```python
import mysql.connector
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, DoubleType,
LongType

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

# Parámetros de conexión MySQL
mysql_config = {
    'user': 'hive',
    'password': 'hive_password',
    'host': 'localhost',
    'database': 'pyspark_test_db',
    'raise_on_warnings': True
}

#  Leer datos
try:
    conn = mysql.connector.connect(**mysql_config)
    cursor = conn.cursor()

    # Ejecutar una consulta
    query = "SELECT * FROM test_data LIMIT 100"
    cursor.execute(query)

    # Obtener los resultados
    results = cursor.fetchall()

    # Obtener los nombres de las columnas
    column_names = [desc[0] for desc in cursor.description]

    print(f"Datos leídos de MySQL: {len(results)} filas")

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

# Definir el esquema para el DataFrame de Spark
# Ajusta los tipos de datos según tu esquema real
schema = StructType([
```

```
    StructField("id", LongType(), True),
    StructField("random_value", DoubleType(), True),
    StructField("normal_distribution", DoubleType(), True),
    StructField("category", StringType(), True)
])

# Crear un DataFrame de Spark a partir de los datos de MySQL
if 'results' in locals():
    df = spark.createDataFrame(results, schema=schema)

    print("DataFrame de Spark creado. Mostrando las primeras 5 filas:")
    df.show(5)

    print("Realizando algunas operaciones con PySpark:")

    # Ejemplo de operación: contar registros por categoría
    df.groupBy("category").count().show()

    # Ejemplo de operación: calcular estadísticas básicas
    df.describe().show()
else:
    print("No se pudieron cargar datos en el DataFrame de Spark.")

# Detener la sesión de Spark
spark.stop()
```

Este código demuestra cómo leer datos desde una tabla MySQL y crear un DataFrame de PySpark a partir de esos datos. Aquí está el análisis detallado:

1. **Importar Librerías:**
    - `mysql.connector`: Para conectar y comunicarse directamente con el servidor MySQL desde el driver de Python.
    - `pyspark.sql.SparkSession`: Para crear la sesión de Spark.
    - `pyspark.sql.types`: Para definir el esquema del DataFrame de Spark.
2. **Crear Sesión de Spark:**
    - Se inicializa una sesión de Spark. **Es importante notar que, para una integración más eficiente y directa de Spark con MySQL (especialmente para grandes conjuntos de datos), se suele utilizar la funcionalidad JDBC de Spark, lo que requiere la inclusión del driver JDBC de MySQL en el classpath de Spark.** Esto se hace mediante la configuración `spark.jars.packages`, como se muestra en la línea:

      Python

      ```
      .config("spark.jars.packages", "mysql:mysql-connector-java:8.0.28")
      ```

Esta línea asegura que Spark descargue el driver JDBC necesario.

3. **Parámetros de Conexión MySQL:**
    - Se define un diccionario `mysql_config` que contiene los parámetros necesarios para conectar al servidor MySQL: `user`, `password`, `host`, `database` (la `pyspark_test_db` que creaste anteriormente) y `raise_on_warnings`.
4. **Leer Datos de MySQL:**
    - Se establece una conexión con el servidor MySQL utilizando la librería `mysql.connector` y la configuración definida.
    - Se crea un cursor para ejecutar consultas SQL.
    - Se ejecuta una consulta `SELECT * FROM test_data LIMIT 100` para obtener los primeros 100 registros de la tabla `test_data`.
    - Los resultados se obtienen con `cursor.fetchall()`.
    - Los nombres de las columnas se extraen de la descripción del cursor.
    - Se imprime un mensaje indicando el número de filas leídas.
5. **Definir Esquema de Spark:**
    - Se define un esquema (`schema`) para el DataFrame de Spark utilizando `StructType` y `StructField`. **Es crucial que los tipos de datos definidos en el esquema coincidan con los tipos de datos de las columnas en la tabla MySQL.** En este caso, se asumen los tipos de datos que probablemente se utilizaron al crear la tabla `test_data`.
6. **Crear DataFrame de Spark:**
    - Se verifica si se leyeron datos de MySQL (`if 'results' in locals():`).
    - Si hay resultados, se crea un DataFrame de Spark (`df`) utilizando `spark.createDataFrame()`, pasando los `results` obtenidos de MySQL y el `schema` definido.
    - Se imprime un mensaje y se muestran las primeras 5 filas del DataFrame creado.
7. **Realizar Operaciones con PySpark:**
    - Se muestran dos ejemplos de operaciones que se pueden realizar con el DataFrame de Spark:
        - `df.groupBy("category").count().show()`: Cuenta el número de registros por cada categoría en la columna "category".
        - `df.describe().show()`: Calcula estadísticas descriptivas (count, mean, stddev, min, max) para las columnas numéricas del DataFrame.
8. **Manejo de Errores y Cierre de Conexión:**
    - Se utiliza un bloque `try...except` para capturar cualquier error que pueda ocurrir durante la conexión o la consulta a MySQL.
    - El bloque `finally` asegura que la conexión a MySQL se cierre correctamente, incluso si ocurre un error.
9. **Detener Sesión de Spark:**
    - Finalmente, se detiene la sesión de Spark para liberar los recursos.

Puntos Importantes y Consideraciones:

- **Driver JDBC de MySQL para Spark (Reiteración):** Para una lectura más eficiente y escalable de grandes cantidades de datos desde MySQL a Spark, se recomienda utilizar la funcionalidad JDBC de Spark en lugar de leer los datos al driver de Python y luego crear el DataFrame. Esto implicaría usar `spark.read.format("jdbc").options(...)` para leer directamente desde MySQL.

- **Mapeo de Tipos de Datos:** Asegúrate de que los tipos de datos definidos en el `schema` de Spark coincidan con los tipos de datos de las columnas correspondientes en la tabla MySQL para evitar problemas de conversión.
- **Rendimiento:** Para grandes tablas en MySQL, la lectura de todos los datos al driver de Python (como se hace con `cursor.fetchall()`) puede ser ineficiente y causar problemas de memoria. En esos casos, la lectura directa a través de JDBC de Spark es la solución preferida.

Este código proporciona una forma básica de transferir datos desde MySQL a PySpark utilizando el driver de Python para MySQL y luego creando un DataFrame de Spark a partir de los resultados. Es útil para conjuntos de datos más pequeños o para entender el proceso fundamental de integración.