Design and Implementation of A Cross-Database Query Tool on Multi-Source Databases

Liang Liang[1], Zhu Guangxin[2], Li Yuqian[2]*, Hu Junjun[2]

[1] State Grid Jiangxi Electric Power Company Information and Telecommunication Branch, Nanchang, Jiangxi, China.
[2] NARI Group Corporation, Nanjing, Jiangsu, China.

* Corresponding author. Tel.: 13260923932; email: liyuqian@sgepri.sgcc.com.cn
Manuscript submitted May 5, 2018; accepted July 20, 2018.
doi: 10.17706/jcp.13.9.1042-1052

Abstract: With the rapid development of the company's business, and the demand for inter-business type data correlation inquiry, an easy and efficient cross-database query tool is urgently needed. Currently, the cross-database query method has the following problems: firstly, it can not meet the needs of multiple data sources; second, there are complicated operations and waste of resources; third, full real-time data can not be guaranteed. In order to solve these problems, this paper designs a cross-database query tool with multiple data sources, which is realized by adapters, distributed parallel collaborative query and heterogeneous data association. The tool can insert a variety of data sources through plug-in, shield different data source differences, use the unified SQL accessing to query relationship，and use the memory to ensure query efficiency and real-time。It does not store data itself, only as data processing Channel to reduce the waste of storage resources.

Key words: Cross-database query tool, multi-source databases, collaborative inquiry, related query.

1. Introduction

Cross-database query, as the name, refers to cross query across different data sources technology. Existing databases, such as MySQL [1], [2], Oracle [3], SQL Server [4], etc., can use SQL statements to query across different databases of the same type, but for different types of databases, it has not succinct SQL statements to achieve cross-database-related queries [5], [6]. For the related queries of different types of data sources [7]-[11], the current method is to extract the data from one data source to another data source, and due to the differences between different data sources, such as storage format, data type and so on. The extraction process also need to convert these data, the process is usually ETL [12], [13] tools to achieve. Therefore, the current query across different types of data sources has the problem of full or large amount of data being copied, resulting in a waste of a large amount of resources. In addition, data synchronization can not be guaranteed by means of ETL tools.

This paper designs and implements a cross-database query tool based on multiple data sources. Without the aid of ETL tools, it can realize the related queries of many different types of data sources. Through the preparation of simple SQL statements can achieve cross-database-related queries, the specific access process is as follows: the client input SQL statement, cross-database query tool to parse the SQL statement to generate the execution plan, according to the range of data to be processed from the access data source to extract data to the inter-database query tool for correlation, merging, aggregation and other operations,

the calculation results are returned to the requesting client. The cross-database query tool designed in this paper can make the user operation simple and quick, shield the operational differences of different data sources at the bottom, and decouple the application from the data source.

The structure of this paper is as follows: Sect.1 briefly introduces the component modules of cross-database query tools, Sect.2 introduces the detailed design and implementation of cross-database query tools, including the data processing flow, data objects, relations and interface call relationships and key technologies. Sect.3 presents a cross-database query tool for a typical application in grid big data. Sect.4 concludes the full text.

2. Module Structure

Cross-database query tools mainly provide data query service for the data and data model of the data source in the system. The query mode is SQL, and it mainly contains four modules: data source plug-in module, data service module, system service module and service interface module. The design of each module is shown in Fig. 1.

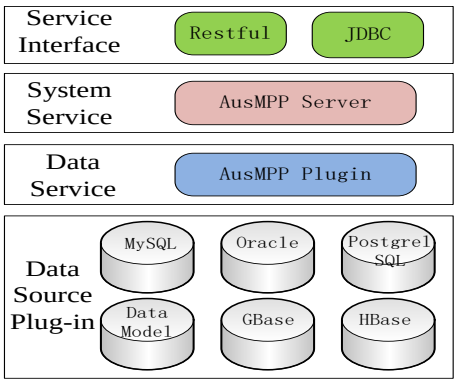

Fig. 1. Cross-database query tool module design.

Data source plug-in module provides the connection adapter of the underlying data source, that currently includes MySQL, Oracle, PostgrelSQL, GBase, HBase database access, to add other database connection adapter, you need to add other databases through the data model adapter.

Data services module provides a cross-database query processing plug-ins AusMPP, through the plug-in can connect to the underlying different data sources.

System service module is the core module of cross-database query tool. AusMppServer is the core service of cross-database query. The service contains one primary node and one to many child nodes. The primary node is used to query the task dispatching and distribution, and provides the JDBC service interface , the child nodes handle the specific distribution of the query task.

