Table of Contents

Abstract

Multidimensional analysis is the basic function of modern BI technology, and the multidimensional analysis on a single table (data set) is already quite mature, users can easily perform a variety of analysis actions such as pivot, aggregation and drilling. However, when performing multidimensional analysis on multiple associated data tables, it is not easy. Several solutions commonly used in industry all have flaws in different degrees, and are very hard for business users to understand, thereby making them often seek help from technical professionals. Consequently, it is difficult to implement a true online analysis.

Currently, most of the backend computing mechanisms of modern multi-dimensional analysis have returned to the relational database, that is to say, it needs to use SQL to operate. In our opinion, the reason for the above problem is that the definition of JOIN operation in SQL is too simple, and SQL has no sufficient ability to describe the intricate association.

This article sorts out the JOIN operations appeared in the process of multi-dimensional analysis, designs the DQL syntax, which keeps the style of SQL and can describe the association between data in a simpler way to enable the business users to perform complex association analysis.

Keywords

SQL, join, dimension, foreign key table, homo-dimension table, primary-sub table, DQL

0 Background

1. Data structures examples

The following lists 7 related tables in total, and we will take them as example in this article:

Tables	Columns	Notes
area		Area table
	aid	Area number, primary key
	name	Area name
employee		Employee table
	eid	Employee number, primary key
	name	Employee name
	gender	Employee gender
	age	Employee age
	aid	The area where employee is located, associating with area.aid
	sid	The sector the employee belongs to, associating with sector.sid
	salary	Employee salary
manager		Manager table
	eid	Employee number, primary key. Manager is also an employee; associating with employee.eid
	bonus	Manager bonus
sector		Sector table
	sid	Sector number, primary key
	name	Sector name
	mid	Sector manager, associating with employee.eid
	pid	Parent sector, associating with sector.sid
order		Order table
	oid	Order number, primary key
	eid	The employee who operates the order, associating with employee.eid
	date	Order date
	aid	Order area, associating with area.aid
	amount	Order amount
detail		Order details table
	oid	Order number, primary key; associating with order.oid
	seq	Sequence number, primary key
	product	Product
	quantity	Quantity
payment		Order payment table
	oid	Order number, primary key; associating with order.oid
	seq	Sequence number, primary key
	date	Payment date

	amount	Payment amount

Only the fields (columns) to be used in the examples are listed above. In practice, there are usually more fields.

Now, let's analyze some existing SQL-based multidimensional analysis methods for handling association and their disadvantages.

2. Pre-prepared wide table

A very common method to deal with association is to pre-join the tables to be associated to form a wide table (it's probably a logical table, i.e., a view). Having formed the wide table, the operation object of the user is equivalent to a single table, so they don't have to understand complex associations.

However, this solution will generate a large number of fields. In practice, hundreds of available fields are often generated after join, which makes it difficult for users to choose during operation. In addition, many fields generated during join are the same in type and similar in meaning, the users will find it hard to distinguish them by the name, resulting in confusion. For example, there is an aid in both the order table and the employee table, and both are associated with the area table. As a result, the area table will be joined multiple times, and the same field needs to be separately named multiple times.

Moreover, when the self-association (the pid and sid of sector table are associated) or the cycle association (the sid of employee table is associated with the sid of sector table; the mid of sector table is associated with the eid of employee table) occurs, infinite number of fields may be generated theoretically. Therefore, it is impossible to generate a wide table in advance.

3. Automatic association

Some BI products use automatic association. Users only need to select fields in multiple tables at will as they do on a single table, BI products will help them perform the join operation according to the association relationship known in advance.

Normally, the automatic association works based on the data type and business meaning of the field, which works well in simple cases, but will cause confusion in complex cases. For example, in the above case, the same area table may be associated with the aid field of order table, and may also be associated with the aid field of employee table after the order table is associated with the employee table, or may even need to be associated twice. Consequently, this solution cannot distinguish which case it is.

