## 数据库基本概念

逻辑表：水平拆分的数据库(表)的相同逻辑和数据结构表的总称

真实表：分片数据库中真实存在的物理表

数据节点：数据分片的最小单元，由数据源+数据表组成，如ds\_0.t\_order\_0

绑定表：指定分片规则一致的主表和子表，如都按照order\_id分片的t\_order,t\_order\_item表。绑定表之间的多表关联查询不会出现迪卡尔积关联。如果没有绑定，那么如果有两个物理库，三张物理表时，会出现2\*3个组合情况

广播表：表结构和表中的数据在每个数据库中都一致，使用与数据量不大并且需要与海量数据关联查询的场景，如字典表

## 分片基本算法/策略概念

策略代表具体的场景，算法代表该场景下具体的一种计算规则，实现方式，算法目前提供了四种抽象算法

2.1算法

PreciseShardingAlgorithm：精确算法，使用单一键作为分片键的=和in操作的分片场景，与StandardShardingStrategy使用

RangeShardingAlgorithm：范围处理算法，用于单一键作为分片键的between and,>,<,>=,<=的场景，与StandardShardingStrategy使用

ComplexKeysShardingAlgorithm：复杂处理算法，适用于多键作为分片键的场景，配合ComplexShardingStrategy

HintShardingAlgorithm：使用hint行分片的场景，配合HintShardingStrategy使用

从策略的角度来说，分片策略=分片键+分片算法，翻译过来就是，针对不同的分片键的场景，采用不同的算法，他俩组合产生的情景就叫分片策略

**2.2策略**

标准分片策略(StandardShardingStrategy)

适用与分片键有=,>=,<=,>,<,in,between and的场景，支持PreciseShardingAlgorithm,RangeShardingAlgorithm两种算法。前者是比选的，处理=和in的场景，后者是可选的，处理剩余的场景。如果没有配置RangeShardingStrategy,那么between and将按照全库表路由处理

复合分片策略(ComplexShardingStrategy)

适用与多个分片键的=,<,>,<=,>=,in,between and的场景

行表达式分片策略(InlineShardingStrategy)

使用Groovy表达式计算，提供对sql语句中的=和in的分片支持。目前只支持单分片。

Hint分片策略(HintShardingStrategy)

结合HintShardingAlgorithm算法实现，通过hint指定分片值而不是从sql提取分片值的方式进行分片的策略

不分片策略(NoneShardingStrategy)

不分片策略

## 配置基本概念

3.1分片策略配置

数据源分片

对应DatabaseShardingStrategy，配置数据被分配的目标数据源

表分片策略

对应TableShardingStrategy，用于配置数据被分配的目标表，目标表存在于该数据的目标数据源内，所以表分配策略是依赖于数据源分片策略的结果的。

#### 3.2自增主键生成策略

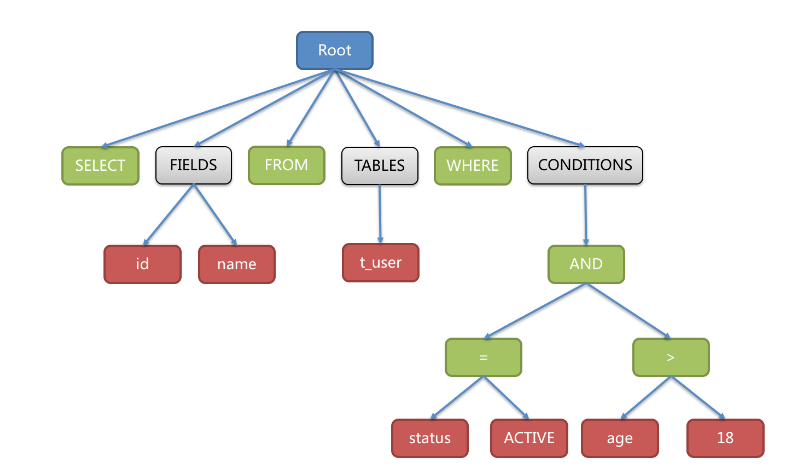
通过在客户端生成自增主键替换以数据库原生自增主键的方式，做到分布式主键无重复

## 执行流程

解析引擎[词法分析，语法分析]->查询优化[执行器优化]->路由引擎->改写引擎->执行引擎->归并引擎

4.1解析引擎

先将sql解析，过程分为词法解析和语法解析。词法解析器将其拆分为原子（称为token），根据不同数据库方言所提供的字典，将其归类为关键字，表达式，字面量和操作符。再使用语法解析器将SQL转换为抽象语法树



