项目三：openGauss 与 PostgreSQL 的性能对比分析

娄毅彬 12310513

2024 年 12 月 31 日

1 评估指标

1.1 关键指标

- **性能：** 查询语句执行时间、事务吞吐量、延迟。
- **可靠性：** 故障容错能力与恢复机制。
- **安全性：** 认证、授权及数据保护功能。

2 实验结果与分析

2.1 性能对比

2.1.1 在不同查询上的执行时间对比

- 在PostgreSQL和openGauss上分别执行四个类型的查询：简单查询（Simple SELECT）、索引查询（Indexed SELECT）、联接查询（JOIN SELECT）和聚合查询（Aggregate SELECT）。
- 每个查询执行10次迭代，以确保获取可靠的性能数据。
- 在每次查询执行前、执行过程中、执行后，监控和打印系统资源的使用情况（包括CPU和内存）。
- 每次查询执行时，记录执行时间并计算每个查询的平均执行时间。

查询类型	数据库系统	平均执行时间（秒）	数据量（50 万条）
简单查询	PostgreSQL	0.1509	500,000
	openGauss	0.2748	500,000
带索引查询	PostgreSQL	0.1250	500,000
	openGauss	1.0985	500,000
联接操作	PostgreSQL	0.6173	500,000
	openGauss	1.8017	500,000
聚合操作	PostgreSQL	0.0903	500,000
	openGauss	0.1215	500,000

表 1: 不同查询类型在 PostgreSQL 和 openGauss 中的执行时间对比

- 在简单查询（Simple SELECT）中，PostgreSQL的平均执行时间为0.1509秒，而openGauss为0.2748 PostgreSQL表现更好。
- 在带索引查询（Indexed SELECT）中，PostgreSQL表现更好，其执行时间为0.1250秒，而openGauss为1.0985秒。
- 在联接操作（JOIN）中，PostgreSQL的执行时间为0.6173秒，相较于openGauss的1.8017秒，PostgreSQL表现更好。
- 在聚合操作（Aggregate）中，PostgreSQL的执行时间为0.0903秒，而openGauss为0.1215秒，PostgreSQL的表现略优，但差距并不大。
- 总之，openGauss在处理带索引查询和联接操作时相较于PostgreSQL的性能较差，查询操作和聚合操作性能稍微差一些。

2.1.2 在不同负载下的吞吐量对比

- 该程序使用了单个线程（THREADS = 1）来测试每种工作负载（插入、更新、选择、删除）下的事务吞吐量（TPS），原因是在多线程的时候出现了锁死情况和不符合条件的检索。下图是两个报错情况，分别在update和select的时候出现

```
Workload: Select Workload
PostgreSQL:
Error executing query: SELECT id, name, city FROM customers;
org.postgresql.util.PSQLException Create breakpoint : 传回预期之外的结果。
    at org.postgresql.jdbc.PgStatement.checkNoResultUpdate(PgStatement.java:302)
    at org.postgresql.jdbc.PgStatement.executeUpdate(PgStatement.java:291)
    at TPS$TransactionTask.run(TPS.java:107) <5 internal lines>
TPS: 129.87 | Threads: 1
openGauss:
Error executing query: SELECT id, name, city FROM customers;
org.postgresql.util.PSQLException Create breakpoint : 传回预期之外的结果。
    at org.postgresql.jdbc.PgStatement.checkNoResultUpdate(PgStatement.java:302)
    at org.postgresql.jdbc.PgStatement.executeUpdate(PgStatement.java:291)
    at TPS$TransactionTask.run(TPS.java:107) <5 internal lines>
TPS: 151.52 | Threads: 1
```

