users and other roles, and they form a hierarchy of privileges.
- Users who hold roles inherit all privileges granted to those roles.

Authorizations on Views

- Users can be authorized to access views that restrict their access to certain parts of the data.
- When a user queries a view, the system checks authorization based on the user's privileges.

Transfer and Revocation of Privileges

- Users with granted privileges may be allowed to pass on (grant) or withdraw (revoke) those privileges to/from other users.
- Privileges can cascade, meaning revoking a privilege from a user may also revoke it from users who received the privilege from the original user.

Row-Level Authorization

- Some database systems offer row-level authorization, where users can be authorized to access specific tuples (rows) in a relation.
- This fine-grained authorization is based on associating a function that returns predicates for restricting access to specific rows.

Limitations and Considerations

- Care must be taken when granting and revoking privileges to avoid unintended cascading effects or security risks.
- Authorization mechanisms can vary between different database systems.

Class activity

- Join types
  - Natural join
  - Inner join with using and on
  - Left, right and full outer join
  - Outer join with using and on
- View definition
  - Materialized views
  - View maintenance
  - View update
- Transactions
  - Commit work
  - Rollback work
  - Atomic transaction
- Constraints
  - Integrity constraints
  - Domain constraints
  - Unique constraint
  - Check clause
  - Referential integrity
  - Cascading deletes
  - Cascading updates
  - Assertions
- Data types
  - Date and time types
  - Default values
  - Large objects
  - clob
  - blob
  - User-defined types
  - distinct types
  - Domains
  - Type conversions
- Catalogs
- Schemas
- Indices
- Privileges
  - Types of privileges
  - select
  - insert
  - update
  - Granting of privileges
  - Revoking of privileges
  - Privilege to grant privileges
  - Grant option
- Roles
- Authorization on views
- Execute authorization
- Invoker privileges
- Row-level authorization
- Virtual private database (VPD)

Chapter 5 Advanced SQL

Advanced SQL goes beyond intermediate-level concepts and delves into more specialized and sophisticated topics for handling complex data scenarios and optimizing database performance.

Accessing SQL from a Programming Language

Dynamic SQL

- Dynamic SQL allows a general-purpose program to connect to a database server and interact with it by constructing SQL queries as character strings at runtime.
- Programs using dynamic SQL can submit these dynamically constructed SQL queries to the database server and retrieve the results one tuple (row) at a time.
- It is a flexible approach that enables programs to build SQL queries on the fly to handle complex scenarios that SQL alone might not cover.
- Examples of APIs for dynamic SQL include JDBC for Java, Python Database API for Python, and ODBC (originally for C, but extended to other languages).

Embedded SQL

- Embedded SQL also enables programs to interact with a database server but with a different approach.
- In embedded SQL, SQL statements are identified and included in the program's source code, typically as annotations.
- A preprocessor is used to translate these embedded SQL statements into function calls that interact with the database.
- The SQL statements are resolved at compile time, and the resulting program contains calls to the database API that correspond to the embedded SQL statements.
- The embedded SQL approach can help ensure that SQL statements are correctly formed and checked at compile time.
- The actual interaction with the database still occurs at runtime using dynamic SQL facilities provided by the database API.

Challenges

- Mixing SQL with general-purpose languages involves handling the fundamental difference in data manipulation. SQL operates on relations, while programming languages work with variables.
- Returning query results in a format that the program can handle is a significant challenge, as SQL queries return entire relations, and programming languages work with individual variables.

Integration in Real Applications

- Real-world applications often consist of multiple components, and while SQL is powerful for data manipulation, other parts of an application, such as user interfaces, reporting, and business logic, may require a general-purpose programming language.
- Integrating SQL and a general-purpose language is essential for creating comprehensive, functional applications that can interact with databases efficiently.

Java Database Connectivity

Connecting to the Database

- JDBC allows Java programs to open a connection to a database server using the DriverManager.getConnection() method.
- The method takes parameters such as the database URL, username, and password to establish a connection.
- The URL specifies details like the database server's address, port, and the specific database to use.