语法树中的关键字的token用绿色表示，变量的token用红色表示，灰色的表示需要进一步拆分。

遍历抽象语法树去提炼分片所需的上下文，标记有可能需要改写的位置。

4.2路由引擎

对于携带分片键的sql，根据分片键的不同可以划分为单片路由（分片键的操作符是=），多片路由（分片键的操作符是in）和范围路由（分片键的操作副是between），不携带分片键的采用广播路由

#### 4.2.1 分片路由

直接路由：需要通过hint方式分片，并且是只分库不分表的前提下。可以避免sql解析和之后的结果归并。可以执行包括子查询，自定义函数等复杂情况的任意sql，它也可以用于分片键不在sql中的场景。

标准路由

适用范围是不包含关联查询或者仅包含绑定表之间关联查询的sql。当分片运算符是=时，路由结果落入单库，当分片运算符是between或者是in时，路由结果不一定落入唯一的库。因此一条逻辑sql可能最后会被拆分为多个sql执行于多个库上。

迪卡尔路由

无法根据绑定表的关系定位分片规则，因此非绑定表之间的关联查询需要拆解为迪卡尔积组合执行。

#### 4.2.2 广播路由

不携带分片键的sql，采取广播路由的方式。根据sql的类型可以划分为全库表路由，全库路由，全实例路由，单播路由和阻断路由这五种路由

全库表路由

用于处理对数据库中与其逻辑表相关的所有真实表的操作，主要包括不带分片键的DQL，DML以及DDL

全库路由

处理对数据库的操作，包括用于库设置的set类型的数据库管理命令（设置类的DAL），以及TCL这样的事务控制。这种情况下，会根据逻辑库的名字遍历所有符合名字匹配的真实库。

全实例路由

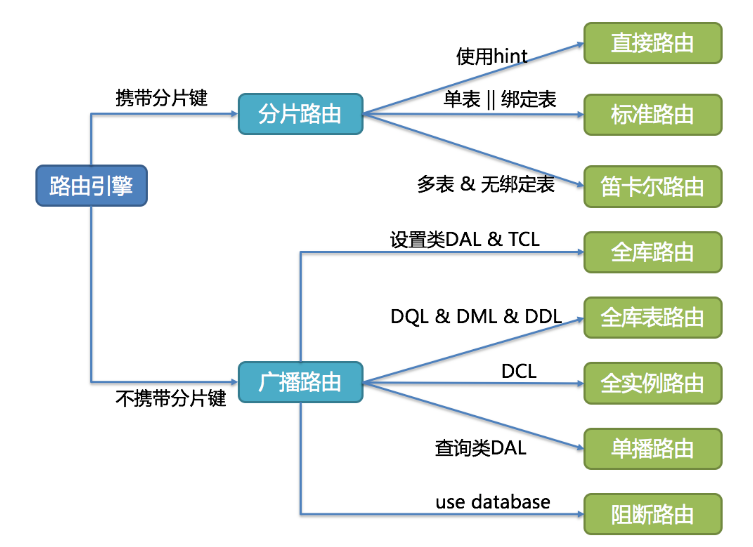
用于DCL操作，授权语句针对的是数据库的实例，无论一个实例中包含多少个schema，每个数据库的实例只执行一次