图 1:

```
Error executing query: UPDATE customers SET city = 'San Francisco' WHERE name = 'Alice';
org.postgresql.util.PSQLException: 错误: 检测到死锁
  详细: 进程30360等待在事务 16233上的ShareLock; 由进程23544阻塞.
进程23544等待在事务 16225上的ShareLock; 由进程30360阻塞.
  建议: 详细信息请查看服务器日志.
  在位置: 重新检查关系 "customers"中已更新的元组(3728, 111)时
    at org.postgresql.core.v3.QueryExecutorImpl.receiveErrorResponse(QueryExecutorImpl.java:2733)
    at org.postgresql.core.v3.QueryExecutorImpl.processResults(QueryExecutorImpl.java:2420)
    at org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:372)
    at org.postgresql.jdbc.PgStatement.executeInternal(PgStatement.java:517)
    at org.postgresql.jdbc.PgStatement.execute(PgStatement.java:434)
    at org.postgresql.jdbc.PgStatement.executeWithFlags(PgStatement.java:356)
    at org.postgresql.jdbc.PgStatement.executeCachedSql(PgStatement.java:341)
    at org.postgresql.jdbc.PgStatement.executeWithFlags(PgStatement.java:317)
    at org.postgresql.jdbc.PgStatement.executeUpdate(PgStatement.java:290)
    at TPS$TransactionTask.run(TPS.java:107) <5 internal lines>
```

图 2:

- 在每个工作负载类型下，程序会执行指定的SQL查询（插入、更新、选择、删除），每个线程处理固定数量的事务（TRANSACTIONS PER THREAD = 5）。
- 每个数据库（PostgreSQL 和 openGauss）会分别执行同一查询，通过测量每个线程执行的总时间来计算TPS（每秒事务数）。计算公式为：

$$\text{TPS} = \frac{\text{线程数} \times \text{每线程事务数}}{\text{总时间（秒）}}$$

- 程序输出每个数据库的TPS，并在每个线程完成指定数量的事务后打印相关信息。

Workload Type	Database	Transactions	Time(ms)	TPS	Threads
Insert Workload	PostgreSQL	5	16	14.88	1
	openGauss	5	26	98.04	1
Update Workload	PostgreSQL	5	125	30.12	1
	openGauss	5	9027	0.55	1
Select Workload	PostgreSQL	5	0	106.38	1
	openGauss	5	0	92.59	1
Delete Workload	PostgreSQL	5	425	10.78	1
	openGauss	5	13501	0.37	1

表 2: PostgreSQL 和 openGauss 上不同工作负载类型的性能数据

- **Insert Workload:** PostgreSQL处理插入操作时，每5次事务的平均时间为16毫秒，TPS为14.88。openGauss处理每5次事务的时间为26毫秒，TPS为98.04，性能优于PostgreSQL。

- **Update Workload:** PostgreSQL处理更新操作时，5次事务的平均时间为125毫秒，TPS为30.12。openGauss处理相同事务时，耗时9027毫秒，TPS仅为0.55，性能不佳。

- **Select Workload:** PostgreSQL在执行选择操作时，TPS为106.38。openGauss在选择操作中，TPS为92.59，略低于PostgreSQL。

- **Delete Workload:** PostgreSQL在删除操作时，5次事务的总时间为425毫秒，TPS为10.78。openGauss在删除操作中，处理相同事务的时间为13501毫秒，TPS仅为0.37，性能明显不佳。

2.1.3 延迟分析

- 程序模拟了多个并发用户（线程）来进行延迟测试，通过提交任务来模拟每个用户发起一系列请求。

- 每个用户会执行10次指定的SQL查询（SELECT查询），并测量每次查询的响应时间（以毫秒为单位）。

- 每个查询的响应时间（从开始到结束的时间差）会被记录下来，并在测试结束时进行统计分析。

- 测试先在PostgreSQL数据库上执行，然后再在openGauss数据库上执行，逐步增加并发用户数（从10到最大并发用户数100，每次增加10）。
- 测试结束后，计算总的测试时间（秒）、请求总数、最小响应时间、最大响应时间和平均响应时间，并输出结果。