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnectionExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "myuser";
        String password = "mypassword";

        try {
            Connection connection = DriverManager.getConnection(url, username, password);
            System.out.println("Connected to the database!");
            // Perform database operations here
            connection.close();
        } catch (SQLException e) {
            System.err.println("Connection failed: " + e.getMessage());
        }
    }
}
```

Executing SQL Statements

- After establishing a connection, Java programs can use a Statement object to execute SQL statements.
- SQL statements can be executed using methods like executeQuery() for queries and executeUpdate() for non-query statements (e.g., insert, update, delete).

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class ExecuteSQLStatementExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "myuser";
        String password = "mypassword";

        try (Connection connection = DriverManager.getConnection(url, username, password);
             Statement statement = connection.createStatement()) {
            // Execute a SQL statement (e.g., insert)
            int rowsAffected = statement.executeUpdate("INSERT INTO users (name, age) VALUES ('Alice',
30)");
            System.out.println("Rows affected: " + rowsAffected);
        } catch (SQLException e) {
            System.err.println("SQL error: " + e.getMessage());
        }
    }
}
```

Exception Handling and Resource Management

- JDBC calls may throw exceptions (e.g., SQLException), so programs should include error-handling code (try-catch blocks).
- Proper resource management is crucial to close connections and statements after use to avoid resource leaks. The try-with-resources construct is recommended.

Retrieving Query Results

- When executing a query, the result is typically retrieved into a ResultSet object.
- ResultSet provides methods like next() to iterate through results and getXXX() to retrieve data from the result set.

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class RetrieveQueryResultsExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "myuser";
        String password = "mypassword";

        try (Connection connection = DriverManager.getConnection(url, username, password);
             Statement statement = connection.createStatement()) {
            ResultSet resultSet = statement.executeQuery("SELECT name, age FROM users");
            while (resultSet.next()) {
                String name = resultSet.getString("name");
                int age = resultSet.getInt("age");
                System.out.println("Name: " + name + ", Age: " + age);
            }
        } catch (SQLException e) {
            System.err.println("SQL error: " + e.getMessage());
        }
    }
}
```

Prepared Statements

- Prepared statements allow for the creation of SQL queries with placeholders (usually represented as ?) for parameterized values.
- These statements can be precompiled, improving efficiency when executing the same query with different parameter values.
- Prepared statements also protect against SQL injection by automatically escaping input values.

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class PreparedStatementExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "myuser";
        String password = "mypassword";

        try (Connection connection = DriverManager.getConnection(url, username, password)) {
            String insertQuery = "INSERT INTO users (name, age) VALUES (?, ?)";
            PreparedStatement preparedStatement = connection.prepareStatement(insertQuery);
            preparedStatement.setString(1, "Bob");
            preparedStatement.setInt(2, 25);
            int rowsAffected = preparedStatement.executeUpdate();
            System.out.println("Rows affected: " + rowsAffected);
        } catch (SQLException e) {
            System.err.println("SQL error: " + e.getMessage());
        }
    }
}
```

Callable Statements

- JDBC provides CallableStatement for invoking SQL stored procedures and functions.
- These statements are used to call functions and procedures and can handle return values and output parameters.

Metadata Features

- JDBC provides metadata interfaces like ResultSetMetaData and DatabaseMetaData to retrieve information about the database schema, tables, columns, and more.
- These interfaces help in making Java applications adaptable to different database schemas.

Other Features

- JDBC offers features like updatable result sets, which enable updates to a result set to be reflected in the corresponding database table.
- It also supports transactions with features like setAutoCommit() to control automatic transaction commits.
- JDBC handles large objects (e.g., BLOBs and CLOBs) efficiently by using streaming to read/write their data.
- Row sets allow result sets to be collected and manipulated in memory, providing more flexibility.

Security Considerations

- The text highlights the importance of using prepared statements to prevent SQL injection attacks, which can be used to manipulate or damage a database.
- It also mentions that multiple SQL statements in a single JDBC execute method should be avoided due to potential security risks.

Open Database Connectivity

ODBC Overview

- ODBC is a standard API (Application Programming Interface) that allows applications to connect to various database systems.
- It provides a common interface for interacting with databases, enabling applications like GUIs, statistics packages, and spreadsheets to work with different database servers that support ODBC.

ODBC Library

- Each database system supporting ODBC provides a library that must be linked with the client program.
- This library facilitates communication between the client program and the database server.

Setting up a Connection

- The example begins by allocating an SQL environment and a database connection handle using SQLAllocEnv and SQLAllocConnect.
- It then establishes a connection to the database using SQLConnect, providing server information, username, and password.