单播路由

用于获取某一真实表信息的场景，仅需要从任意库中的任意真实表中获取数据即可（查询类的DAL）。也就是随便选一个。

阻断路由

屏蔽SQL对数据库的操作。



4.3改写引擎

将逻辑SQL改写为在真实数据库中可以正确执行的SQL，包括正确性改写和优化改写

#### 4.3.1 正确性改写

包含分表的场景中，需要将分表配置中的逻辑表名称改写为路由之后所获取的真实表名称(t\_order -> t\_order\_1)。如果只是分库的话，那么不需要改写表名。

除此之外，还包括补列和分页信息修正等内容

##### 4.3.1.1标识符改写

包括表名，索引名称以及schema名称

表名改写：找到逻辑表在原始SQL中的位置，将其改写为真实表的过程

索引名称：在部分数据库中（Mysql，SQLServer），索引是以表为维度创建的，在不同的表中的索引是可以重名的;但是在另外一些数据库中（Oracle，PostgreSQL）中，索引是以数据库为维度创建的，即使是作用在不同的表上的索引，它们的名字也要求具有唯一性。

Schema名称：管理方式和管理表一样，采用逻辑schema去管理一组数据源，将用户在sql中书写的逻辑schema替换为真实的schema。目前不支持在DQL和DML语句中使用schema，仅支持在数据库管理语句中使用Schema。这部分的改写是指将逻辑schema采用单播路由的方式，随便找到一个符合条件的真实schema并执行。

数据库schema：一种是概念上的schema，只得是一组DDL语句集，完整的描述了数据库结构;还有一种是物理上的schema，指的是数据库中的一个名字空间，包含一组表，视图和存储过程等命名对象。物理Schema可以通过sql语句来创建，更新和修改。它就是一个数据库用户所拥有的数据库的对象。

##### 4.3.1.2补列

补列智慧补充缺失的列，不会全部补充

场景一：情况是shardingspher需要在结果归并时获取相应的数据，但是sql并没有返回。主要针对group by和order by。结果归并时，根据group by和order by的字段项进行分组和排序。但是如果原始sql的选择项中没有包含分组项或者排序项[就是说group by或者order by的字段没有出现在select中]，需要对原始sql进行改写。

场景二：使用AVG聚合函数，分布式的场景下，（avg1+avg2+avg3）/3计算平均值是不对的，需要改写为（sum1+sum2+sum3）/（count1+count2+count3），这就需要将sql从avg(c)改为sum(c)/count(c)

场景三：执行insert的sql语句，使用数据库的自增主键，写sql是一般是insert into values ,into后面的列名一般不包含主键，改写后将其补上，调用主键生成策略生成id

##### 4.3.1.3分页修正

从多个数据库获取分页与单个数据库的场景是不同的。不能将源sql直接放到数据库中执行。越获取偏移量位置靠后的数据，使用limit的方式效率越低，需要避免这种方式。

##### **4.3.1.4批量拆分**

使用批量插入的sql时，如果插入的数据是跨分片的，需要对sql改写防止将多余的数据写入到数据库中。插入与查询的不同在于，查询中即使使用了不存在于当前分片分片键字段，也不会对数据产生影响（就是查不到），但是插入则需要将多余的分片键删除。