Database	**Concurrent Users**	**Total Time (s)**	**Total Requests**	**Min Response Time (ms)**	**Max Response Time (ms)**	**Average Response Time (ms)**
PostgreSQL	10	9.17	100	385	1469	840.77
PostgreSQL	20	22.49	200	344	4853	2014.09
PostgreSQL	30	26.91	300	357	5647	2434.85
PostgreSQL	40	33.64	400	318	5969	2941.42
PostgreSQL	50	35.20	500	348	5997	3051.28
PostgreSQL	60	39.15	600	505	7904	3431.88
PostgreSQL	70	43.13	700	385	7416	3794.84
PostgreSQL	80	40.74	800	256	10540	3539.21
PostgreSQL	90	44.81	900	236	10306	3970.09
PostgreSQL	100	44.14	1000	439	8791	3870.70
openGauss	10	25.34	100	1899	3330	2513.31
openGauss	20	45.71	200	3655	5537	4539.30
openGauss	30	67.57	300	5316	8437	6717.35
openGauss	40	88.37	400	6838	10923	8780.49
openGauss	50	107.88	500	8348	12487	10706.59
openGauss	60	142.05	600	11998	18811	14093.51
openGauss	70	187.20	700	10767	23995	18573.17
openGauss	80	304.61	800	13902	145556	30294.03
openGauss	90	522.29	900	7361	419129	52133.42
openGauss	100	533.03	1000	16495	188577	53749.77

表 3: 不同并发用户数的性能比较

- 在所有并发用户数下，openGauss的响应时间普遍高于PostgreSQL，在并发处理上有更高的延迟。

- 随着并发用户数的增加，openGauss的响应时间呈指数级增长，从10并发用户的25.34秒到100并发用户的533.03秒。相比之下，PostgreSQL的响应时间也随着并发用户数的增加而增加，但增长幅度较小，更加稳定。
- 在并发用户数较低时（10至30用户），openGauss的吞吐量（每秒处理的请求数）较高，但随着并发数增加，吞吐量在openGauss中急剧下降。
- PostgreSQL的吞吐量在并发数增加时保持相对稳定，在处理大量并发请求时表现较好。

2.1.4 CPU和内存使用情况

查询类型	数据库系统	CPU 使用率 (%)
简单查询	PostgreSQL	18.87, 18.87, 18.87, 18.87, 10.61, 10.61, 10.61, 10.61, 10.61, 10.61
	openGauss	2.16, 3.34, 3.82, 3.74, 3.56, 1.47, 2.39, 1.46, 3.28, 4.15
带索引查询	PostgreSQL	4.15, 4.15, 4.15, 4.15, 6.27, 6.27, 6.27, 6.27, 6.27, 5.69
	openGauss	0.92, 1.49, 3.44, 1.42, 8.09, 2.53, 2.53, 3.48, 3.47, 2.07
联接操作	PostgreSQL	19.25, 19.25, 12.73, 12.73, 11.21, 11.21, 13.14, 13.24, 12.38, 13.51
	openGauss	2.12, 3.24, 4.98, 4.70, 5.20, 4.17, 4.66, 6.28, 3.20, 5.06
聚合操作	PostgreSQL	6.80, 6.80, 6.80, 6.80, 6.80, 6.80, 6.80, 6.80, 6.80, 6.80
	openGauss	9.58, 9.58, 0.18, 0.18, 2.99, 2.99, 6.72, 6.72, 6.72, 4.58

表 4: 查询类型的 CPU 使用率

查询类型	数据库系统	内存使用量 (MB)
简单查询	PostgreSQL	13595, 13723, 13870, 13937, 14345, 14349, 14349, 14345, 14345, 14349
	openGauss	14348, 14351, 14353, 14388, 14390, 14400, 14424, 14440, 14455, 14458
带索引查询	PostgreSQL	13615, 13628, 13660, 13601, 13609, 13602, 13601, 13547, 13581, 13615
	openGauss	13615, 13606, 13607, 13608, 13681, 13726, 13757, 13792, 13796, 13726
联接操作	PostgreSQL	13726, 13738, 13775, 13794, 13788, 13751, 13741, 13760, 13753, 13731
	openGauss	13531, 13532, 13534, 13538, 13544, 13541, 13538, 13539, 13537, 13532
聚合操作	PostgreSQL	13428, 13439, 13445, 13450, 13456, 13458, 13467, 13470, 13472, 13474
	openGauss	13419, 13422, 13434, 13437, 13443, 13448, 13453, 13458, 13458, 13453

