Let's start with Dynamic SQL:

Dynamic SQL:

Dynamic SQL refers to the creation and execution of SQL statements during runtime. This is often used when the structure of the query is not known until the program is running. Here are two examples, one in Python using the `sqlite3` module, and another in SQL Server using T-SQL.

Example 1: Python with sqlite3

```python
import sqlite3

def dynamic_query(table_name):
    # Connect to the SQLite database
    connection = sqlite3.connect("example.db")
    cursor = connection.cursor()

    # Build and execute dynamic query
    query = f"SELECT * FROM {table_name};"
    cursor.execute(query)

    # Fetch and print results
    results = cursor.fetchall()
    for row in results:
        print(row)

    # Close the connection
    connection.close()

# Example usage
dynamic_query("my_table")
```

In this example, the `dynamic_query` function takes a table name as an argument and dynamically builds and executes a SELECT query on that table using the `sqlite3` module in Python.

Example 2: SQL Server with T-SQL

```sql
DECLARE @tableName NVARCHAR(50)
SET @tableName = 'Employee'
```

```
DECLARE @sqlQuery NVARCHAR(MAX)
SET @sqlQuery = 'SELECT * FROM ' + @tableName

-- Execute dynamic query
EXEC sp_executesql @sqlQuery
```

In this T-SQL example, we use variables to hold the table name and the dynamic SQL query. The `sp_executesql` stored procedure is then used to execute the dynamically built query.

Now, let's move on to Backup and Restore operations.

Backup and Restore:

Backing up and restoring databases is a crucial aspect of database management. Here are two examples, one in T-SQL for SQL Server and another in Python using the `sqlite3` module.

Example 1: SQL Server Backup

```sql
-- Backup database
BACKUP DATABASE YourDatabaseName
TO DISK = 'C:\Backup\YourDatabaseName.bak'
WITH FORMAT, MEDIANAME = 'BackupMedia', NAME = 'Full Backup';

-- Optional: Verify the backup
RESTORE VERIFYONLY
FROM DISK = 'C:\Backup\YourDatabaseName.bak';
```

In this SQL Server example, the `BACKUP DATABASE` statement is used to create a backup of the database. The `RESTORE VERIFYONLY` statement is optional and is used to verify the integrity of the backup.

Example 2: Python with sqlite3 (Backup and Restore)

```python
import sqlite3
import shutil

def backup_and_restore_db():
    # Connect to the SQLite database
    source_connection = sqlite3.connect("source.db")
```

```
    destination_connection = sqlite3.connect("backup.db")

    # Backup database by copying the file
    shutil.copyfile("source.db", "backup.db")

    # Close the connections
    source_connection.close()
    destination_connection.close()

# Example usage
backup_and_restore_db()
```

In this Python example, the `shutil.copyfile` function is used to create a backup of an SQLite database by copying its file. This is a simple method for SQLite, but for larger databases or other database management systems, a more sophisticated backup strategy might be needed.

Certainly! Let's expand on the Dynamic SQL and Backup/Restore examples.

Dynamic SQL:

Example 3: Python with SQLAlchemy

```python
from sqlalchemy import create_engine, text

def dynamic_query_with_sqlalchemy(table_name):
    # Connect to the SQLite database using SQLAlchemy
    engine = create_engine('sqlite:///example.db')
    connection = engine.connect()

    # Build and execute dynamic query using SQLAlchemy's text() function
    query = text(f"SELECT * FROM {table_name}")
    result = connection.execute(query)

    # Fetch and print results
    for row in result:
        print(row)

    # Close the connection
    connection.close()

# Example usage
dynamic_query_with_sqlalchemy("my_table")
```

```

In this example, we use SQLAlchemy, a SQL toolkit for Python. The `text()` function is used to create a SQL expression, and the query is executed using SQLAlchemy's `execute` method.

**Example 4: Dynamic Update with Python and MySQL**

```python
import mysql.connector

def dynamic_update(table_name, column_name, new_value, condition_column, condition_value):
    # Connect to MySQL database
    connection = mysql.connector.connect(
        host="localhost",
        user="username",
        password="password",
        database="mydatabase"
    )
    cursor = connection.cursor()

    # Build and execute dynamic update query
    query = f"UPDATE {table_name} SET {column_name} = %s WHERE {condition_column} = %s"
    values = (new_value, condition_value)
    cursor.execute(query, values)

    # Commit the changes
    connection.commit()

    # Close the connection
    cursor.close()
    connection.close()

# Example usage
dynamic_update("employees", "salary", 50000, "employee_id", 101)
```

In this example, the function `dynamic_update` takes parameters to build and execute a dynamic update query using the MySQL connector for Python.

Backup and Restore:

Example 3: SQL Server Restore from Backup
```

```sql
-- Restore database from backup
USE master;
RESTORE DATABASE YourDatabaseName
FROM DISK = 'C:\Backup\YourDatabaseName.bak'
WITH REPLACE;
```

This SQL Server example demonstrates restoring a database from a backup using the `RESTORE DATABASE` statement with the `WITH REPLACE` option.

Example 4: Python with SQLAlchemy (Backup and Restore)

```python
from sqlalchemy import create_engine

def backup_and_restore_with_sqlalchemy():
    # Connect to databases using SQLAlchemy
    source_engine = create_engine('sqlite:///source.db')
    destination_engine = create_engine('sqlite:///backup.db')

    # Backup database by copying data
    with source_engine.connect() as source_conn, destination_engine.connect() as dest_conn:
        source_data = source_conn.execute('SELECT * FROM my_table').fetchall()
        dest_conn.execute('CREATE TABLE IF NOT EXISTS my_table AS SELECT * FROM source_table')

# Example usage
backup_and_restore_with_sqlalchemy()
```

In this example, we use SQLAlchemy to connect to two SQLite databases and copy data from one table to another, essentially performing a basic form of database backup and restore.