#### 4.3.2优化改写

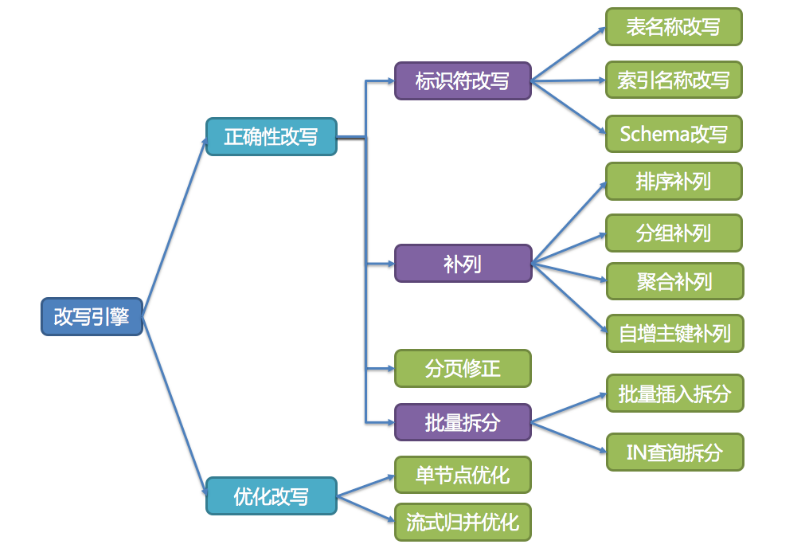
在不影响正确性的情况下，对性能进行提升的手段。分为单节点优化和流试归并优化。

**单节点优化**

路由至单节点的sql，不需要优化改写，当获得一次查询的路由结果之后，如果是路由到唯一的数据节点，那么不需要设计到结果归并。因此上一章节说的补列和分页信息等改写都没必要进行。

**流式归并优化**

仅为包含group by的sql增加order by以及和分组项相同的排序项和排序顺序[意思就是将group by中的字段逐个order by一下，并且指定这些字段的排序顺序是先序还是倒序]。用于将内存归并转化为流式归并。



4.4执行引擎

平衡数据源链接创建以及内存占用所产生的消耗，最大限度地合理利用并发等。目标是自动化的平衡资源控制和执行效率。

可能场景：在一个数据库实例中存在较多分表的情况下，一条不包含分片键的逻辑sql将产生在同库不同表的大量真实sql，每条真实sql都占用一个独立链接的话，对于一次查询将会占用过多的资源。

为每个分片维持一个独立的数据库链接，可以更加有效的利用多线程来执行效率。为每个数据库开启独立的线程，可以将IO所产生的消耗并行处理。同时为每个分片维持一个独立的数据库链接，可以避免过早的将查询结果数据加载到内存，此时只是持有游标位置的引用，在需要获取相应的数据时，移动游标即可。

所以数据库和分片的关系一般是1:n,n>=1,所以对应过来线程数和链接数的关系也是1:n,n>=1

**流式归并**

以结果集有表下移进行结果归并的方式，无需将结果数据全加载进内存，可以有效的节省内存资源，减少垃圾回收的频次

针对一条SQL经过shardingsphere计算之后拆分为200个分表的查询的情况，为了平衡效率和链接可靠的关系，提出了链接模式的概念

**内存限制模式（一般使用OLAP）**

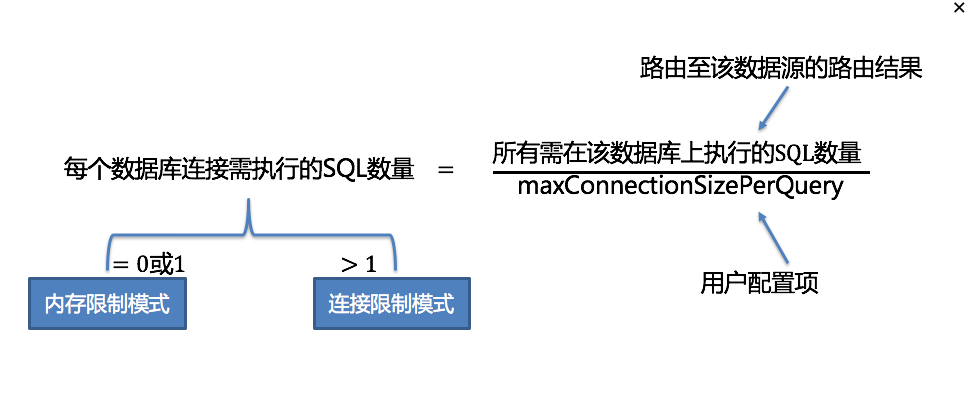
前提是对一次操作所耗费的数据库链接数量不做限制，也就是链接可以不停增加。举个例子就是，执行的sql需要对200张表操作，那么对每张表创建一个新的数据库链接，并且通过多线程方式并发处理，达成执行效率最大化。此时更加可以结合流式归并，减少内存的使用以及频繁的垃圾回收。

