Advanced SQL Programming

EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

Week 4, Topic 3, Lesson 7

- Embedded SQL
- Dynamic SQL

Agenda (what we will build)

- Embedded SQL: host variables, cursors, transactions
- Dynamic SQL: safe patterns (parameters, quoting, whitelists)
- Embed SQL into app code using host variables and cursors
- Generate dynamic SQL safely (avoid SQL injection)

Sample schema used in examples

You can adapt these patterns to your own tables.

```
-- Minimal schema (compact)
CREATE TABLE departments(dept_id int primary key, dept_name text not null);

CREATE TABLE employees(
  emp_id int primary key,
  dept_id int references departments(dept_id),
  full_name text not null,
  email text unique,
  salary numeric(12,2) not null default 0,
  updated_at timestamptz not null default now()
);

CREATE TABLE products(product_id int primary key, name text not null, price numeric(12,2) not null);
CREATE TABLE inventory(product_id int primary key references products(product_id), qty_on_hand int not null);

CREATE TABLE orders(order_id bigint generated always as identity primary key,
  customer_id int not null, order_date date not null default current_date);

CREATE TABLE order_items(order_id bigint references orders(order_id),
  product_id int references products(product_id), quantity int not null,
  unit_price numeric(12,2) not null, primary key(order_id, product_id));
```

Embedded SQL: what it is

- You write SQL in your program; a precompiler/driver sends it to the DB
- Host variables carry values between your code and SQL
- Best practice: use parameter markers (don’t string-concatenate values)
- Common patterns: SELECT INTO, INSERT/UPDATE, cursors, transactions
- Embedded SQL means writing SQL statements inside a host programming language such as C, C++, Java, COBOL, or Python.

Why use embedded SQL?

You keep SQL close to the app logic while still letting the database do set-based work.

```
-- Key idea: values come from host variables (not string concat)
SELECT full_name, salary
FROM employees
WHERE dept_id = ? AND salary > ?
ORDER BY salary DESC;
```

Embedded SQL...

Structure -

EXEC SQL
  SQL statement
END-EXEC;

- Variables from the program are shared with SQL using host variables (preceded by :).

Embedded SQL Example using Java – JDBC style

```
Connection con = DriverManager.getConnection(
   "jdbc:mysql://localhost:3306/company", "user", "pass");

PreparedStatement ps =
   con.prepareStatement("SELECT name FROM employee WHERE id=?");

ps.setInt(1, 101);

ResultSet rs = ps.executeQuery();

while (rs.next()) {
   System.out.println(rs.getString("name"));
}
```

Embedded SQL: host variables

"Host variables" are placeholders bound from your programming language.

Pattern

1) Declare host variables 2) Bind values 3) Run SQL 4) Read output

Embedded SELECT … INTO (pseudo-code):

```
/* PSEUDO-C (idea is similar in many languages) */
int emp_id = 101;
char name[80];
double salary;

EXEC SQL SELECT full_name, salary
         INTO :name, :salary
         FROM employees
         WHERE emp_id = :emp_id;
```

Embedded SQL: cursor loop

Use a cursor when you must process rows one-by-one in the host language.

- Declare cursor for a SELECT query
- OPEN → FETCH in a loop → CLOSE
- Prefer set-based SQL when possible; cursors can be slower

```
/* PSEUDO-C cursor pattern */
int dept_id = 10;
int emp_id;
char full_name[80];

EXEC SQL DECLARE emp_cur CURSOR FOR
  SELECT emp_id, full_name
  FROM employees
  WHERE dept_id = :dept_id
  ORDER BY emp_id;

EXEC SQL OPEN emp_cur;
```

```
while (true) {
  EXEC SQL FETCH emp_cur INTO :emp_id,
:full_name;
  if (SQLCODE != 0) break;   -- no more rows /
error
  /* process row */
}
EXEC SQL CLOSE emp_cur;
```