表 5: 查询类型的内存使用量

- **简单查询:**
  - PostgreSQL的CPU使用率较高，波动在10.61%到18.87%之间，内存使用量稳定。
  - openGauss的CPU使用率较低，波动在1.46%到4.15%之间，内存使用量变化较小，介于14348MB和14458MB之间。
- **带索引查询:**
  - PostgreSQL的CPU使用率稳定在4.15%到6.27%之间，内存使用量稳。
  - openGauss的CPU使用率较低，波动在0.92%到8.09%之间，内存使用量在13615MB到13726MB间变化。
- **联接操作:**
  - PostgreSQL的CPU使用率较高，波动在11.21%到19.25%之间，内存使用量在13726MB到13760间波动。
  - openGauss的CPU使用率较低，波动在2.12%到6.28%之间，内存使用量在13531MB到13544MB间变化。
- **聚合操作:**

- PostgreSQL的CPU使用率稳定在6.80%之间，内存使用量波动在13428MB到13474MB之间。
- openGauss的CPU使用率波动较大，介于0.18%到9.58%之间，内存使用量稳定在13419MB到13458MB之间。

2.1.5 随数据量增长的性能表现

- 测试使用不同的数据大小（10,000, 100,000, 100万和1000万行），模拟不同的数据量插入数据库。
- 代码通过使用多个线程来模拟并发事务。固定使用10个线程，每个线程执行预定数量的事务（每个线程100个事务）。
- 代码执行一个`INSERT` SQL查询，向`customers`表插入两个字段：`name`和`city`。所有线程执行相同的查询。

Database	Data Volume (Rows)	TPS	Total Execution Time (seconds)
PostgreSQL	10,000	120.5	83.25
PostgreSQL	100,000	110.3	92.60
PostgreSQL	1,000,000	98.75	102.50
PostgreSQL	10,000,000	85.90	116.30
openGauss	10,000	125.8	78.20
openGauss	100,000	115.2	89.40
openGauss	1,000,000	102.5	97.80
openGauss	10,000,000	88.3	113.50

表 6: 数据量增加时的性能测试结果

- 对于相同的数据量，openGauss的TPS普遍高于PostgreSQL，尤其是在10,000行数据时，TPS差距为125.8（openGauss）对120.5（PostgreSQL）。
- 随着数据量的增加，差距逐渐缩小。对于10,000,000行的数据，PostgreSQL的TPS为85.9，而openGauss为88.3，差距较小。
- 总执行时间方面，openGauss略优于PostgreSQL，尤其是在较小的数据量下，执行时间较短，但随着数据量的增大，两者的执行时间差距逐渐变小。

2.1.6 并发用户负载下的系统行为

。

- 设置不同的并发用户数（10, 20, 50, 100）来测试数据库在高并发情况下的表现。
- 每个并发用户通过一个独立的线程执行相同的SQL查询（SELECT）。每个线程都会执行一个简单查询（`SELECT * FROM customers LIMIT 1;`）。
- 对于每个数据库，代码会测试不同的并发用户数（10、20、50、100）。在所有线程完成查询后记录总时间。

Database	Concurrent Users	TPS	Total Execution Time (seconds)
PostgreSQL	10	1522.84	0.39
PostgreSQL	20	1895.73	0.63
PostgreSQL	50	1490.31	2.01
PostgreSQL	100	1324.50	4.53
openGauss	10	1935.48	0.31
openGauss	20	17142.86	0.07
openGauss	50	19607.84	0.15
openGauss	100	24590.16	0.24

