

In CodeIgniter 3, you can query the database using two primary methods: the Active Record (or Query Builder) pattern and Raw SQL queries. Below are explanations and examples for both methods.

1. Active Record (Query Builder) in CodeIgniter

The Active Record (Query Builder) pattern in CodeIgniter provides a more secure, readable, and maintainable way to work with the database. It automatically escapes your data, preventing SQL injection attacks and making your code cleaner.

Example 1: SELECT Query

php

Copy code

```
// Load the database library (if not already loaded)
$this->load->database();

// Get all employees from the employee table
$query = $this->db->get('employee');
$result = $query->result_array(); // Fetch result as an array

print_r($result); // Output the result
```

- Explanation: The `get()` method fetches all records from the employee table.

Example 2: SELECT with WHERE Condition

php

Copy code

```
// Load the database library
$this->load->database();

// Get a specific employee by ID
$this->db->where('id', 10);
$query = $this->db->get('employee');
$result = $query->row_array(); // Fetch single row as an array

print_r($result);
```

- Explanation: The `where()` method adds a condition, and `row_array()` fetches a single row.

Example 3: INSERT Query

php

Copy code

```
// Load the database library
$this->load->database();

// Insert a new employee
$data = array(
    'first_name' => 'Alice',
    'last_name' => 'Wonderland',
    'email' => 'alice@example.com',
    'phone' => '555-6789',
    'department' => 'Marketing',
    'salary' => 60000.00,
    'hire_date' => '2021-03-15'
);

$this->db->insert('employee', $data);

echo "Inserted ID: " . $this->db->insert_id(); // Get the last
inserted ID
```

- Explanation: The `insert()` method inserts data into the table. The `insert_id()` method retrieves the ID of the inserted row.

Example 4: UPDATE Query

php

Copy code

```
// Load the database library
$this->load->database();

// Update an employee's salary
$data = array(
    'salary' => 65000.00
);

$this->db->where('id', 10);
$this->db->update('employee', $data);

echo "Affected Rows: " . $this->db->affected_rows();
```

- Explanation: The `update()` method updates the records in the table. The `affected_rows()` method returns the number of rows affected by the update.

Example 5: DELETE Query

php

Copy code

```
// Load the database library
$this->load->database();

// Delete an employee by ID
$this->db->where('id', 10);
$this->db->delete('employee');

echo "Affected Rows: " . $this->db->affected_rows();
```

- Explanation: The `delete()` method deletes records from the table. The `affected_rows()` method returns the number of rows affected by the deletion.

Example 6: Complex Query with JOIN and ORDER BY

php

Copy code

```
// Load the database library
$this->load->database();

// Get employees and their department details with JOIN and ORDER BY
$this->db->select('employee.*, department.name as department_name');
$this->db->from('employee');
$this->db->join('department', 'department.id = employee.department_id');
$this->db->order_by('employee.hire_date', 'DESC');
$query = $this->db->get();

$result = $query->result_array();
print_r($result);
```

- Explanation: The `join()` method allows you to perform a SQL JOIN, and `order_by()` adds an ORDER BY clause.

2. Raw SQL Queries in CodeIgniter

Sometimes you might need to write raw SQL queries for complex queries or when you prefer using SQL directly.

Example 1: SELECT Query

php

Copy code

```
// Load the database library
```

```
$this->load->database();
```

```
// Run a raw SQL SELECT query
```

```
$query = $this->db->query("SELECT * FROM employee");
```

```
$result = $query->result_array(); // Fetch result as an array
```

```
print_r($result);
```

- Explanation: The query() method executes a raw SQL query, and result_array() fetches the result as an array.

Example 2: SELECT with WHERE Condition

php

Copy code

```
// Load the database library
```

```
$this->load->database();
```

```
// Run a raw SQL SELECT query with WHERE condition
```

```
$query = $this->db->query("SELECT * FROM employee WHERE id = ?",  
array(10));
```

```
$result = $query->row_array(); // Fetch single row as an array
```

```
print_r($result);
```

- Explanation: The ? placeholder is used to safely bind parameters, preventing SQL injection.

Example 3: INSERT Query

php

Copy code

```
// Load the database library
```

```
$this->load->database();
```

```
// Run a raw SQL INSERT query
```

```
$this->db->query("INSERT INTO employee (first_name, last_name,  
email, phone, department, salary, hire_date)
```

```
VALUES ('Alice', 'Wonderland',
```

```
'alice@example.com', '555-6789', 'Marketing', 60000.00,
```

```
'2021-03-15')");
```

```
echo "Inserted ID: " . $this->db->insert_id();
```

- **Explanation:** This example directly runs a SQL INSERT query. You can still use `insert_id()` to get the last inserted ID.

Example 4: UPDATE Query

php

Copy code

```
// Load the database library
```

```
$this->load->database();
```

```
// Run a raw SQL UPDATE query
```

```
$this->db->query("UPDATE employee SET salary = 65000.00 WHERE id = 10");
```

```
echo "Affected Rows: " . $this->db->affected_rows();
```

- **Explanation:** This example updates the salary of an employee using a raw SQL query.

Example 5: DELETE Query

php

Copy code

```
// Load the database library
```

```
$this->load->database();
```

```
// Run a raw SQL DELETE query
```

```
$this->db->query("DELETE FROM employee WHERE id = 10");
```

```
echo "Affected Rows: " . $this->db->affected_rows();
```

- **Explanation:** This example deletes a record using a raw SQL query.