Service interface module provides JDBC and Restful two call interface, support for standard SQL syntax.

3. Design and Implementation

3.1. Data Processing Flow

Cross-database query using distributed mode for data processing, each child node of AusMppServer cluster through the AusMppPlugin plug-in access to various types of data sources. Which AusMppServer data processing flow shown in Fig. 2.

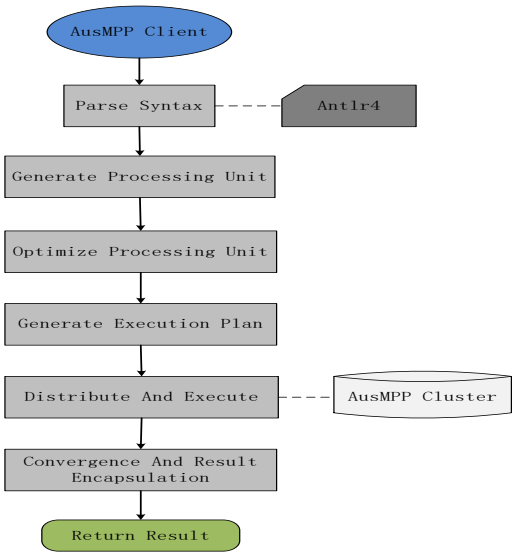

Fig. 2. AusMppServer service implementation process.

After receiving the request from the client, AusMppServer parses the syntax and the lexical code, and generates the processing unit. After that, AusMppServer optimizes the processing unit. At the same time, the operation may include operations such as data-based authority verification and joint table processing. And then generate an optimized execution plan and distribution to the sub-node of AusMpp cluster to execute the query, the result will eventually be gathered and packaged as AusMpp ResultSet result set and return.

3.2. Data Objects, Relationships and Interface Calls

This section will detail each process and sub-module data objects, relationships and interface calls.

3.2.1. Client request

When a JDBC QueryRequest is initiated, the JDBC driver encapsulates the query and client information, including parameters, session information, etc., and submits it to the master node of AusMpp. QueryRequest data structure design in Table 1.

Table 1. QueryRequest Data Structure

Field Name	Data Type	Description
server	String	The main node HttpServer address (transient)
body	String	Query statement
user	String	Username
catalog	String	Default data source name
schema	String	Default database name
timeZone	TimeZone	Session time zone
userAddress	String	Client address
userAgent	String	Client Agent Information
startTime	long	Session time
systemProperties	Map<String, String>	Common configuration properties
connectorProperties	Map<String, Map<String, String>>	Configuration attributes for each data source
prepareStatement	Map<String, String>	JDBC prepareStatement information

After AusMpp receives the query, the master node submits the encapsulated structure to the query executor through REST. The query executor maintains the life cycle of the request, and finally encapsulates and returns the QueryResult to the client. The design of the QueryResult data structure is shown in Table 2.

Table 2. QueryResult Data Structure

Field Name	Data Type	Description
id	String	Job ID
uri	URI	Query information address
columns	List<Column>	Username
data	Iterable<List<Object>>	Default data source name
stats	StatementStats	Statement status information
error	QueryError	Query error
updateType	String	Whether to support the update, such as UPDATE, QUERY
updateCount	long	Update record number

3.2.2. Syntax parsing

Submit it to the parsing layer for validation, and build the syntax tree. Syntax parsing uses Antlr4 technology to define the syntax in AusMpp by building g4 files. g4 file syntax design.

Statement Clause

```
statment
    : query
 | EXPLAIN ANALYZE?
        ('(' explainOption (',' explainOption)* ')')? statement
    | SHOW TABLES ((FROM | IN) qualifiedName)? (LIKE pattern=STRING)?
    | SHOW SCHEMAS ((FROM | IN) identifier)? (LIKE pattern=STRING)?
    | SHOW CATALOGS (LIKE pattern=STRING)?
    | SHOW COLUMNS (FROM | IN) qualifiedName
    | DESCRIBE qualifiedName
    | DESC qualifiedName
    | SHOW FUNCTIONS
    | DIRECT rawQuery
;
```

SELECT Clause

```
query
    : SELECT setQuantifier? selectItem (',' selectItem)*
      (FROM relation (',' relation)*)?
      (WHERE where=booleanExpression)?
      (GROUP BY groupBy)?
      (HAVING having=booleanExpression)?
;
```