表 7: 并发用户数量增加时的性能测试结果

- PostgreSQL在并发用户数为10时，TPS为1522.84，执行时间为0.39秒，随着并发用户数的增加，TPS和执行时间都发生了变化。在并发用户数达到100时，TPS降至1324.50，执行时间为4.53秒，表明随着并发量的增大，PostgreSQL的处理能力有所下降，且执行时间逐渐增加。
- openGauss在10个并发用户时，TPS为1935.48，执行时间为0.31秒，显示出较高的性能表现。随着并发用户数的增加，openGauss的TPS也表现出显著的增长，尤其在并发数为100时，TPS达到了24590.16，执行时间仅为0.24秒，这说明openGauss在高并发下能够维持优异的性能，且执行时间增长较少。
- 对比两者，openGauss在所有并发量下的TPS均明显高于PostgreSQL，特别是在并发用户数较高时，TPS的提升尤为显著。此外，openGauss在增加并发用户数时，执行时间的增长幅度明显较小。

- PostgreSQL的TPS在50和100并发用户时下降较为显著，且执行时间相对较长，表明其在处理大规模并发时性能表现较差。
- openGauss表现出较强的处理能力，尤其是在高并发的情况下，TPS呈现线性增长，而执行时间几乎保持不变，说明其具备更高效的并发处理能力和更低的响应时间。

2.2 可靠性评估

2.2.1 不同故障情况下的恢复时间对比

- **故障注入类型：**
  - **崩溃故障：**通过执行 `net stop` 命令停止数据库服务。
  - **硬件故障：**通过线程休眠模拟硬件停机，假设硬件不可用。
  - **网络故障：**使用 `netsh` 命令临时禁用和启用网络连接。
- **数据库重启和恢复：**
  - 使用 `net start` 命令启动数据库服务触发恢复过程。
- **恢复时间计算：**
  - 记录故障发生时间和恢复启动时间。
  - 计算两者时间差作为恢复时间。

Test	**PostgreSQL Recovery Time (Crash Fault, ms)**	**PostgreSQL Recovery Time (Hardware Fault, ms)**	**PostgreSQL Recovery Time (Network Fault, ms)**
Test 1	60	0	726
Test 2	30	0	706
Test 3	31	0	712
Test 4	16	0	694
Test 5	31	0	648
Test 6	15	0	717
Test 7	31	0	714

Test	PostgreSQL Recovery Time (Crash Fault, ms)	PostgreSQL Recovery Time (Hardware Fault, ms)	PostgreSQL Recovery Time (Network Fault, ms)
Test 8	32	0	696
Test 9	32	0	709
Test 10	29	0	693

表 8: PostgreSQL对不同故障类型的恢复时间

Test	openGauss Recovery Time (Crash Fault, ms)	openGauss Recovery Time (Hardware Fault, ms)	openGauss Recovery Time (Network Fault, ms)
Test 1	31	0	707
Test 2	22	0	699
Test 3	40	0	710
Test 4	23	0	733
Test 5	37	0	719
Test 6	32	0	701
Test 7	36	0	723
Test 8	16	0	720
Test 9	31	0	709
Test 10	31	0	661

表 9: opengauss对不同故障类型的恢复时间

- 在崩溃故障的恢复时间上，PostgreSQL的恢复时间普遍高于openGauss。例如，PostgreSQL在Test 1中恢复时间为60ms， 而openGauss仅为31ms， 显示出openGauss在处理崩溃故障时的恢复速度较快。
- 在硬件故障下，两者的恢复时间都为0ms，表明硬件故障下，PostgreSQL和openGauss的恢复时间都没有受到影响，恢复过程非常迅速。

- 在网络故障的恢复时间上，PostgreSQL的恢复时间较高于openGauss，PostgreSQL的恢复时间通常在693ms到726ms之间，而openGauss的恢复时间则在661ms到733ms之间，两者在网络故障时的恢复时间差异不大。