3. When to Use Active Record vs. Raw SQL

- **Active Record/Query Builder:**
 - Use when you need a more secure, readable, and maintainable approach.
 - Automatically handles escaping and prevents SQL injection.
 - Preferred for simple and common queries (SELECT, INSERT, UPDATE, DELETE).
- **Raw SQL:**

- Use when you need full control over the SQL query.
- Suitable for complex queries that may not be easily handled by Active Record.
- Can be faster in some cases where you want to bypass Active Record's abstraction.

Conclusion

CodeIgniter's Query Builder (Active Record) provides a powerful and secure way to interact with the database, while raw SQL queries offer flexibility when you need to write complex SQL statements. Depending on your application's needs, you can use either or a combination of both.

a. Would you like to explore more advanced query examples, such as subqueries or stored procedures?

b. Need help optimizing or debugging specific queries in your application?

You said:

in case of using dynamic data in raw query how to use it and what about to prevent the sqlinjection

ChatGPT said:



When using dynamic data in raw SQL queries in CodeIgniter, it's crucial to prevent SQL injection by properly escaping the input. CodeIgniter provides a few methods to help you safely include dynamic data in raw SQL queries, primarily by using query bindings.

1. Using Query Bindings

Query bindings are the safest and most recommended way to include dynamic data in raw SQL queries in CodeIgniter. With bindings, CodeIgniter automatically escapes the input, preventing SQL injection.

Example: SELECT Query with Bindings

php

Copy code

```
$id = 10;
$query = $this->db->query("SELECT * FROM employee WHERE id = ?",
    array($id));
$result = $query->row_array();
```

```
print_r($result);
```

Explanation:

- **? Placeholder:** This is a placeholder for the dynamic value in the query.
- **array(\$id):** This array contains the actual value to be bound to the placeholder. CodeIgniter automatically escapes it.

You can bind multiple variables by adding more placeholders and values in the array:

php

Copy code

```
$first_name = 'Patricia';  
$last_name = 'Thomas';  
$query = $this->db->query("SELECT * FROM employee WHERE first_name =  
? AND last_name = ?", array($first_name, $last_name));  
$result = $query->result_array();
```

```
print_r($result);
```

2. Using Named Bindings

CodeIgniter also supports named bindings, which can make your queries more readable.

Example: SELECT Query with Named Bindings

php

Copy code

```
$id = 10;  
$query = $this->db->query("SELECT * FROM employee WHERE id = :id",  
array(':id' => $id));  
$result = $query->row_array();
```

```
print_r($result);
```

Explanation:

- **:id Placeholder:** This is a named placeholder for the dynamic value.
- **array(':id' => \$id):** This array associates the placeholder with the actual value.

3. Escaping Data Manually (If Necessary)

While query bindings are the safest method, if you need to include dynamic data directly in your query string (not recommended), you should manually escape the data using CodeIgniter's `escape()` method.

Example: SELECT Query with Manual Escaping

php

Copy code

```
$id = 10;
$escaped_id = $this->db->escape($id);

$query = $this->db->query("SELECT * FROM employee WHERE id =
$escaped_id");
$result = $query->row_array();

print_r($result);
```

Explanation:

- `$this->db->escape($id)`: This method escapes the data, making it safe to include directly in the query string.

4. Example with INSERT Query and Bindings

Using query bindings with an INSERT query is straightforward and safe.

Example: INSERT Query with Bindings

php

Copy code

```
$data = array(
    'first_name' => 'Alice',
    'last_name' => 'Wonderland',
    'email' => 'alice@example.com',
    'phone' => '555-6789',
    'department' => 'Marketing',
    'salary' => 60000.00,
    'hire_date' => '2021-03-15'
);

$this->db->query("INSERT INTO employee (first_name, last_name,
email, phone, department, salary, hire_date)
VALUES (?, ?, ?, ?, ?, ?, ?)",
array_values($data));
```



```
echo "Inserted ID: " . $this->db->insert_id();
```

Explanation:

- **? Placeholders:** Placeholders are used for each column value.
- **array_values(\$data):** This extracts the values from the \$data array to bind to the placeholders.

5. Preventing SQL Injection Summary

- **Use Query Bindings (? or named bindings):** This is the safest way to insert dynamic data in raw SQL queries. CodeIgniter automatically escapes the input.
- **Avoid Concatenating User Input Directly:** Never concatenate user input directly into SQL queries without escaping, as it can lead to SQL injection vulnerabilities.
- **Manual Escaping:** If necessary, use `$this->db->escape()` to escape user input manually before including it in your SQL queries.

Example: Full Code with Safe Raw Query

php

Copy code

```
$id = 10;
```

```
$first_name = 'Patricia';
```

```
$last_name = 'Thomas';
```

```
// SELECT with bindings
```

```
$query = $this->db->query("SELECT * FROM employee WHERE id = ? AND  
first_name = ? AND last_name = ?", array($id, $first_name,  
$last_name));
```

```
$result = $query->row_array();
```

```
print_r($result);
```

```
// INSERT with bindings
```

```
$data = array(  
    'first_name' => 'Alice',  
    'last_name' => 'Wonderland',  
    'email' => 'alice@example.com',  
    'phone' => '555-6789',  
    'department' => 'Marketing',  
    'salary' => 60000.00,  
    'hire_date' => '2021-03-15'  
);
```

```
$this->db->query("INSERT INTO employee (first_name, last_name,  
email, phone, department, salary, hire_date)  
VALUES (?, ?, ?, ?, ?, ?, ?)",  
array_values($data));  
  
echo "Inserted ID: " . $this->db->insert_id();
```

Conclusion

Using query bindings is the best practice for including dynamic data in raw SQL queries in CodeIgniter. It ensures that your queries are safe from SQL injection attacks and keeps your code clean and readable.