4. Conduct association by users

In recent years, BI products have begun to allow users to describe the association between tables by themselves, and gradually add new tables to associate with the existing query objects (which have been joined into wide table). Since the associated fields between the new table and existing wide table are clear, the above confusion situation can be avoided.

But, this solution will force users to understand the complex association between tables. If the number of tables is small, this solution works. Unfortunately, however, there are often dozens of tables in practice, and the association relationship is a mess, and it is not easy to sort out even for a

technical professional.

Additionally, this kind of association established by steps is difficult to modify. If it is found that a certain association in the previous steps is wrong, it is likely to overturn the previous steps and redo, for the reason that a forcibly modifying and resetting will also cause the above confusion situation. Moreover, if the association is conducted by users who are not familiar with the structure between tables, it will probably cause a logical error. For example, insufficient associations will cause the query scale to inflate in a wrong manner and may even exhaust the database resources, while excessive associations will result in query failure.

1 Analysis of JOIN operation

1. JOIN in SQL

The definition of JOIN in SQL is very simple: calculate the Cartesian product of two tables, and then filter according to a certain condition. Accordingly, the syntax form is *A* JOIN *B* ON.... In principle, the result of the Cartesian product operation should take the 2-tuples formed by the records of two tables as the member. However, since the table in SQL does not support the use of generic data to describe a 2-tuple whose members are the records, the result is simply processed into a new table formed after joining the record fields of two tables. This is the original meaning of the word JOIN in English, and it does not mean multiplication (Cartesian product).

The form of the filtering condition is not specified in the definition of JOIN. Theoretically, as long as the target result set is a subset of the Cartesian product of two source tables, it can be understood as JOIN operation. However, almost all JOINs we used in multidimensional analysis are the equivalent JOIN, which means that the filtering condition is one or more equality relationships (the relationship between these conditions is AND), and the syntax form is like *A* JOIN *B* ON $A.a_i = B.b_i$ AND ..., where a_i and b_i are fields of *A* and *B* respectively.

We only discuss the equivalent JOIN in this article.

2. Foreign key table

When the fields of table *A* are associated with the primary key of table *B*, *A* is called the **fact table**, *B* is called the **dimension table**, and the fields are called the **foreign key** pointing to *B* from *A*, and *B* is called *A*'s foreign key table. The foreign key table is the many-to-one relationship.

For example, in the data structures example:

The sid of the employee table is the foreign key pointing to the sector table;

The aid of employee table is the foreign key pointing to the area table;

The mid of sector table is the foreign key pointing to employee table;

…

In particular,

The pid of area table is the foreign key pointing to area table, i.e., the self-association;

In principle, the following statements are also true:

The eid of the manager table is the foreign key pointing to employee table;

The oid of the detail table is the foreign key pointing to order table;

However, since the primary keys (or parts) are involved in the fact table, we are more used to classifying such cases into the following types.

3. Homo-dimension table and primary-sub table

When the primary key of table *A* is associated with that of table *B*, we call *A* and *B* the **homo-dimension table** of each other. The homo-dimension table is the one-to-one relationship.

In the above case, the manager table and the employee table are the homo-dimension table of each other.

When the primary key of table *A* is associated with part of primary key of table *B*, *A* is called

the **primary table** of *B*, and *B* is the **sub-table** of *A*. The primary-sub table is the one-to-many relationship.

In the above case, the order table is the primary table of the detail table, and the detail table is the sub-table of order table.

From the definition point of view, a primary table may have multiple sub-tables, and a sub-table may also have multiple primary tables.

SQL does not distinguish between foreign key table and primary-sub table. From the SQL point of view, many-to-one and one-to-many relationships are only different in direction. As mentioned earlier, the order table can indeed be understood as the foreign key table of detail table, but the reason why we distinguish them here is that we will use different methods when simplifying the syntax.

4. Summary

After carefully examining the above three cases, we will find that all these association types involve the primary key, and there is no many-to-many relationship. Then, can we ignore the many-to-many situation?