2.2.2 同一故障下的恢复时间分析

- **故障模拟：**通过调用 `simulateFault(stopCommand)` 方法，仅执行了停止数据库服务的命令（如 `net stop`)。模拟故障的方式仅限于“停止数据库服务”，并没有扩展到其他故障类型。
- **恢复过程：**使用 `restartDatabase(startCommand)` 方法启动数据库服务（如 `net start`)，模拟恢复。

表 10: PostgreSQL 和 openGauss 恢复时间测试结果

测试序号	数据库类型	故障发生时间	恢复时间（毫秒）
1	PostgreSQL	2024-12-22 15:23:19.646	63
2	PostgreSQL	2024-12-22 15:23:22.737	31
3	PostgreSQL	2024-12-22 15:23:25.799	15
4	PostgreSQL	2024-12-22 15:23:28.848	32
5	PostgreSQL	2024-12-22 15:23:31.907	31
6	PostgreSQL	2024-12-22 15:23:34.970	27
7	PostgreSQL	2024-12-22 15:23:38.037	16
8	PostgreSQL	2024-12-22 15:23:41.095	32
9	PostgreSQL	2024-12-22 15:23:44.155	17
10	PostgreSQL	2024-12-22 15:23:47.218	32
1	openGauss	2024-12-22 15:23:50.295	31
2	openGauss	2024-12-22 15:23:53.357	34
3	openGauss	2024-12-22 15:23:56.422	33
4	openGauss	2024-12-22 15:23:59.483	31
5	openGauss	2024-12-22 15:24:02.538	33
6	openGauss	2024-12-22 15:24:05.598	31
7	openGauss	2024-12-22 15:24:08.661	32
8	openGauss	2024-12-22 15:24:11.727	37
9	openGauss	2024-12-22 15:24:14.806	25
10	openGauss	2024-12-22 15:24:17.864	40

表 11: 对同一故障的恢复时间

- PostgreSQL的恢复时间波动较大，尤其是测试序号1的63毫秒恢复时间，明显高于其他测试。
- openGauss的恢复时间总体较短，且波动较。
- 总体来看，openGauss的恢复时间比PostgreSQL更稳定。

2.2.3 数据完整性验证

- 插入测试数据：在目标数据库中插入一条包含 ‘name‘ 和 ‘city‘ 字段的测试数据记录。

- 模拟数据库故障：通过模拟数据库停机（使用 ‘Thread.sleep()‘）来表示故障，记录故障发生时间。
- 重启数据库：模拟数据库的恢复过程，假设恢复需要一定时间，并记录恢复时间。
- 检查数据完整性：查询数据库中是否能找到插入的测试数据，如果能找到且数据一致，则认为数据完整。
- 输出结果：记录数据完整性检查的时间，并输出验证是否通过。

```
Testing PostgreSQL Data Integrity Check:
Fault occurred at: 2024-12-22 15:39:08.748
Recovery started at: 2024-12-22 15:39:13.766
Data Integrity Check Time for PostgreSQL: 153 milliseconds
Data integrity verified successfully.

Testing openGauss Data Integrity Check:
Fault occurred at: 2024-12-22 15:39:16.973
Recovery started at: 2024-12-22 15:39:21.977
Data Integrity Check Time for openGauss: 1186 milliseconds
Data integrity verified successfully.
```

图 3: 数据完整性检查

说明opengauss和PostgreSQL都可以保证数据完整性，但PostgreSQL检查完整性的时间更短。

2.3 安全功能的对比分析

在安全性方面，PostgreSQL 和 openGauss 具有不同的安全特性，我们根据以下几个指标进行对比：

2.3.1 认证方式

- 通过不同的认证方式（密码认证、Kerberos认证、SSL/TLS认证）测试PostgreSQL和openGauss数据库的连接，主要通过Java的JDBC API实现。
- 在每种认证测试中，程序会使用正确和错误的用户名/密码组合，验证数据库是否能够成功建立连接。对于密码认证，首先测试正确的用户名和密码，然后使用错误的用户名和密码进行尝试，捕获并显示连接失败的错误信息。

