```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import rand, randn, when, col
import mysql.connector

# Crear una sesión de Spark
spark = SparkSession.builder \
    .appName("MySQL Integration") \
    .getOrCreate()

# Parámetros de conexión
mysql_host = "localhost"
mysql_port = 3306
mysql_user = "hive"
mysql_password = "hive_password"
new_database = "pyspark_test_db"

# Crear un DataFrame de prueba
test_data = spark.range(0, 10000) \
    .withColumn("random_value", rand()) \
    .withColumn("normal_distribution", randn()) \
    .withColumn("category", when(col("random_value") < 0.3, "Low")
                            .when(col("random_value") < 0.7, "Medium")
                            .otherwise("High"))

# Mostrar los primeros registros
print("Muestra de los datos generados:")
test_data.show(5)

# Conectar a MySQL
try:
    conn = mysql.connector.connect(
        host=mysql_host,
        port=mysql_port,
        user=mysql_user,
        password=mysql_password,
        database=new_database
    )
    cursor = conn.cursor()

    # Crear la tabla en MySQL
    cursor.execute("""
    CREATE TABLE IF NOT EXISTS test_data (
        id BIGINT PRIMARY KEY,
        random_value DOUBLE,
        normal_distribution DOUBLE,
        category VARCHAR(10)
    )
```

```
    """)

    # Insertar datos en MySQL
    data = test_data.collect()
    for row in data:
        cursor.execute("""
        INSERT INTO test_data (id, random_value, normal_distribution, category)
        VALUES (%s, %s, %s, %s)
        """, (row.id, row.random_value, row.normal_distribution, row.category))

    conn.commit()
    print("Datos de prueba escritos en la tabla 'test_data' de MySQL.")

    # Verificar la escritura leyendo los datos de vuelta
    cursor.execute("SELECT * FROM test_data LIMIT 5")
    result = cursor.fetchall()
    print("\nVerificación de los datos escritos en MySQL:")
    for row in result:
        print(row)

    # Comparar el número de filas
    cursor.execute("SELECT COUNT(*) FROM test_data")
    mysql_count = cursor.fetchone()[0]
    original_count = test_data.count()
    print(f"Número de filas en el DataFrame original: {original_count}")
    print(f"Número de filas en la tabla MySQL: {mysql_count}")
    assert original_count == mysql_count, "El número de filas no coincide"

except mysql.connector.Error as e:
    print(f"Error de MySQL: {e}")
finally:
    if 'conn' in locals() and conn.is_connected():
        cursor.close()
        conn.close()
        print("Conexión a MySQL cerrada.")
```

Este script demuestra cómo integrar PySpark con MySQL para realizar las siguientes acciones:

1. **Crea un DataFrame de prueba en PySpark:** Genera un DataFrame con 10,000 filas y cuatro columnas: `id` (una secuencia numérica), `random_value` (números aleatorios entre 0 y 1), `normal_distribution` (muestras de una distribución normal) y `category` (categorías basadas en los valores de `random_value`).
2. **Se conecta a un servidor MySQL:** Utiliza la librería `mysql.connector` para establecer una conexión con la base de datos `pyspark_test_db` que creaste en el paso anterior. Asegúrate de que los parámetros de conexión (`mysql_host`, `mysql_port`, `mysql_user`, `mysql_password`) sean correctos para tu entorno MySQL.

3. **Crea una tabla en MySQL:** Ejecuta un comando SQL `CREATE TABLE IF NOT EXISTS` para crear una tabla llamada `test_data` en la base de datos MySQL. La estructura de la tabla coincide con el esquema del DataFrame de PySpark.
4. **Inserta datos del DataFrame en la tabla MySQL:**
    - Utiliza la acción `collect()` para traer todos los datos del DataFrame de PySpark al driver como una lista de objetos Row. **Advertencia:** Para DataFrames muy grandes, esto puede consumir mucha memoria en el driver.
    - Itera sobre cada fila de los datos recolectados y ejecuta una sentencia `INSERT INTO` en MySQL para insertar los valores correspondientes en la tabla `test_data`. Se utilizan marcadores de posición (`%s`) y una tupla de valores para prevenir inyecciones SQL.
    - `conn.commit()` confirma los cambios realizados en la base de datos.
5. **Verifica la escritura leyendo datos de MySQL:**
    - Ejecuta una consulta `SELECT * FROM test_data LIMIT 5` para obtener los primeros 5 registros de la tabla MySQL.
    - Imprime estos registros para verificar que los datos se hayan insertado correctamente.
6. **Compara el número de filas:**
    - Cuenta el número de filas en la tabla MySQL utilizando `SELECT COUNT(*)`.
    - Obtiene el número de filas en el DataFrame original de PySpark utilizando `test_data.count()`.
    - Compara ambos recuentos utilizando una aserción para asegurarse de que todos los datos se hayan transferido correctamente.
7. **Manejo de errores y cierre de la conexión:**
    - Se utiliza un bloque `try...except` para capturar cualquier error que pueda ocurrir durante la conexión o las operaciones con MySQL.
    - El bloque `finally` asegura que la conexión a MySQL se cierre correctamente, incluso si ocurre un error.

Puntos Importantes y Consideraciones:

- **Driver JDBC de MySQL para Spark:** Aunque este código utiliza `mysql.connector` para la conexión directa desde el driver de Python para crear la tabla e insertar datos, si quisieras leer o escribir DataFrames de PySpark directamente a MySQL usando las funcionalidades JDBC de Spark (lo cual es más eficiente para grandes volúmenes de datos), necesitarías asegurarte de que el driver JDBC de MySQL esté disponible para Spark. Esto se configura en el `SparkSession` builder con la opción `spark.jars.packages`, como se mostró en el ejemplo anterior.
- **Eficiencia de la Inserción:** La forma en que se insertan los datos en este ejemplo (recolectando todo el DataFrame al driver y luego insertando fila por fila) no es la más eficiente para DataFrames muy grandes. Para grandes cantidades de datos, es preferible utilizar la funcionalidad de escritura JDBC de Spark, que distribuye la escritura entre los workers del cluster.
- **Permisos en MySQL:** Asegúrate de que el usuario MySQL (`hive`) tenga los permisos necesarios para crear tablas e insertar datos en la base de datos `pyspark_test_db`.

Este script proporciona una buena base para entender cómo interactuar entre PySpark y MySQL, aunque para escenarios de producción con grandes volúmenes de datos, se recomendaría explorar las capacidades de lectura y escritura JDBC de Spark para una mayor eficiencia.