Yes! In the multi-dimensional analysis business, there is almost no many-to-many situation. Therefore, we only need to consider three types of association: foreign key association, homo-dimension table association, and primary-sub table association, because these three associations have been able to meet most of the association requirements in multi-dimensional analysis (in fact, the many-to-many situation with business significance in a wider range of SQL operations are also very rare).

The way of defining JOIN in SQL is indeed very simple, i.e., Cartesian product operation, followed by filtering. But simple definition cannot fully reflect the characteristics of the operation, and BI software cannot use these characteristics to reduce the difficulty of understanding. Having re-sorted out the associations, we can make full use of these characteristics to simplify the operation.

2 DQL syntax

We continue to use the style of SQL syntax to describe DQL syntax, and all examples are based on the data structures given in chapter 0.

1. Foreign key attributization

DQL syntax rule 1: When the foreign key *k* of table *A* points to table *B*, and *b* is the field of *B*, *A.k.b* can be used to represent the field *B.b* after *A* JOIN *B*.

Example 1: Query the name of the sector to which the employee belongs

SQL	DQL
SELECT e.eid,e.name,s.name FROM employee e JOIN sector s ON e.sid=s.sid	SELECT eid,name,sid.name FROM employee

The field of the foreign key table can be regarded as the attribute of foreign key fields.

Example 2: Query manager's gender of the sector to which the employee belongs

SQL	DQL
SELECT e.eid,e.name,m.gender FROM employee e JOIN sector s ON e.sid=s.sid JOIN employee m ON s.mid=m.eid	SELECT eid,name,sid.mid.gender FROM employee

When the employee table is joined twice, it needs to take another name to distinguish in SQL, while in DQL, it only needs to reference one more layer, which seems to be a single-table operation.

We have assumed that the association field of foreign key table is the primary key, so the record in the dimension table corresponding to the foreign key is unique, that is to say, the employee.sid will correspond to a unique record in the sector table, and the employee.sid.mid will also correspond to a unique record in the employee table. Therefore, these writing methods are meaningful. However, if we continue to use the definition of JOIN in SQL, there will be no such assumption, and this syntax will be unavailable.

In the foreign key table relationship, the fact table and dimension table are not equivalent. We can only reference the fields of dimension table based on the fact table, not the other way around.

2. Homo-dimension table equatization

DQL syntax rule 2: When table *A* and *B* are the homo-dimension table of each other, and *b* is the field of *B*, *A.b* can be used to represent *B.b* after *A* JOIN *B*.

Example 3: Query all incomes of an employee

SQL	DQL
SELECT e.eid,e.name,e.salary+m.bonus FROM employee e JOIN manager m ON e.sid=m.sid	SELECT eid,name,salary+bonus FROM employee

The homo-dimension table will be treated as one table.

The records of homo-dimension table associated with the primary key can be uniquely determmined by the associated primary key without ambiguity.

In the homo-dimension table relationship, the two tables are equivalent, and any table can reference the fields of the other table.

3. Primary-sub table integration

DQL syntax rule 3: When primary table *A* and sub table *B* are in the primary-sub relationship, and *a* is the field of *A*, then *B.a* can be used to represent *A.a* after *A* JOIN *B*.

Example 4: List the products sold by employee 1

SQL	DQL
SELECT DISINTCT o.eid,d.product FROM order o JOIN detail d ON o.oid=d.oid WHERE o.eid=1	SELECT DISTINCT eid,product FROM detail WHERE eid=1

DQL syntax rule 4: When the primary table *A* and sub table *B* are in the primary-sub relationship, and *b* is the field of *B*, then *A.b* can be understood as a set composed of *B.b* after *A* JOIN *B*, and the aggregation operation can be performed.

Example 5: List the salesperson, sales volume and total collection of payment of each order

SQL	DQL
SELECT o.eid,d.quantity,p.amount FROM order o JOIN (SELECT oid,SUM(quantity) quantity FROM detail GROUP BY oid) d ON o.oid=d.oid JOIN (SELECT oid,SUM(amount) amount FROM payment GROUP BY oid) p ON o.oid=p.oid	SELECT eid,SUM(quantity),SUM(amount) FROM order

When the subtable references the primary table, the relationship between the two tables is similar to the homo-dimension table, that is, the record of primary table is uniquely determined by the associated primary key. When the primary table references the sub-table, the fields of sub-table should be regarded as set data, and only after aggregating the set data can it be aligned with the records of primary table.

