1. SQL Dependency (SqlDependency)

This is a common way to listen for changes in a SQL Server database. When data changes (e.g., insert, update, delete), a notification is triggered.

Example:

```
using System;
using System.Data.SqlClient;

class Program
{
	static void Main(string[] args)
	{
	string connectionString = "your_connection_string";
	string query = "SELECT ColumnName FROM TableName";

	using (SqlConnection connection = new SqlConnection(connectionString))
	{
	connection.Open();

	SqlCommand command = new SqlCommand(query, connection);
	SqlDependency dependency = new SqlDependency(command);

	// Subscribe to the OnChange event
	dependency.OnChange += new OnChangeEventHandler(OnDatabaseChange);

	// Execute the query
	command.ExecuteReader();
	}

	Console.WriteLine("Listening for database changes...");
	Console.ReadLine();
	}

	static void OnDatabaseChange(object sender, SqlNotificationEventArgs e)
	{
	Console.WriteLine("Database change detected!");
	// Logic for handling changes
	}
}
```

Note: This requires configuring your SQL Server to support query notifications.

2. Entity Framework Core with Triggers

You can use **triggers** in the database to log changes into a table and then use **polling** in your application to fetch the changes.

---

3. Change Tracking APIs

SQL Server has features like **Change Tracking** or **Change Data Capture (CDC)**, which allow applications to track changes to a database.

You can then poll for changes periodically and process them.

---

4. SignalR for Real-Time Notifications

You can integrate **SignalR** to broadcast database changes in real time. This is particularly useful for web applications.

Steps:

1. Use a mechanism like **SqlDependency** or **Change Tracking** to detect changes.
2. Push changes to clients via **SignalR**.

---

5. FileSystemWatcher for File-Based Databases

If the data source is file-based (e.g., SQLite), you can use `.NET FileSystemWatcher` to monitor changes to the database file.