Executing SQL Statements

- SQL statements are executed using SQLExecDirect.
- The example demonstrates running a SQL query to retrieve data from the database.

Binding Variables

- SQLBindCol is used to bind C language variables to the attributes of the query result.
- This allows fetched result values to be stored in C variables.

Fetching Results

- A while loop with SQLFetch is used to iterate through the result set and retrieve rows.
- Retrieved data is printed to the console.

Resource Management

- At the end of the session, the program frees the statement handle, disconnects from the database, and frees up connection and SQL environment handles.

```c
void ODBCexample()
{
    RETCODE error;
    HENV env; /* environment */
    HDBC conn; /* database connection */
    SQLAllocEnv(&env);
    SQLAllocConnect(env, &conn);
    SQLConnect(conn, "db.yale.edu", SQL NTS, "avi", SQL NTS, "avipasswd", SQL NTS);
    {
        char deptname[80];
        float salary;
        int lenOut1, lenOut2;
        HSTMT stmt;
        char * sqlquery = "select dept name, sum (salary) from instructor group by dept name";
        SQLAllocStmt(conn, &stmt);
        error = SQLExecDirect(stmt, sqlquery, SQL NTS);
        if (error == SQL SUCCESS) {
            SQLBindCol(stmt, 1, SQL C CHAR, deptname , 80, &lenOut1);
            SQLBindCol(stmt, 2, SQL C FLOAT, &salary, 0 , &lenOut2);
            while (SQLFetch(stmt) == SQL SUCCESS) {
                printf (" %s %g\n", deptname, salary);
            }
        }
        SQLFreeStmt(stmt, SQL DROP);
    }
    SQLDisconnect(conn);
    SQLFreeConnect(conn);
    SQLFreeEnv(env);
}
```

Functions and Procedures

Functions and Procedures

- Functions and procedures are used to encapsulate business logic or operations that can be invoked from SQL statements.
- They are particularly useful for handling specialized data types and complex operations.

Advantages of Storing in the Database

- Storing functions and procedures in the database offers several advantages.
- It allows for multiple applications to access and use the same procedures, provides a central location for business rules, and ensures consistency in rule enforcement.

SQL Standard and Implementation Variations

- While the SQL standard defines the syntax for functions and procedures, most database systems have their variations and non-standard implementations.
- For example, Oracle, Microsoft SQL Server, and PostgreSQL have their own syntax and features for procedures and functions.

Declaring and Invoking Functions and Procedures

- Functions and procedures can be declared in SQL using the CREATE FUNCTION and CREATE PROCEDURE statements.
- They can have parameters (input and output) and return values.
- Functions can return values, while procedures may not return values directly but can use output parameters.

```sql
-- Create a stored procedure
CREATE PROCEDURE UpdateProductPrices
    @CategoryID INT
AS
BEGIN
    -- Update product prices in the specified category
    UPDATE Products
    SET Price = Price * 1.1
    WHERE CategoryID = @CategoryID;
END;

GO

-- Execute the stored procedure
EXEC UpdateProductPrices @CategoryID = 1;
```

```sql
-- Create a function
CREATE FUNCTION CalculateTotalPrice
    (@Price DECIMAL(10, 2), @Quantity INT)
RETURNS DECIMAL(10, 2)
AS
BEGIN
    DECLARE @TotalPrice DECIMAL(10, 2);

    -- Calculate the total price
    SET @TotalPrice = @Price * @Quantity;

    RETURN @TotalPrice;
END;
GO
-- Use the function in a query
SELECT ProductName, Price, Quantity, dbo.CalculateTotalPrice(Price, Quantity) AS TotalPrice
FROM ShoppingCart;
```

SQL Procedural Constructs

- SQL supports various procedural constructs like variable declarations, assignments, compound statements, loops (such as WHILE and REPEAT), conditional statements (IF-THEN-ELSE), and exception handling using SIGNAL and DECLARE HANDLER.

External Language Routines

- In addition to SQL-defined functions and procedures, external language routines can be defined in languages like Java, C#, C, or C++.
- These routines offer greater flexibility and can be executed from SQL queries.
- The specific syntax for defining external language routines varies by database system.

Security Considerations

- Executing code outside the database system may carry security risks, as it can potentially corrupt database structures or bypass access controls.
- Some database systems use sandboxes to execute code securely within the database query execution process.