# Efficient Data Exchange: Leveraging SQL JSON Results with C#

In my custom API development, I have found that utilizing SQL JSON results for read-only operations - especially within the pharmacy operations and patient prescription management domain - offers significant flexibility in data structuring and facilitates seamless data exchange with front-end applications and third-party systems, a crucial factor in healthcare applications where data formats and structures often vary. Some of the advantages of this methodology are:

**Simplified Data Retrieval**: Using SQL's JSON functionality allows you to return complex hierarchical data structures directly from the database. This can be particularly useful for retrieving patient information, medication lists, and prescription details in a single query, reducing the need for multiple round trips to the database.

**Reduced Payload Size**: When you retrieve only the necessary JSON fields rather than entire database records, you can minimize the amount of data sent over the network. This is beneficial for mobile applications or web interfaces where bandwidth may be limited, improving overall performance.

**Easy Integration**: JSON is a widely adopted data format, especially in web applications. Returning results in JSON format allows for seamless integration with front-end frameworks and other systems, making it easier to present data in a user-friendly manner.

**Enhanced Readability**: JSON format is human-readable, making debugging and logging easier. When working with API responses, developers can quickly understand the structure of the data returned, aiding in development and maintenance.

**Aggregation and Filtering**: SQL's JSON functions enable you to perform aggregations and filters directly within the database. For example, you can aggregate prescription data by patient, medication, or provider, returning a summary JSON object that can be easily consumed by the API.

For a pharmacy operations API, consider a scenario where a healthcare provider requests a patient's current list of prescriptions. A single SQL query could return a JSON object containing the patient's details, a list of medications, dosage instructions, and refill information. A sample query of what the SQL query could look like.

Although this query provides a clear illustration, I recommend encapsulating it within a stored procedure, such as "GetPatientMeds", to enhance organization, promote reusability, and ensure scalability.

```
use [your_db]
go

declare @patient_id int
declare @jsonoutput nvarchar(max)


begin try
  set @jsonoutput =
  (
    select
    distinct [person].[patient_id],
    [person].[fname],
    [person].[lname],
    (
      select
      [code].[name] as [status],
      [pat_meds].[rxid],
      [pat_meds].[patient_id],
      [pat_meds].[scriptno],
      isnull(rtrim([dispense].[disp_drug_name]), rtrim([pat_meds].[drug_name])) as
[drugname],
      format([pat_meds].[expire_date], 'MM-dd-yyyy') as [expire_date],
      isnull(rtrim([pat_meds].[sig_text]), '') as [sig_text],
      isnull([pat_meds].[cmt], '') as [comments],
      [pat_meds].[ndc],
      rtrim([md].[fname]) + ' ' + rtrim([md].[mname]) + ' ' + rtrim([md].[lname]) + ' ' +
rtrim([md].[suffix_lu]) as [doctor],
      isnull([md].[cmt], '') as [dr_comments],
      isnull(rtrim([md].[deano]), '') as [deano],
      [md].[npiid]
      from [dbo].[rxlisting] as [pat_meds]
      left outer join [dbo].[rxdispense] as [dispense] on [pat_meds].[last_rxdisp_id] =
[dispense].[rxdisp_id] -- get disp_drug_name
      join [dbo].[patinfo] as [patient] on [pat_meds].[patient_id] =
[patient].[patient_id] -- get patent name
      join [dbo].[mdinfo] as [md] on [pat_meds].[md_id] = [md].[md_id] -- get dr name
      for json path
    ) as [patient_meds]

    from [dbo].[patinfo] as [person] -- primary table
    where [person].[patient_id] = @patient_id
    for json auto, include_null_values, root -- root because we want to see the root
  )
  end try
begin catch
  set @jsonoutput = 'ERROR:'
end catch
```

**Here is a sample of the raw SQL JSON Output**

{"person":[{"patient_id":1234,"fname":"Mary","lname":"Smith","patient_meds":[{"status
":"Deleted","rx_id":45321,"patient_id":1234,"drugname":"ZOVIRAX 400 MG
TABLET","expiredate":"06-15-
2024","sigtext":"","comments":"","ndc":"41176123456","doctor":"Neil A Dondero
","dr_comments":"","deano":"AB2682082","npiid":"1726543217"}]}]}