In the primary-sub table relationship, the two tables are not equivalent as well, but the references in both directions are meaningful.

4. Mixed use of DQL syntax rules

These syntax rules can be used in a mixed way.

Example 6: List the orders and sales volume where the salesperson's location and the order's

sales location are the same:

SQL	DQL
SELECT o.oid,SUM(d.quantity) quanity FROM order o JOIN detail ON o.oid=d.oid JOIN employee e ON o.eid=e.eid WHERE e.aid=o.aid GROUP BY o.oid	SELECT oid, SUM(quantity) FROM order WHERE eid.aid=aid

5. The significance of DQL for multidimensional analysis

From the writing method of the above examples, we can see that these operations which should be described with JOIN in SQL do not require JOIN any more when being described in DQL, and there is only one table in the FROM clause, which means that all these multi-table association operations are already converted to single table operations.

BI software can use the logic of DQL to provide users with single table query function. After the user selects a table, it can provide the user with the fields of the tables associated with the selected table for selection in the form of extended fields of the selected table. In this way, as long as the users still master the single table query skills only, they are able to conduct a multi-table association query that is actually very complex, thus solving the association query difficulty faced by them.

Different from a pure single table, DQL extends the fields in a way that is not difficult to understand. The fields of foreign key table appear as the attribute of foreign key, which may be in multiple layers, and presented in a tree form; The fields of homo-dimension table are presented in the same form as they are in respective table; The fields of subtable are presented as set field, and the aggregation operation is a must.

Let's take the order table as an example, the list of fields presented is roughly as follows:

Fields	Corresponding fields
oid	
eid	
name	employee.name
gender	employee.gender
aid	employee.aid
name	area.name
sid	employee.sid
name	sector.name
mid	sector.mid
name	employee.name
…	
salary	employee.salary
bonus	manager.bonus
aid	

name	area.name
date	
product	{detail.detail}
quantity	{detail.quantity}
amount	{paryment.amount}
…	

(Note: the {} in the list represents the set field, and the aggregation operation is required when referencing).

This presenting form has the following advantages: for multi-layer foreign keys, self-association and cycle association, these fields can be expanded and collapsed instantly without having to work out a full wide table in advance; The hierarchical relationship is clearly presented and can be understood without special naming; In terms of difficulty of understanding, there is little difference compared to the pure single table.

DQL works only after correctly identifying the association type and association fields between the tables to be associated, so it needs to define the primary and foreign keys of the data table in advance, that is, the **metadata** information of the database. Metadata can be defined by the technical professionals, and the relationship between the primary key and foreign key in the metadata reflects the association attribute of the data itself, which has nothing to do with the query requirements of business users and can be set at one time. On the contrary, traditional wide-table solution often requires technical professionals to remodel for new business query requirements.

Once the relationship between the primary key and foreign key is defined in the metadata, the DQL-based BI software will present a single-table query to the user, and the user does not need to describe the association, which not only reduces the difficulty for users to use, but also avoids various errors that may occur when users conduct association by themselves because excessive or less association situation will not occur.

3 Further discussion

1. Dimension

Dimension is a common concept in multidimensional analysis, but there is no strict definition in industry. Usually, we call the attribute (field), which can be used for categorizing, the dimension, such as the year, area and product; as for the attribute used for aggregation operation like production, sales, etc., we don't call them the dimension.

In a set of relational database schema that fully defines the primary key and foreign key, we can try to define such a concept in a strict way: the field of primary key that is not a foreign key is called a **dimension**, and the table in which the field is in is called a **dimension table**.

For example, in the table structure of the previous example, the area.aid, employee.eid, sector.sid, order.oid, detail.seq and payment.seq are all the dimension. But manager.eid is not a dimension, in the database definition, it is a foreign key associated with employee.eid. Neither detail.oid nor payment.oid is a dimension.

