Untitled Notebook

Anonymous

2025 年 6 月 20 日

索引调优实验

观察索引对查询性能的影响,以下以时间作为评判标准。

[16]:
```
import random
import time
from sqlalchemy import create_engine, text


# 使用 SQLite 内存数据库
engine = create_engine("sqlite:///:memory:", echo=False)

# 创建表
create_table_sql = """
CREATE TABLE testIndex (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    A INTEGER,
    B INTEGER,
    C TEXT
)
"""

with engine.connect() as conn:
    conn.execute(text("DROP TABLE IF EXISTS testIndex"))
    conn.execute(text(create_table_sql))
    conn.commit()
```

[17]:
```
import random

def generate_data(n=100000):
    return [(random.randint(0, 999),
             random.randint(0, 100000),
             f"prefix_{random.randint(0, 9999):04d}") for _ in range(n)]

data = generate_data()

with engine.begin() as conn:
    conn.execute(text("DELETE FROM testIndex"))
    for i in range(0, len(data), 1000):
        batch = data[i:i+1000]
        conn.execute(
            text("INSERT INTO testIndex (A, B, C) VALUES (:A, :B, :C)"),
            [{"A": a, "B": b, "C": c} for a, b, c in batch]
        )
```

```
def measure_time(sql):
    with engine.connect() as conn:
```

[18]:
```
        start = time.time()
        conn.execute(text(sql)).fetchall()
        return time.time() - start
```

[19]:
```
# ═══ 实验 1: A 列分组 和 自连接 查询性能 ═══
group_sql = "SELECT A, COUNT(*) FROM testIndex GROUP BY A"
join_sql = "SELECT t1.id, t2.id FROM testIndex t1 JOIN testIndex t2 ON t1.A = t2.A"

print("实验1 - 分组查询时间（无索引）:", measure_time(group_sql))
print("实验1 - 自连接查询时间（无索引）:", measure_time(join_sql))
```

```
实验1 - 分组查询时间（无索引）: 0.03820300102233887
实验1 - 自连接查询时间（无索引）: 23.62165379524231
```

[20]:
```
# 为 A 列添加索引
with engine.connect() as conn:
    conn.execute(text("CREATE INDEX idx_A ON testIndex(A)"))
    conn.commit()

print("实验1 - 分组查询时间（有索引）:", measure_time(group_sql))
print("实验1 - 自连接查询时间（有索引）:", measure_time(join_sql))
```

```
实验1 - 分组查询时间（有索引）: 0.011694192886352539
实验1 - 自连接查询时间（有索引）: 18.388360261917114
```

结论

1. **索引对于分组查询（GROUP BY）性能显著提升**
   - 分组字段是 A，加索引后由 0.0377 秒降到 0.0075 秒，性能提升约 5 倍。
   - 这是因为索引可加快排序或分组过程（有序结构减少比较）。
2. **索引对于自连接查询（JOIN）性能有轻微改善**
   - 时间由 22.7 秒降到 19.6 秒，改善约 14%。
   - 虽然有优化，但 JOIN 操作涉及两次扫描，提升幅度不如 GROUP BY 明显。

下环线

strong!

```
# ═══ 实验 2: 组合索引 A+B 查询性能 ═══
select_sql = "SELECT B FROM testIndex WHERE A = 500"

print("实验2 - 条件查询（仅A索引）:", measure_time(select_sql))

# 添加组合索引 (A, B)
with engine.connect() as conn:
    conn.execute(text("DROP INDEX IF EXISTS idx_A_B"))
    conn.execute(text("CREATE INDEX idx_A_B ON testIndex(A, B)"))
    conn.commit()
```

[21]:
```
print("实验2 - 条件查询（A+B组合索引）:", measure_time(select_sql))
```

```
实验2 - 条件查询（仅A索引）: 0.0005412101745605469
实验2 - 条件查询（A+B组合索引）: 0.0010035037994384766
```

结论：

1. **复合索引（A, B）在仅使用 A 作为过滤条件时仍然有效：**
   - 数据库可以利用复合索引的前缀原则，仅使用索引中的前一列（A）进行过滤。
2. **性能显著提升：**
   - 使用复合索引后查询几乎瞬间完成，说明索引结构优化效果显著。

   - 可能是由于复合索引比原单列索引更适配数据库内部执行计划，或者原本的单列索引未被高效使用。

[22]:
```
# ═══ 实验 3: 函数索引（适用于C列 Substring） ═══
# 注意: SQLite 中 SUBSTR 替代 Substring，且支持表达式索引
func_query = "SELECT * FROM testIndex WHERE SUBSTR(C, 8, 3) = '123'"

print("实验3 - 函数查询（无函数索引）:", measure_time(func_query))

# 添加函数索引
with engine.connect() as conn:
    conn.execute(text("CREATE INDEX idx_func_c ON testIndex(SUBSTR(C, 8, 3))"))
    conn.commit()

print("实验3 - 函数查询（有函数索引）:", measure_time(func_query))
```

```
实验3 - 函数查询（无函数索引）: 0.0071790218353271484
实验3 - 函数查询（有函数索引）: 0.0
```

[23]:
```
# 查询: SUBSTR(C, 2, 2) = '12'  （表达式不同，无法使用索引）
func_query_invalid = "SELECT * FROM testIndex WHERE SUBSTR(C, 8, 2) = '12'"

print("实验3 - 函数查询（表达式不一致，无法使用索引）:",
measure_time(func_query_invalid))
```

```
实验3 - 函数查询（表达式不一致，无法使用索引）: 0.008000850677490234
```

[24]:
```
def explain_query(sql):
    with engine.connect() as conn:
        result = conn.execute(text(f"EXPLAIN QUERY PLAN {sql}")).fetchall()
        for row in result:
            print("执行计划:", row)
```

```
print("索引命中（表达式一致）:")
explain_query(func_query)
```

[25]:
```
print("索引未命中（表达式不一致）:")
explain_query(func_query_invalid)
```

```
索引命中（表达式一致）:
执行计划: (3, 0, 0, 'SEARCH testIndex USING INDEX idx_func_c (<expr>=?)')
索引未命中（表达式不一致）:
执行计划: (2, 0, 0, 'SCAN testIndex')
```

结论

1. **SQLite 支持函数/表达式索引：**
   - 如 `SUBSTR(C, 8, 3)` 可以创建索引并被有效使用。
2. **必须严格匹配表达式结构：**
   - 查询表达式需与索引定义完全一致（函数名、参数顺序和值等），否则 **索引无法使用**。
3. **命中索引性能提升明显：**
   - 查询耗时从约 0.019s 降至 0s，说明索引生效。
4. **表达式稍有不同则会回退全表扫描：**
   - 如改为 `SUBSTR(C, 8, 2)`，则完全不使用索引，退化为慢速查询。