FROM Clause

```
relation
: left=relation
      (joinType JOIN rightRelation=relation joinCriteria)
      | aliasedRelation
;
aliasedRelation
      : relationPrimary (AS? identifier columnAliases?)?
;
relationPrimary
      : qualifiedName | '(' query ')'
      | '(' relation ')'
;
columnAliases
      : '(' identifier (',' identifier)* ')'
;
```

WHERE Clause

```
booleanExpression
      : predicated
      | NOT booleanExpression
      | left=booleanExpression operator=AND right=booleanExpression
      | left=booleanExpression operator=OR right=booleanExpression
;
```

GROUP BY Clause

```
groupBy
      : setQuantifier? groupingExpressions (',' groupingExpressions)*
;
setQuantifier
      : DISTINCT
      | ALL
;
```

Compile g4 files and generate the required SQL parser (SqlParser). SqlParser is responsible for receiving execution statements and building them into syntax trees.

3.2.3. Execution plan

Constructs a processing unit (Map <PlanId, PlanNode>) of the execution syntax for the generated syntax tree by using the visitor design pattern and constructs an execution plan topology with the following structure:

PlanRoot -> SubPlan

SubPlan -> List<PlanId>

By PlanId corresponding PlanNode, PlanNode mainly includes: TableScanNode, SortNode, JoinNode, FilterNode, GroupNode, AggregationNode, ExplainNode. TableScanNode data structure shown in Table 3,

data structure of SortNode, JoinNode, FilterNode, GroupNode, AggregationNode, ExplainNode is the same.

Table 3. TableScanNode Data Structure

Field Name	Data Type	Description
id	PlanId	Execution plan ID
table	TableHandle	Table handle
outputColumns	List<Symbol>	Return column
columns	Map<Symbol,ColumnHandle>	Column handle
filter	Expression	Expression of the original constraint
constraint	TupleDomain<ColumnHandle>	Constraint

3.2.4. Processing unit

The generated processing unit is optimized to be dispatched to each TaskExecutor. The TaskExecutor maintains a separate thread pool and executes the task slices. The slice is the minimum processing unit, by breaking the task into multiple slice, to achieve distributed job processing. Slice actuators really responsible for the implementation of the query logic. The implementation of slice data structure design in Table 4.

Table 4. Data Structure of the Implementation of Slice

Field Name	Data Type	Description
splitId	String	Slice ID
taskHandle	TaskHandler	Task handle
driverExecutor	DriverExecutor	Job execution operation driven
pipelineContext	PipelineContext	Conversation context
stats	Feature	Slice execution status

Among them, driverExecutor is actuator of AusMpp. AusMpp driverExecutor Maintain an AusMppOperator. AusMppOperator is the core of AusMpp, mainly includes:

AggregationOperator: to handle data aggregation operations.

DistinctOperator: distinct operation to maintain the processing logic.

ExplainOperator: processing logic to maintain execution plan operations.

FilterOperator: to handle data filtering operations.

GroupOperator: for handling data packet operations.

OrderByOperator: used to process data sorting operations.

TableScanOperator: the operation of browsing data.

The result of AusMppOperator is returned as Page, Page contains the page Block, data structure of Block is the same with the common database.

Operator interface design is as follows:

1. getOperatorContext

Interface	OperatorContext getOperatorContext()
Description	Get the operation context information
Parameter	none
Return	Operation context: contains operatorId, PlanNode information, operation type, actuator, statistical counter, etc.

2. getTypes

Interface	List<Type> getTypes();
Description	Get the data type corresponding to the operation

Parameter	none
Return	Operation Data Type (nullable)

3. finish

Interface	void finish();
Description	Set operation is completed (including statistical counter records)
Parameter	none
Return	none

4. isFinished

Interface	boolean isFinished();
Description	Get the implementation of the operation
Parameter	none
Return	Whether the operation is completed

5. isBlocked

Interface	default ListenableFuture<?> isBlocked()
Description	Get the blocking ID
Parameter	none
Return	The blocking listener default returns. Get the blocking identifier:NOT_BLOCKED

6. needsInput

Interface	boolean needsInput();
Description	Get whether to receive new page information
Parameter	none
Return	Can receive a new page

7. addInput

Interface	void addInput(Page page);
Description	Add operation input source
Parameter	page (Operation input page)
Return	none

8. getOutput

Interface	Page getOutput();
Description	Get operation output page
Parameter	none
Return	Operation output page

9. close

Interface	void close()
Description	Close the operation handle and the corresponding connection
Parameter	none
Return	none

3.2.5. Pipeline

Build an optimized execution plan into a pipeline / topology. Query actuator data structure design in Table 5.