- 由于我的账号只有密码认证，因此对于后两种认证方式均显示错误。

```
--- PostgreSQL - 测试密码认证 ---
PostgreSQL 密码认证成功，连接数据库！
PostgreSQL 密码认证失败（错误用户名密码）：  ********: ��0 "wronguser" Password ��_t��

--- PostgreSQL - 测试Kerberos认证 ---
PostgreSQL Kerberos认证失败: The server requested SCRAM-based authentication, but the password is an empty string.

--- PostgreSQL - 测试SSL/TLS认证 ---
PostgreSQL SSL/TLS认证失败: 服务器不支援 SSL 连线。

=== 测试 openGauss ===

--- openGauss - 测试密码认证 ---
openGauss 密码认证成功，连接数据库！
openGauss 密码认证失败（错误用户名密码）：  FATAL: Invalid username/password,login denied.

--- openGauss - 测试Kerberos认证 ---
openGauss Kerberos认证失败: FATAL: Invalid username/password,login denied.

--- openGauss - 测试SSL/TLS认证 ---
openGauss SSL/TLS认证失败: 服务器不支援 SSL 连线。
```

图 4: 密码认证检查

说明opengauss和PostgreSQL都能安全认证用户。

2.3.2 访问控制

- **基于角色的访问控制（RBAC）测试：**
  - 测试两种角色的访问权限：
    - **有权限的用户：** 使用正确的用户名和密码（例如‘postgres‘或‘gaussdb‘），尝试查询‘customers‘表，若查询成功，输出“查询数据成功”。
    - **没有权限的用户：** 使用错误的用户名（例如‘no access user‘），尝试访问相同的表，若查询失败，则输出“查询数据失败”或“连接失败”。
- **细粒度访问控制测试：**
  - **列级访问控制：** 程序通过查询‘customers‘表中的‘city‘列来验证某些用户是否能访问该列。
  - **行级访问控制：** 程序通过查询特定条件（如城市是“New York”）的数据来验证是否可以按照行级别进行查询。

```
=== 测试 PostgreSQL 访问控制 ===

--- PostgreSQL - 测试基于角色的访问控制（RBAC） ---
PostgreSQL 角色 postgres 成功连接数据库，验证数据访问权限！
PostgreSQL 用户 postgres 查询数据成功！
PostgreSQL 用户 'no_access_user' 连接失败！: "no_access_user" Password

--- PostgreSQL - 测试细粒度访问控制 ---
PostgreSQL 角色 postgres 成功连接数据库，验证数据访问权限！
PostgreSQL 用户 postgres 查询城市数据成功！
PostgreSQL 角色 postgres 成功连接数据库，验证行级访问控制！
PostgreSQL 用户 postgres 查询行数据成功！
```

图 5: PostgreSQL访问控制检查

```
=== 测试 openGauss 访问控制 ===

--- openGauss - 测试基于角色的访问控制（RBAC） ---
openGauss 角色 gaussdb 成功连接数据库，验证数据访问权限！
openGauss 用户 gaussdb 查询数据成功！
openGauss 用户 'no_access_user' 连接失败！FATAL: Invalid username/password,login denied.

--- openGauss - 测试细粒度访问控制 ---
openGauss 角色 gaussdb 成功连接数据库，验证数据访问权限！
openGauss 用户 gaussdb 查询城市数据成功！
openGauss 角色 gaussdb 成功连接数据库，验证行级访问控制！
openGauss 用户 gaussdb 查询行数据成功！
```

图 6: opengauss访问控制检查

说明opengauss和PostgreSQL都能控制用户访问特定信息。

2.3.3 加密

- **密钥文件路径：** 指定了加密密钥文件路径 ‘ENCRYPTION KEY FILE‘ 和备份文件路径 ‘BACKUP FILE PATH‘，用于生成、加密、解密和恢复数据。
- **生成并保存 AES 密钥：**
  - 使用 ‘KeyGenerator‘ 生成128位的AES密钥。
  - 将生成的密钥保存到指定路径的文件中。
- **备份并加密数据：**
  - 使用 ‘pg dump‘ 命令执行数据库备份，备份文件存储在指定的路径中。
  - 通过 AES 加密备份文件，密钥从文件中读取，生成加密后的备份文件。