The detail.seq is a dimension, but it does not act as the primary key field of foreign key table to associate with other tables, such dimension is called the **solitary dimension**, and this is also the case for payment.seq. Although the solitary dimension is the dimension by definition, it hardly participates in dimension-related operations.

There is an exception. Generally, we consider the date to be a dimension, but, according to the previous definition, the date field will not be regarded as a dimension.

In fact, the database should have a single-field date table logically, and the date type fields of other data tables can all be regarded as the foreign key associated with this date table. However, because the date related information can be calculated by the date itself (year, month, etc.), and there is no attribute to store, we usually don't create the date table in the physical database. Therefore, as long as the date table is restored logically, the date will conform to the dimension definition. Similarly, year and month can also be defined on logical dimension table.

Now let's see the age. Whether age is regarded as a dimension depends on the specific situation. When it is used for grouping (count the number of people of each age), it should be regarded as a dimension; but when it is used for operation (calculate the average age), it should not be regarded as a dimension.

Acturally, when we use age to group, we do not use the age value itself, but the age range. Since the age itself is a real number value, it cannot be used for grouping. Only the age range calculated from age (for example, one year corresponds to one age range) can be used for grouping, and can also be defined as a dimension in the way of logical dimension table.

In this case, we need to introduce the concept of dimension function. The **dimension function** is a function that takes the field value as parameter, and that the return value belongs to a certain determined dimension value set. The so-called dimension value set refers to the set consisting of all possible values of the said dimension in the dimension table (possibly a logical table).

With the help of dimension function, we can keep the strict definition of dimension without

contradicting conventional concept.

Similarly, the year and month calculated from date can also be understood as a dimension function, and any dimension level in multidimensional analysis can be regarded as dimension function.

2. Redefine the DQL

Now we can use the dimension concept to define the DQL in a stricter way.

First, let's redefine the dimension of field:

1. The dimension of the dimension field is itself;
2. The dimension of the foreign key field is that of the associated field in the foreign key table corresponding to this foreign key field;

This is a recursive definition.

For example, the dimension of order.oid is itself, and the detail.oid itself is not a dimension, and its dimension as a field is order.oid; the dimension of employee.eid is itself, and manager.eid is not a dimension, and its dimension as a field is employee.eid.

Then, let's define the homo-dimension table and primary-sub table.

Homo-dimension table: when the dimension sets of the primary key fields of two tables correspond to each other and are the same, the two tables are called the **homo-dimension**.

As mentioned earlier in this article, the homo-dimension table is defined from association, but two tables may not be associated directly. For example, if there is an outsourcing employee table E, and its primary key is associated with employee.eid, but not associated with manager.eid. In this case, E and manager will not be considered to be the homo-dimension if the previous definition is adopted, but will be regarded as homo-dimension according to the current definition.

Primary-sub table: when the primary key field dimension set of a table is the proper subset of that of another table, the former is called the primary table of the latter, and the latter is the sub-table of the former.

Similarly, once the primary-sub table relationship is defined from dimension, this relationship can be determined between tables without a direct foreign key association.

Therefore, it is easy to prove that the homo-dimension table of homo-dimension table is also the homo-dimension table, and the homo-dimension table is an equivalence relationship. A homo-dimension table of primary table is a primary table, a homo-dimension table of sub-table is a sub-table, and a sub-table of sub-table is also a sub-table.

Finally, let's define the **generalized field** of a table:

1. A field of table itself is its generalized field;
2. A field of the homo-dimension table of a table where a generalized field as an ordinary field is in is a generalized field;
3. When a generalized field is a foreign key, any field of its associated foreign key table is a generalized field;
4. The dimension function of a generalized field is a generalized field.

This is also a recursive definition.