Table 5. Data Structure of Query Executor

Field Name	Data Type	Description
queryId	String	QueryID
query	String	Query statement
metadata	Metadata	Query Metadata
queryState	String	Query status, mainly including: QUEUED,PLANNING,STARTING,RUNNING,FINISHED, FAILED
session	Session	Session information
inputs	Set<Input>	Request source
output	Optional<Output>	Request output (empty when client request type is not UPDATA)
resource	Resource	System resource
scheduler	NodeScheduler	Scheduler

3.2.6. Execute the plug-in

The execution plug-in is implemented through the customized AusMppJdbc plug-in. It integrates the catalogs, schemas, and data types of various data sources, including MySQLJdbcPlugin, PostgreSQLJdbcPlugin, GBaseJdbcPlugin, PhoenixJdbcPlugin, OracleJdbcPlugin, DataModelPlugin,among them, DataModelPlugin is for modeling models.

JdbcClient plug-in interface is as follows:

1. getSchemaNames

Interface	Set<String> getSchemaNames();
Description	Get database/schemas name
Parameter	none
Return	Database/schemas name collection

2. getTableHandle

Interface	JdbcTableHandle getTableHandle (SchemaTableName schemaTableName);
Description	Get JDBC table handle
Parameter	schemaTableName (table name and schemas name)
Return	JDBC table handle: Contains table name, table connection information and catalog, schema and other table information

3. getColumns

Interface	List<JdbcColumnHandle> getColumns(JdbcTableHandle tableHandle);
Description	Get JDBC column handle
Parameter	tableHandle (table handle)
Return	JDBC column handle list：Each column corresponds to an element

4. getSplits

Interface	ConnectorSplitSource getSplits (JdbcTableLayoutHandle layoutHandle);
Description	Set slices and package to slice execution connector
Parameter	layoutHandle (Table information and column information)
Return	Slice execution connector

5. getConnection

Interface	Connection getConnection(JdbcSplit split) throws SQLException;
Description	Get JDBC connection of the data source
Parameter	split (Task slices, including table information, connection information)
Return	JDBC connection of the data source

6. buildSql

Interface	PreparedStatement buildSql(Connection connection, JdbcSplit split, List<JdbcColumnHandle> columnHandles) throws SQLException;
Description	Build sql execution PreparedStatement
Parameter	connection (JDBC connection built by getConnection) Split(Task slices, including table information, connection information) columnHandles(Column handle)
Return	PreparedStatement of the data source

3.3. key Technologies

The key technologies of Cross-database query tools used is as follows:

Distributed parallel collaborative query technology: AUSMPP query engine through the query to parse, optimize, and ultimately generate multiple executable query slices and build the pipeline for the implementation of the scheduled tasks, through the scheduler in the query nodes of the cluster collaborative execution in parallel.

Heterogeneous data association technology: through various data sources of different data structures such as data integration, slices aggregation, association and other operations, it supports all kinds of heterogeneous data association.

4. Tool Application

With the deepening of the construction and application of SGCC's business lines and information systems as well as the goal of accelerating the establishment of a global Internet and the completion of a "one strong, three gifted and one" modern company, we have put forward higher goals for all business collaboration and the entire process requirements, dig deeper into the value of data, data management business, the demand for information-driven business more urgent. At present, multi-source business data is stored and multi-application scenarios involve inter-database cross-query. Therefore, the inter-database cross-database service problem that is easy to use and fast and highly efficient is solved in this paper.

Taking the cross-database query scenario of data warehouse GBase and big data platform HBase as an example, this paper describes the specific use of cross-database query tools in this paper. Cross-database query business application execution process shown in Fig. 3.

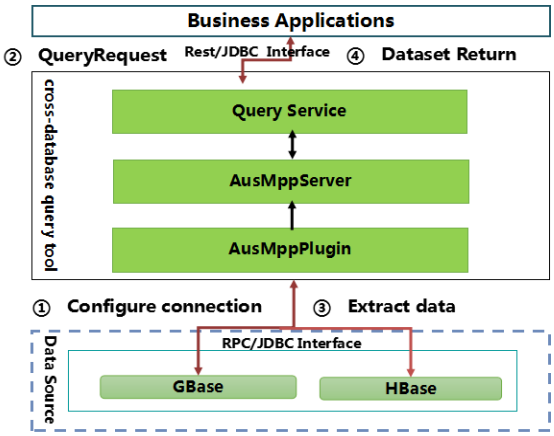

Fig. 3. Cross-database query execution process.

