DECIMAL功能设计文档

修订历史

版本	日期	修改描述	作者	备 注
Cedar 0.3	2017-9-28	DECIMAL 功能设计文档	徐石磊	无

1 需求分析

Cedar是华东师范大学计算机科学与软件工程学院基于OceanBase数据库研发的可扩展的关系数据库，实现了巨大数据量上的跨行跨表事务。在cedar 0.2版本中缺乏对DECIMAL数据类型的支持，为满足业务功能上的需求，在cedar 0.3版本中增加DECIMAL数据类型，使得数据库能够实现存储，读取decimal类型的数据，并能进行相关的数学运算。

2 设计要求

DB2中Decimal数据类型：DB2中Decimal类型的最大支持精度为31位十进制数，当输入数据的精度大于最大支持精度时，直接按精度截取。
Oracle中的Decimal类型：Oracle中Decimal类型的最大支持精度是38位十进制数，当输入数据的精度大于最大支持精度时，四舍五入。
为了最大化实现与DB2兼容，我们设计的Decimal数据类型与DB2中的Decimal类型一致。主要体现在Decimal数据类型在结果显示，运算以及表达范围的各方面要求。

- **Decimal的显示规则**
  能够表示的十进制位数最大为38位。采用与DB2中Decimal类型相同的设置方式，即使用两个参数来确定decimal的位数和小数部分的精度，如decimal(p,s);其中p表示位数precision，s表示小数部分的精度scale。

> 示例：
> decimal(3, 2) 6.789 -> 6.78 超过小数部分限制，需进行截取
> decimal(4, 3) 3.14 -> 3.140 小数位数不足，进行补0操作

注：本设计中要求scale的长度小于precision，即scale < precision。scale作为小数部分位数的限制。s大于0的时候，超过s位的后面的小数将会被截掉，而不是四舍五入。如果用户输入的数的scale小于定义时候的scale，将会进行补0操作。

3 功能简述

3.1 SQL基本功能

- **建表：**能够创建包含有Decimal类型的数据表。建表语句以SQL方式输入，要求该语句语能够被Cedar识别接受，并成功建立一张包含一列或者多列Decimal类型的数据表。

> 示例SQL：
> create table table_name(c1 int primary key, c2 decimal(5, 2));

- **查看表结构：**能够用描述表结构的SQL语句，查看数据表结构的时候，并正确返回含有Decimal数据列的信息，查询的结果包括DECIMAL数据列的精度大小。

> 示例SQL：
> desc table_name;

- **插入数据：**在包含有Decimal类型的表中插入数据：能够使用SQL的插入insert语句，往一张已存在的数据表中插入包含Decimal类型的数据，要求能够通过词法语法解析，执行逻辑/物理执行计划后实现数据的写入。

> 示例SQL：
> insert into table_name values （0，3.14）；

注：相类似的SQL语句，如repalce同样能够成功运行。

- **更新数据：**更新包含有Decimal数据类型的行数据：能够使用SQL的update语句，将表中已经存在的，且包含有Decimal类型的行数据修改，要求能够通过词法语法解析，执行逻辑/物理记性计划后实现数据的修改。

> 示例SQL：
> Update table_name set c2 = 3.15 where c1 = 0;

- **查询数据：**查询数据表时，对于包含Decimal的数据列，返回数值、格式均正确的数据。

> 示例SQL：
> select * from table_name;

注：因为Decimal数据类型本身的特殊性，对于显示结果格式有特别需求，将在下一节进行描述。

- **计算与类型转化：**能够实现基本的数学运算，以及与其他类型，诸如整型，浮点数，double类型或者varchar类型的相互转换，这些计算与转换能够在SQL语句中得以实现。

> 示例SQL：
> select c2 + 3.14 from table_name;

注：在SQL语句中实现了如SUM，AVG等查询函数。

4 设计思路

- **Schema设计**
  对于Cedar 0.2来说，每个server都有自己的ObSchemaManagerV2类。在ObSchemaManagerV2类里存储并管理所有表的schema，包括每张表有多少列，每一列是什么数据类型，并且保存所有server上。 ObSchemaManagerV2所管理的表schema是一致的。但是，由于CEDAR 0.2不支持decimal，所以在实现ObSchemaManagerV2的时候，没有设计数据结构来存储用户定义的精度和有效数字。所以，为了实现decimal，必须对ObSchemaManagerV2进行修改。所以，我们要在在ObColumnSchemaV2里面增加了一个数据结构：

```
struct ObDecimalHelper
{
uint32_t dec_precision_ :7;
uint32_t dec_scale_ :6;
};
```

注：在修改了ObSchemaManagerV2之后，要对整个集群进行boot_strap（删除集群原有数据）之后才能重启，否则会报错。

- **Decimal类型数据的存储设计：**
  Decimal类型数据的存储与定长类型（如，int，double等），或者变长类型（如varchar）类型均不相同。因为Decimal类型的存储将影响到OB的cs中内存、sstable和ups中memtable中的数据存储，以及cs，ms，ups中间处理结果的存储（如数据修改，新数据生成等），以及相应的序列化，反序列化。OB的数据存储（如表连接、投影等）与传统数据库中的处理不同（将定长、变长数据分别存储在两个地方），大幅增加了内存管理的复杂度。 由于DECIMAL类型需要的存储空间较大并且应用变长存储方式，所以参考varchar存储方式设计。

- **四则运算设计**
  为了简化数据处理编码的复杂度，并提升处理性能，项目组决定采用开源、成熟的大整数处理类库，TTMath。TTMath对big unsigned integer, big signed integer and big floating point提供算数运算操作（如，加减乘除等）。TTMath 在BSD license下开源，可满足个人以及商业用途。TTMath最大的特点是使用了堆栈内存，以提升内存分配、释放性能，进而提升数据处理的性能。
  在进行加，减，乘，除计算的时候，直接取出TTInt类型的word进行四则运算，因为总长度是对齐的，所以只需要在最后的结果根据两个数的scale，precision进行结果处理即可。而对于乘法和除法，直接做运算的时候会造成结果溢出，所以用typedef ttmath::Int<8> TTLInt，用于乘除计算。因为这个类型长度四倍于TTInt，所以不会造成溢出，计算时，先将两个乘数转化成TTLInt，再进行乘除计算，结果与MaxDecimalValue/MinDecimalValue，做溢出处理。
  CEDAR在存储Decimal时进行数值检测，运算是在存入的数值上进行，因此运算时不需要再进行数值检测，只需要检测是否溢出。

注：TTMath类库访问地址 http://www.ttmath.org/

- **数据合并和数据恢复设计**
  数据合并：将chunkserver和updateserver的数据进行merge合并之后，数据能够成功写入chunkserver 的sstable当中，写入的数据要求精确无误，并能够完整正确的被读取出来。
  数据恢复：有时OceanBase会由于种种原因需要重启，此时updateserver的数据还未冻结或进行merge操作，重启后会根据commit_Log中的内容将数据还原到内存表的sstable当中，添加的Decimal类型数据需要能够实现此功能，还原的数据需要正确无误。