数据库分区设计

王思成

Jul 04, 2026

当单表数据量突破千万甚至亿级后，传统 B+ 树索引的随机 IO 与全表扫描成本会迅速上升，查询延迟与写入锁竞争成为瓶颈。分区通过将物理文件拆分为多个独立段落，使得数据库引擎在执行计划阶段即可利用谓词条件裁剪掉无关文件，从而把 IO 范围从整张表降到若干个分区。分区并非分库分表，它依然运行在同一实例内，共享同一套事务日志与连接池；分库则需要应用层或中间件路由，而分表通常指手工建多张结构相同的表，再由应用拼接查询。理解三者边界有助于在不同场景下做出最小代价的选择。
本文目标是为具备一定 SQL 经验的工程师提供一套可落地的分区设计方法论，涵盖从需求建模、分区键选型、在线改造到日常运维的全流程。读者可直接复用文中 DDL 与脚本，结合自身业务特点快速落地。

1　数据库分区基础

分区是将逻辑表映射到多个物理存储对象的过程，水平分区按行拆分，垂直分区按列拆分。MySQL 的 InnoDB 引擎把每个分区实现为独立的 .ibd 文件，因此可以独立执行 OPTIMIZE、ANALYZE、DROP 等操作；PostgreSQL 的 declarative partitioning 则把分区映射为子表，继承父表约束与索引。列存引擎（如 ClickHouse MergeTree）同样支持分区，但裁剪粒度更细，可达 granule 级别。
常见分区类型包括 Range、List、Hash/Key 及它们的组合。Range 分区按连续区间划分，适合时间或自增 ID；List 分区按离散枚举值划分，适合地域或状态；Hash/Key 分区把分区键做哈希取模，追求数据均匀；Composite 分区先做一级 Range，再在每个 Range 内做二级 Hash，可同时满足时间过滤与写入均衡。
选择分区键时需同时满足单调性、过滤性与低倾斜三条黄金法则。单调性保证新数据只落入最新分区，避免全量重分布；过滤性保证 WHERE 条件能触发 Partition Pruning；低倾斜要求各分区数据量与访问频率差异在 3 倍以内，否则热点分区仍会成为瓶颈。

2　分区策略选型与对比

冷热数据分离是 Range 分区最常见的应用。把最近 90 天数据放在 SSD 热分区，90 天前数据迁移到 HDD 冷分区，可在保持查询性能的同时把存储成本降低 60% 以上。多租户 SaaS 场景常用 List 或 Hash 租户 ID 做分区键，既能隔离租户数据，又能在 schema 变更时只影响单个租户。地理分布式系统则采用 Range 时间 + List 地域的复合分区，让跨地域查询只扫描本地域分区，网络流量下降明显。高并发写入场景下，用 Hash 自增 ID 代替 Range 自增 ID，可把写入热点从单分区扩散到所有分区，TPS 提升 5-8 倍。
各策略在性能、可维护性、扩展性、成本四个维度存在权衡。Range 分区性能最优但扩展性差，新增分区需改 DDL；Hash 分区扩展性好但范围查询无法裁剪；List 分区语义清晰但枚举值变化需重建分区；复合分区兼顾多维度但元数据开销最大。

3 分区设计全流程实操

需求评估阶段需绘制数据增长曲线、统计核心 SQL 的访问模式，并定义 SLA 指标。容量模型需要回答两个问题：单分区上限（通常 100 GB 以内）和总分区数预估（MySQL 建议不超过 1 万）。以日增 5000 万行、保留 3 年数据为例，若单分区上限 50 GB，则需 365×3 ≈ 1095 个分区，接近上限，需考虑按周或按月合并。

分区键与分片算法需在一致性哈希、取模、Range 之间做取舍。一致性哈希引入虚拟节点可降低再平衡数据量，但实现复杂；取模简单但扩缩容需全量迁移；Range 实现最简单但写入热点明显。虚拟节点数量通常取 100-1000，过少会导致数据倾斜，过多则增加元数据压力。

DDL 实施需区分冷表与热表。冷表可直接执行 ALTER TABLE … PARTITION BY；热表需借助 pt-online-schema-change 或 gh-ost 做无锁改造。以 MySQL 8.0 为例，创建按天 Range 分区的订单表可使用如下语句：

```
CREATE TABLE orders (
   id BIGINT UNSIGNED NOT NULL,
   user_id BIGINT UNSIGNED NOT NULL,
   created_at DATETIME NOT NULL,
   PRIMARY KEY (id, created_at)
) PARTITION BY RANGE (TO_DAYS(created_at)) (
   PARTITION p202401 VALUES LESS THAN (TO_DAYS('2024-02-01')),
   PARTITION p202402 VALUES LESS THAN (TO_DAYS('2024-03-01'))
);
```