**链接限制模式（一般使用OLTP）**

严格控制对一次操作所耗费的数据库链接数量。如果实际执行的sql需要对某个数据库实例中的200张表操作，那么只会创建唯一的数据库链接，对200张表串行处理。

如果此次操作的分片散落在不同的数据库中，仍然采用多线程处理对不同库的操作，但是每个库的每次操作仍然只创建一个唯一的数据库链接。可以防止一次请求对数据库链接占用过多带来的问题。

上面两种模式在新版本的shardingsphere中，被内聚到框架内部，变成根据每一次sql动态的判断该采取那种方式来高效率的执行。通过maxConnectionSizePerQuery参数可以在执行时，一次查询时每个数据库所允许使用的最大连接数

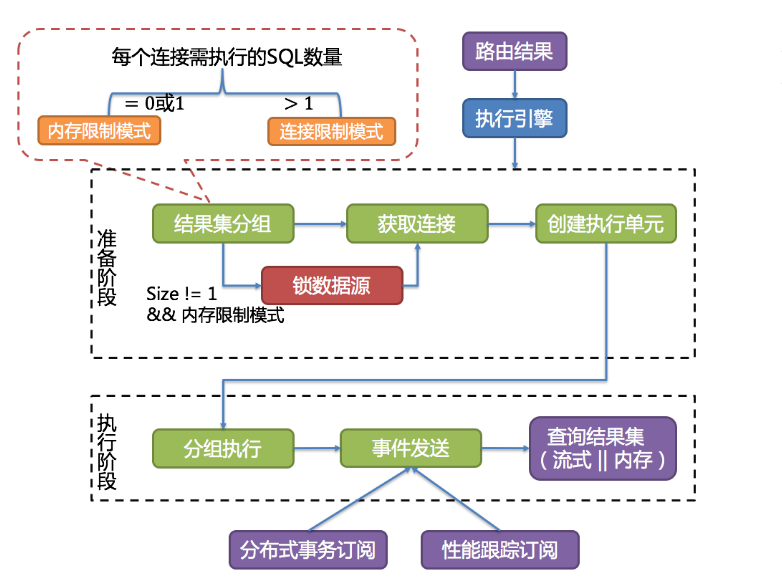


这张图的意思就是所有的连接数\*每个连接数能查询的sql数量=一次查询需要执行的sql的数量。

换算成除法就是总sql数量/总的连接数=每个连接需要处理的sql数量

如果每个链接需要处理的sql>1，那就是说超过原先设定的负荷了，这个时候就需要使用连接限制模式进行严格的控制，防止连接数耗尽

Shardingsphere通过在执行准备阶段的获取的链接模式，生成内存归并结果及或者流式贵并结果集，将其传递至结果归并引擎，以进行下一步的操作。



4.5归并引擎

目的是将从各个数据节点获取的多数据结果集，组合成为一个结果集并正确的返回至请求客户端。

功能上分为遍历，排序，分组，分页和聚合五种类型，他们之间是组合的关系，而非互斥。

从结构划分，可以分为流式归并，内存归并和装饰者归并。其中流式归并和内存归并是互斥的，装饰者归并可以在他们俩上面做进一步的功能增强处理。

由于结果集是逐条返回，并不需要将所有的数据都加载到内存中，在归并时，这样可以极大减少内存的消耗。