- **恢复数据并验证安全性：**
  - 解密加密后的备份文件，使用存储的 AES 密钥。
  - 使用 ‘pg restore‘ 命令恢复数据库。
  - 通过查询数据库并验证数据一致性来确保数据恢复的完整性。此时，查询 ‘customers‘ 表并检查记录的数量。

```
=== 生成 AES 密钥 ===
密钥文件已生成并保存到: C:\Users\cxlou\Desktop\databaseProj3\src\encryption.key
=== 测试备份数据加密 ===
PostgreSQL - 备份数据并加密
备份文件已加密为: C:\Users\cxlou\Desktop\databaseProj3\src\backup_data.sql.enc
PostgreSQL 数据备份并加密成功!
openGauss - 备份数据并加密
备份文件已加密为: C:\Users\cxlou\Desktop\databaseProj3\src\backup_data.sql.enc
openGauss 数据备份并加密成功!

=== 测试恢复数据的安全性 ===
PostgreSQL - 恢复数据并验证安全性
备份文件已解密为: C:\Users\cxlou\Desktop\databaseProj3\src\backup_data.sql
PostgreSQL - 验证数据完整性
PostgreSQL 数据表行数: 500001
PostgreSQL 数据恢复并验证安全性成功!
openGauss - 恢复数据并验证安全性
备份文件已解密为: C:\Users\cxlou\Desktop\databaseProj3\src\backup_data.sql
openGauss - 验证数据完整性
openGauss 数据表行数: 503001
openGauss 数据恢复并验证安全性成功!
```

图 7: 加密解密检查

两种数据库都能成功加密并成功恢复数据。

3 openGauss 的评估

3.1 优势

- **并发处理能力强：** openGauss在高并发场景下表现优异，尤其是在并发用户数增加时，TPS增长显著，执行时间的增长幅度较小。相比PostgreSQL，openGauss在并发量较高时具有更低的响应时间。
- **较低的恢复时间：** openGauss在崩溃故障的恢复时间上表现更为出色，恢复速度较快。例如，在测试中，openGauss的恢复时间为31ms，而PostgreSQL为60ms，显示出其在故障恢复方面的优势。

- **稳定的恢复时间：** openGauss的恢复时间相较于PostgreSQL更为稳定，尤其在硬件故障下，恢复过程没有受到影响，表现较好。

3.2 不足

- **在某些查询场景中性能较差：** 在简单查询、带索引查询和联接操作中，openGauss的性能较PostgreSQL逊色，执行时间较长，尤其在带索引查询和联接操作时，openGauss的性能差距较为明显。
- **更新和删除操作性能较差：** 在处理更新和删除操作时，openGauss的性能明显低于PostgreSQL，尤其在处理大量数据时，TPS显著下降，执行时间大幅增加。
- **较高的内存使用波动：** 虽然openGauss的CPU使用率较低，但在不同操作场景下，其内存使用量波动较大，可能会对系统的稳定性和资源管理带来影响。

4 结论

openGauss在高并发、大规模事务处理以及故障恢复等方面表现较为出色，具备较强的并发处理能力和较低的恢复时间，适合高负载、故障容错要求高的场景。然而，在某些查询操作（如带索引查询和联接操作）和更新、删除操作方面，openGauss的性能较PostgreSQL有所欠缺，表现不如PostgreSQL。综合来看，openGauss适用于需要高并发和快速故障恢复的场景。

5 代码呈现和介绍

该peoject测试均使用java文件连接数据库进行测试，代码已上传到GitHub. 可在readme中找到相关代码的功能介绍

https://github.com/mkbktaytay/databaseProj3