该语句把 created_at 转换为天数，再按天划分区间。主键必须包含 created_at，否则无法保证唯一性约束在分区内成立。PostgreSQL declarative partitioning 语法如下：

```
CREATE TABLE orders (
   id BIGINT,
   user_id BIGINT,
   created_at TIMESTAMPTZ
) PARTITION BY RANGE (created_at);

CREATE TABLE orders_2024_01 PARTITION OF orders
FOR VALUES FROM ('2024-01-01') TO ('2024-02-01');
```

pg_partman 扩展可自动按月或按周创建未来分区，并把过期分区迁移到归档表，显著降低运维脚本复杂度。索引与约束必须包含分区键，否则全局唯一索引会退化为本地索引，跨分区唯一性无法保证。数据迁移采用影子表 + 双写 + 割接三阶段：先建好分区表，再通过 Canal 或 Debezium 实时同步存量数据，最后把应用读写流量一次性切换到新表。

4 高级话题

分区裁剪依赖执行计划中的 partition pruning 步骤。以 MySQL 为例，EXPLAIN PARTITIONS 可看到实际访问的分区列表，若出现 pMIXED 则表示裁剪失效。ORM 框架在生成 SQL 时若把分区键放在子查询或函数中，裁剪会失效，需在应用层把分区键显式拼接到最外层 WHERE。

自动分区通过定时任务或 pg_cron 实现。脚本需检查未来 7 天分区是否存在，若不存在则提前创建；同时检查 90 天前分区是否已归档，若已归档则执行 DROP PARTITION。跨分区事务在单实例内可直接使用 XA，但在分库场景下需引入 Saga 或 TCC 框架保证最终一致。

扩展性瓶颈出现在单分区写入超过磁盘 IOPS 或网络带宽时，此时需做 Re-sharding。策略包括双写新旧两套分区表、CDC 实时同步、灰度切换流量三步走，期间需监控双写延迟与数据一致性校验结果。

5 真实案例拆解

某电商订单库采用按天 Range 分区 + 冷热分离后，热数据查询延迟从 800 ms 降到 80 ms，整体查询性能提升 10 倍。某社交 Feed 库把 user_id 做 Hash 分区，写入 TPS 从 2000 提升到 15000，单分区大小稳定在 30 GB。某 IoT 时序库采用 Range(time) + List(region) 复合分区，把冷数据迁移到对象存储，存储成本降低 60%。

6 常见陷阱与避坑指南

分区键选择错误会导致全表扫描，典型反例是把创建时间放在二级索引而非分区键。分区数量超过 1 万会使元数据内存膨胀，information_schema.PARTITIONS 查询变慢。唯一索引缺失分区键会引发死锁，因为 InnoDB 需要在所有分区加锁。在线 DDL 对大分区锁表风险极高，需使用 pt-online-schema-change 工具。跨分区 JOIN 与子查询性能骤降，需改写为应用层分片查询或引入冗余列。

7 运维与监控

分区生命周期管理包括自动创建、归档、删除三环节。监控指标需关注单分区大小、裁剪命中率、慢查询在各分区的分布。备份恢复策略可按分区并行执行 mysqldump 或 pg_dump，恢复时间与分区数量成线性关系。

8 未来演进与工具链

云原生数据库 Aurora Limitless、PolarDB-X、CockroachDB 均提供自动分区与自动再平衡能力。HTAP 场景下 TiDB、OceanBase 把分区与 TiFlash 列存结合，可同时满足 TP 与 AP 负载。AI 辅助分区键推荐与自动再平衡已在部分云厂商产品中落地，工程师可通过 API 获取推荐方案。

分区设计 checklist 包括：确认分区键满足单调性、过滤性与低倾斜；单分区大小控制在 100 GB 以内；主键与唯一索引必须包含分区键；建立分区创建/归档/删除的定时任务；监控裁剪命中率与慢查询分区分布。下一步可阅读 MySQL 官方文档、PostgreSQL declarative partitioning 手册及 pt-archiver 源码。

9 附录

MySQL 8.0 分区 DDL 模板见正文示例。PostgreSQL declarative partitioning 示例见正文示例。pt-archiver 自动归档脚本可通过 –source、–dest、–where 参数实现按分区归档。参考文献包括《High Performance MySQL》第四章及 PostgreSQL 官方分区文档。