request.query and request.param and request.body

In Express.js, you can handle different types of data from incoming HTTP requests, including request query parameters, request route parameters (also known as request params), and request bodies. Here are examples of each:

1. **Request Query Parameters**: Query parameters are typically included in the URL and are used for filtering or providing additional information in a GET request. For example, if you have a route for searching books by title, you might have a query parameter for specifying the title.

```
const express = require('express');
const app = express();

app.get('/books', (req, res) => {
  const title = req.query.title;
  // Use the 'title' query parameter to search for books.
  // Example URL: /books?title=Node.js
});
```

2. **Request Route Parameters**: Route parameters are used to capture values from the URL itself. They are specified in the route definition with a colon `:`.

```
const express = require('express');
const app = express();

app.get('/books/:id', (req, res) => {
  const bookId = req.params.id;
  // Use the 'bookId' parameter to fetch a specific book.
  // Example URL: /books/123
});
```

3. **Request Body**: The request body typically contains data sent in the body of a POST, PUT, or PATCH request. It's often in JSON format, but it can also be in other formats such as form data or XML. To handle the request body, you'll need middleware like `express.json()` or `express.urlencoded()` to parse it.

```
const express = require('express');
const app = express();

app.use(express.json()); // Parse JSON in the request body

app.post('/books', (req, res) => {
  const bookData = req.body;
  // Use the 'bookData' to create a new book entry.
});
```

Remember to install the `express` library if you haven't already using `npm install express` and use `app.listen()` to start your Express.js server. These are basic examples, and you may need to add error handling, validation, and additional logic based on your specific use case.