**流式归并：**每次从结果集中获取到的数据，都能够通过逐条的方式返回正确的单条数据，它与数据库原生的返回结果集的方式最契合。遍历，排序以及流式分组都数据流失归并的一种。

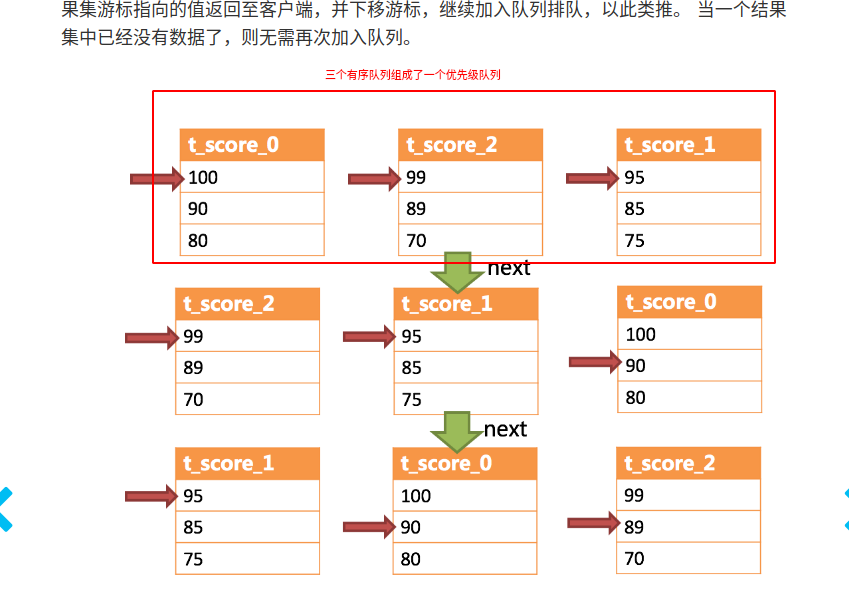
**内存归并：**需要将结果集中的所有数据都遍历并且存在内存中，通过统一的分组，排序以及聚合等计算之后，将其封装成为逐条访问的结果集返回

#### **4.5.1遍历归并**

将多个数据结果集合并为一个单向链表，遍历完成链表中当前数据结果集之后，将链表元素后移一位，继续遍历下一个

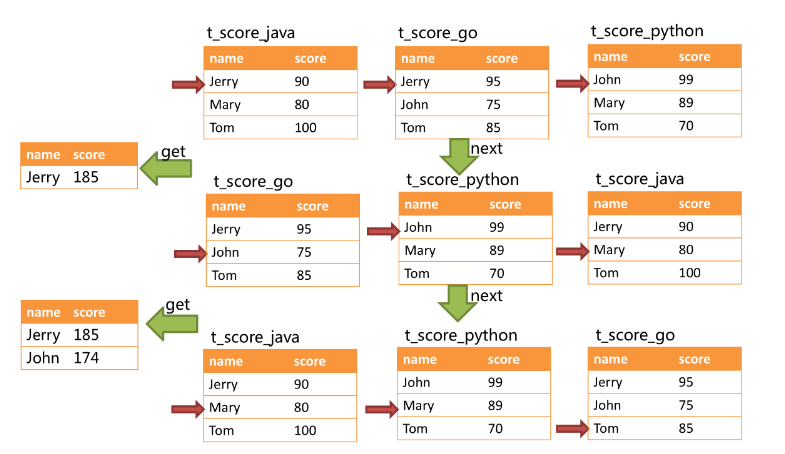
#### **4.5.2排序归并**

对结果集进行排序。由于sql本身就存在order by语句，每个结果集自身是有序的，只需要将结果集当前游标只想的数据值进行排序即可，相当与对多个有序的数组进行排序，归并排序是最适合当前场景的算法。



#### **4.5.3分组归并**

分为流式分组归并和内存分组归并，流式分组归并要求sql的排序项(order by指定)和分组项的字段(group by指定)以及排序类型必须保持一致，否则只能通过内存归并



流式归并的图如上，它和排序归并最大的区别是它会一次性将多个数据结果集中的分组项相同的数据全部取出，它需要根据聚合函数的类型进行聚合计算

#### 4.5.4聚合归并

除了分组的sql之外，不进行分组的sql也可以使用聚合函数，所以它的实现是采用了装饰者模式。它的使用可以分类为比较（max，min），累加（sum，count），求平均值（sum/count）三种类型。

#### 4.5.5分页归并

也是增强的模式，使用了装饰器模式，在流式归并和内存归并的基础上进行了增强

