ORDER BY random()的含义

资料：https://www.educba.com/postgresql-order-by-random/
https://neon.com/postgresql/postgresql-math-functions/postgresql-random
https://www.geeksforgeeks.org/how-to-select-random-row-in-postgresql/

`ORDER BY random()` 使用 `random()` 函数为每行生成随机数（0-1 之间），然后按此排序结果集。 例如，`SELECT * FROM student ORDER BY random() LIMIT 5;` 随机选择 5 行学生数据。

适合小型表，但大型表效率低，可考虑 `TABLESAMPLE BERNOULLI`。对于大表，`ORDER BY random()` 需排序全表，耗时长。替代方案如按 ID 范围随机选择，效率更高。

理解 `ORDER BY random()` 的含义

在 PostgreSQL 中，`ORDER BY random()` 是一种用于从表中随机选择行的 SQL 查询方法，特别适用于需要随机排序结果集的场景。以下是对其工作原理、应用和相关细节的全面分析。

ORDER BY 子句

`ORDER BY` 是 SQL 中的一个子句，用于对查询结果集进行排序。通常，它根据指定的列（如 `name` 或 `salary`）对结果集进行升序或降序排序。例如，`SELECT * FROM student ORDER BY name;` 会根据学生的姓名排序。

random() 函数

PostgreSQL 的 `random()` 函数是一个内置函数，它为每行返回一个介于 0 和 1 之间的随机浮点数（例如 0.12345 或 0.78901）。这个函数是伪随机数生成器，基于确定性算法，但每次查询都会生成新的随机数。

ORDER BY random() 的工作原理

当在 `ORDER BY` 子句中使用 `random()` 时，PostgreSQL 会为表中的每行生成一个随机数，然后根据这些随机数对所有行进行排序。由于随机数是随机的，排序的结果也是随机的，即结果集的行顺序是不可预测的。

结合 LIMIT 子句的示例

用户提供的查询 `SELECT * FROM student ORDER BY random() LIMIT 5;` 是一个典型的例子，其执行过程如下：

1. 生成随机数：PostgreSQL 为 `student` 表中的每行生成一个随机数（0 到 1 之间）。
2. 排序：根据这些随机数对所有行进行排序，排序方式可以是升序或降序（默认升序）。
3. 限制返回行数：由于使用了 `LIMIT 5`，最终只返回排序后的前 5 行，因此结果是表中随机选择的 5 行。

例如，如果 `student` 表有 100 行，`random()` 可能为第一行生成 0.5，第二行生成 0.2，依此类推。排序后，前 5 行可能是任意顺序的 5 行，具体取决于随机数。

适用场景

1.小型表：对于小型或中型表（例如几千行），`ORDER BY random()` 是简单有效的随机选择方法。

2.随机样本：常用于需要随机抽样数据的场景，如数据分析、内容生成或游戏开发中的随机内容展示。

性能考虑

尽管 `ORDER BY random()` 简单易用，但它在大型表上的性能可能较差。原因如下：
1.全表扫描和排序：PostgreSQL 需要为每行生成随机数，并对整个表进行排序。这涉及全表扫描和排序操作，时间复杂度较高，特别是在表行数达到百万级时。
2.资源消耗：排序可能需要额外的内存和磁盘 I/O，尤其是当结果集很大时，可能会写入临时文件，增加性能开销。

对于大型表，`ORDER BY random()` 的执行时间可能显著增加。例如，在百万行表上，执行 `EXPLAIN ANALYZE SELECT * FROM big_data ORDER BY RANDOM() LIMIT 1;` 可能需要几十秒，具体取决于表大小和硬件。

替代方法（适用于大型表）

为了提高性能，PostgreSQL 提供了更高效的随机选择方法：
1.TABLESAMPLE BERNOULLI：这是一种基于 Bernoulli 分布的采样方法，可以快速从表中抽取随机样本。例如，`SELECT * FROM student TABLESAMPLE BERNOULLI(10) LIMIT 5;` 可以随机选择 10% 的行，然后限制为 5 行。
2.按 ID 范围随机选择：如果表有一个连续的 ID 列，可以先计算 ID 的最大值，然后用 `FLOOR(RANDOM() * max_id) + 1` 生成随机 ID，再查询对应行。例如，`SELECT * FROM student WHERE id = FLOOR(RANDOM() * (SELECT MAX(id) FROM student)) + 1;`。这种方法假设 ID 是连续的，且分布均匀，否则随机性可能受影响。

上述方法在大型表上的执行时间可以缩短到毫秒级，远优于 `ORDER BY random()`。

实际应用中的注意事项

1.随机性与性能的权衡：`ORDER BY random()` 提供完全随机的结果，但性能成本高。对于需要“足够随机”而非“完全随机”的场景，可以考虑上述替代方法。
2.可重复性：如果需要生成相同的随机结果，可以在查询前使用 `SETSEED(value)` 设置随机种子，但这通常用于测试环境。

随机选择方法的比较

方法	适用场景	性能	随机性	实现复杂度
ORDER BY random()	小型表，简单需求	低（大型表慢）	完全随机	低
TABLESAMPLE BERNOULLI	大型表，采样	高	近似随机	中
按 ID 范围随机选择	有连续 ID 的表	高	依赖 ID 分布	中