The following is a detailed flow of business applications accessing the underlying data source by cross-database query tools:

1. Configure GBase and HBase access connection in data cross-database query tools by AusMPPPlugin.

2. Business applications through the JDBC or Restful interface call cross-database query tools to access data from GBase and HBase.

3. AusMPPServer of Cross-database query tool parses the request, construct a parallel execution plan, obtain data in GBase and HBase ,and perform correlation and consolidation in AusMPPServer.

4. AusMPPServer returns the data query result to the Query Service, and returns the query result directly to the requester through the Restful / JDBC interface.

5. Conclusion

In order to solve the problem that complex operation, waste of resources and low real-time performance of multi-data source cross-database query, this paper designed and implemented a simple operation, real-time and efficient cross-database query tool. By writing simple SQL statements, users can query cross-database related queries, shield users from different data source operation differences, and decouple applications and data sources. This solves the problem of multiple data sources cross-database query.

Acknowledgment

Supported by the Project of Unified Analysis Services Component - Design Development Implementation of State Grid Corporation. Thanks for the support of State Grid Jiangxi Corporation for our work and for the technical support provided by Nanjing NARI Group Corporation.

References

[1] Zu Hong, F. (2002). A method of acessing mysql-based distributed database. *Journal of Computer Applications, (08),* 4-6.

[2] Shaoyang, Q. (2016). Submitted in partial fulfillment of the requirements. Master dissertation, Nanjing University.

[3] Yong, Y., Guangling, Y., & Xiaolin, S. (2017). Analysis of oracle RAC cluster performance optimization strategy. *Wireless Interconnected Technology, 2017(22),* 108-109.

[4] Tao, Z., Jinwei, G., Huan, Z., Heng, Z., & Aoying, Z. Consistence and Availability in distributed database system. *Journal of Software,* 1-19.

[5] Zhenyu, S., & Huajun, W. (2013). The solution of Sql server and Mysql database's unified management. *Computer Knowledge and Technology, 9(23)*, 5203-5205+5210.

[6] Jiangtao, L. (2012). Access to oracle database under SQL server. *Computer Knowledge and Technology, 8(27),* 6437-6440.

[7] Ankush M. (2016). Gupta, vijay gadepally, michael stonebraker: Cross-engine query execution in federated database systems. *HPEC*, 1-6.

[8] Daniel R., & Harris, D. W. (2017). Henderson, jeffery C. talbert: Using closure tables to enable cross-querying of ontologies in database-driven applications. *BHI 2017,* 493-496.

[9] Souyu, L. (2003). Project of general query heterogeneous database cross plantform based on CORBA. *Computer Engineering, 2003(11),* 82-84+110.

[10] Wenliang, Y. (2011). The development and application of information system based on distributed cross database platform mode. *Electronic Commerce, 2011(04),* 56-57.

[11] Xiaolong, L., & Jing Xin, T. (2011). Research and implementation of cross database persistence layer framework. *Computer Engineering and Design, 32(11),* 3729-3733.

[12] Jie, S. (2008). Research on key techniques of data warehousing and ETL for multi-type data source *Northeastern University,* 2008.

[13] Shaosheng, J., Zuping, Z., & Songqiao, C. (2003). Modeling tools of data and their implementation on DSS. *Computer Engineering and Applications, 2003(10),* 202-205.

Liang Liang was born in 1984, Jin Xian Jiangxi China. In 2010, he graduated from the University of Magdeburg, Germany Computer Institute, studying data knowledge engineering. Since 2010, he works in state grid Jiangxi electric power company information and telecommunication branch. His main research interests include power enterprise informatization, power big data analysis, information engineering management, etc.

Guangxin Zhu was born in 1981, Jixian Hebei China. In 2007, he graduated from Nanjing University of Science and Technology, School of Computer Nanjing, studying pattern recognition and intelligent systems. From 2007 to 2010, he works in state grid EPRI information and communications branch. Since 2010, he was working in Nanjing NARI group information systems integration corporation. Her main research interests include database, data disaster recovery, etc.

Yuqian Li was born in 1992, Jining Shandong China. In 2016, she graduated from Xiamen University, School of Computer Science and Technology, Department of Computer Science, studying computer science and technology. Since 2016, she works in Nanjing NARI group information systems integration corporation. Her main research interests include database, big data,etc.

Junjun Hu was born in 1984, Hubei, Wuhan, China. In 2008, he graduated from Hubei University of Technology, School of Management, specializing in e-commerce. Since 2013, he works in NARI group information systems integration corporation. His main research interests include power enterprise informatization, power big data analysis, IT infrastructure software and hardware products, information engineering management, etc.