SCHEMA VALIDATION

www.geminiqa.com

What is Schema Validation?

Schema validation is the process of ensuring that the structure, format, and data types of input (such as JSON, XML, or database entries) conform to a predefined schema. A schema serves as a blueprint for what data should look like, including rules for required fields, data types, value constraints, and more.

Why Use Schema Validation?

1. **Data Integrity:** Ensures that incoming or outgoing data meets the expected structure and format.
2. **Error Prevention:** Catches invalid data early, reducing bugs and unexpected errors in applications.
3. **Security:** Prevents malicious inputs that could exploit weaknesses in the application.
4. **Interoperability:** Facilitates communication between systems by ensuring data is in the expected format.
5. **Automation:** Allows automated validation of data in APIs, databases, or configurations.

Types of Schema Validation

1. **JSON Schema Validation**
   - Used for validating JSON data structures.
   - Popular tools: `ajv` (for Node.js), `jsonschema`.
2. **XML Schema Validation**
   - Used for validating XML documents.
   - Standards: DTD (Document Type Definition), XSD (XML Schema Definition).
3. **Database Schema Validation**
   - Ensures that database records conform to table structure and constraints.
   - Uses rules like foreign keys, primary keys, and field constraints.
4. **HTML Form Validation**
   - Validates user inputs in forms to ensure they meet criteria like length, format, and data type.
5. **Custom Schema Validation**
   - Developers create custom validation logic to meet unique application requirements.

1. JSON Schema Validation

Schema (JSON):

json
Copy code

```
{
  "type": "object",
  "properties": {
    "id": { "type": "integer" },
    "name": { "type": "string" },
    "email": { "type": "string", "format": "email" }
  },
  "required": ["id", "name"]
}
```

Validation Code Of JSON

Validation Code (JavaScript with AJV):

javascript
Copy code

```
const Ajv = require("ajv");
const ajv = new Ajv();

const schema = {
  type: "object",
  properties: {
    id: { type: "integer" },
    name: { type: "string" },
    email: { type: "string", format: "email" }
  },
  required: ["id", "name"]
};

const data = { id: 1, name: "John", email:
"john@example.com" };
const validate = ajv.compile(schema);
console.log(validate(data)); // true
```

2. XML Schema Validation (XSD)

Schema (XSD):

xml
Copy code

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="id" type="xs:integer"/>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="email" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Validation Code (Java):

java
Copy code

```
SchemaFactory schemaFactory =
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS
_URI);
Schema schema = schemaFactory.newSchema(new
File("schema.xsd"));
Validator validator = schema.newValidator();
validator.validate(new StreamSource(new
File("data.xml")));
```

3. Database Schema Validation

MySQL Schema:

sql
Copy code

```
CREATE TABLE Users (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL
);
```

Violation Example: Trying to insert duplicate email:

```
CREATE TABLE Users (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL
);
```

4. HTML Form Validation

HTML Form:

html
Copy code

```
<form>
  <input type="text" name="username" required
minlength="3">
  <input type="email" name="email" required>
  <button type="submit">Submit</button>
</form>
```

5. Python Code (Custom Rules):

python
Copy code

```
from jsonschema import validate, ValidationError

schema = {
    "type": "object",
    "properties": {
        "id": {"type": "integer"},
        "name": {"type": "string", "minLength": 3}
    },
    "required": ["id", "name"]
}

data = {"id": 1, "name": "Jo"}

try:
    validate(instance=data, schema=schema)
    print("Valid data!")
except ValidationError as e:
    print(f"Validation error: {e}")
```