And formatted as JSON

```
{
  "person": [
    {
      "patient_id": 1234,
      "fname": "Mary",
      "lname": "Smith",
      "patient_meds": [
        {
          "status": "Deleted",
          "rxid": 45321,
          "patient_id": 1234,
          "drugname": "ZOVIRAX 400 MG TABLET",
          "expiredate": "06-15-2024",
          "sigtext": "",
          "comments": "",
          "ndc": "41176123456",
          "doctor": "Neil A Dondero",
          "dr_comments": "",
          "deano": "AB2682082",
          "npiid": "1726543217"
        }
      ]
    }
  ]
}
```

**Reading and Working with the JSON result set**

To read the provided JSON data in C# and display its properties in the debug console, we need to create classes that align with the JSON structure, deserialize the JSON string into these classes, and then output the properties in the debug console. Ultimately, this functionality should be implemented in an API endpoint, such as "GetPatientMeds".

**Define Classes**: Create classes that represent the structure of the JSON data.

**Deserialize the JSON**: Use JsonSerializer to convert the JSON string into the class objects.

**Display the Data**: Print the relevant information to the debug console.

## Example Implementation

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text.Json;

public class Person
{
  public int PatientId { get; set; }
  public string Fname { get; set; }
  public string Lname { get; set; }
  public List<PatientMed> PatientMeds { get; set; }
}

public class PatientMed
{
  public string Status { get; set; }
  public int RxId { get; set; }
  public int PatientId { get; set; }
  public string Drugname { get; set; }
  public string Expiredate { get; set; }
  public string Sigtext { get; set; }
  public string Comments { get; set; }
  public string Ndc { get; set; }
  public string Doctor { get; set; }
  public string DrComments { get; set; }
  public string Deano { get; set; }
  public string Npiid { get; set; }
}

public class Root
{
  public List<Person> Person { get; set; }
}

class Program
{
  static void Main(string[] args)
  {
    string json =
"{\"person\":[{\"patient_id\":1234,\"fname\":\"Mary\",\"lname\":\"Smith\",\"patient_
meds\":[{\"status\":\"Deleted\",\"rx_id\":45321,\"patient_id\":1234,\"drugname\":\"Z
OVIRAX 400 MG TABLET\",\"expiredate\":\"06-15-
2024\",\"sigtext\":\"\",\"comments\":\"\",\"ndc\":\"41176123456\",\"doctor\":\"Neil
A Dondero
\",\"dr_comments\":\"\",\"deano\":\"AB2682082\",\"npiid\":\"1726543217\"}]}]}";

    // serialize the JSON data
    var rootObject = JsonSerializer.Deserialize<Root>(json);

    // display the data in the debug console
    foreach (var person in rootObject.Person)
    {
      Debug.WriteLine($"Patient ID: {person.PatientId}");
      Debug.WriteLine($"First Name: {person.Fname}");
      Debug.WriteLine($"Last Name: {person.Lname}");

      foreach (var med in person.PatientMeds)
      {
        Debug.WriteLine($"  Medication: {med.Drugname}");
        Debug.WriteLine($"  Status: {med.Status}");
        Debug.WriteLine($"  Expiration Date: {med.Expiredate}");
        Debug.WriteLine($"  Prescribing Doctor: {med.Doctor}");
      }
    }
  }
}
```

**String json**: The manual population of this variable is intended solely for demonstration purposes. For production use, you should retrieve the JSON result by querying the "GetPatientMeds" stored procedure through SQLDataClient or by utilizing Entity Framework.

**Classes**: The Person, PatientMed, and Root classes correspond to the JSON structure.

**Deserialization**: The JsonSerializer.Deserialize<Root>(json) method converts the JSON string into an instance of the Root class.

**Debug Output**: The Debug.WriteLine method is used to print the details to the Visual Studio debug output window when running debug mode.

## Conclusion

Utilizing SQL JSON results significantly improves the flexibility and efficiency of data management, particularly in complex fields like pharmacy operations and patient prescription management. This method not only facilitates seamless integration with various applications but also addresses the diverse data requirements of healthcare, ultimately enhancing overall performance and patient outcomes.