For example: all the following fields are the generalized fields of employee table:

employee.aid, a field of table itself is a generalized field;

manager.bonus, a field of homo-dimension table is a generalized field;

employee.aid.name, a field of foreign key associated table is a generalized field;

employee.sid, a field of table itself is a generalized field;

employee.sid.mid, a field of foreign key associated table is a generalized field;

employee.sid.mid.name, a field of foreign key associated table of a generalized field is a generalized field;

employee.sid.mid.bonus, a field of homo-dimension table of foreign key associated table of generalized field is a generalized field.

Now, we can get DQL by allowing the following two points in a single table and non-netsting SQLstatement:

1. The generalized fields can be referenced as ordinary fields in the operation of table itself;
2. The generalized fields of sub-table, as the set value, can also be referenced in the operation of table itself after the aggregation operation is performed;

The dimension is defined on JOIN operation, and the DQL takes the dimension concept as its core, so it is called Dimensional Query Language, which is used to simplify JOIN operation.

Let's take a few more complex query examples:

1) Find out the male employees whose manager is female:

```
SELECT *
FROM employee
WHERE gender='Male' AND sid.mid.gender='Female'
```

Generalized fields are used directly in WHERE.

2) Count by department the average salary of employees whose salary is less than half of their manager's salary:

```
SELECT sid.name,AVERAGE(salary)
FROM employee
WHERE salary*2<(sid.mid.salary+sid.mid.bonus)
GROUP BY sid.name
```

Generalized fields can be used either for operation or as grouping key.

3) Count the sales volume and collection amount by the gender and department of the salesperson:

```
SELECT eid.sid.name,SUM(SUM(quantity)),SUM(SUM(amount))
FROM order
GROUP BY eid.sid.name,eid.gender
```

The aggregated sub-table fields can be understood as ordinary generalized fields, and then aggregated again.

3. Multi-field foreign key

The foreign keys discussed above are all the single-field foregien key. In reality, however, a foreign key may be composed of multiple fields. The aforementioned DQL syntax does not cover

this situation. To solve this problem, we need to name the foreign key.

Suppose the two fields t_1 and t_2 of the fact table T are composed to form a foreign key, and this foreign key is associated with the dimension table D. The primary key of D also consists of two fields (d_1 and d_2), which correspond to t_1 and t_2 respectively. If we name T's foreign key the K, then K can be understood as a generalized field of T, which means that in the association query T JOIN D ON $t_1=d_1$ AND $t_1=d_2$, the field F of D can use $T.K.F$ to reference in DQL.

For the foreign key composed of more fields, it is in a similar situation.

When we change the first definition of generalized fields to:

1. A field and a foreign key of table itself are its generalized field;

It will work for the multiple fields composite foreign key.

For example, in the following simplified data structure:

Tables	Columns	Notes
course		Course table
	year	Year, primary key
	course	Course name, primary key
	teacher	Teacher name
score		Score table
	student	Student name
	year	Academic year, foreign key
	course	Course name, foreign key
	score	Student score

The year and course in the score table form a composite foreign key associated with course table.

Now we want to calculate the average score of students taught by each teacher.

In the score table, name the composite foreign key (year, course) the yc, we can easily write this query in DQL:

```
SELECT yc.teacher,AVERAGE(score)
FROM score
GROUP BY yc.tearcher
```

Corresponding SQL:

```
SELECT course.teacher,AVERAGE(score.score)
FROM score JOIN course ON score.year=course.year AND score.course=course.course
GROUP BY course.tearcher
```

If the name of composite foreign key is defined in the metadata in advance, then DQL syntax and BI software interface can use the name directly.

In fact, the single-field foreign key can be regarded as a special case of the multi-field foreign key. If it is a single-field foreign key, there is no need to specifically name it. Instead, we can directly use the foreign key field name as its name. Of course, we can give it a new name.

A slight difference between the single-field foreign key and multi-field foreign key is that the former itself has the field value, which can be directly referenced to participate in the calculation or

be taken out, while the field value of the latter is a set (including multiple field values), which can be understood as the corresponding records in dimension table. However, for the latter one, no matter it is a set or record, it is the data type that cannot be directly and explicitly supported in SQL (as the field value). Therefore, the latter itself cannot be referenced as a generalized field value, instead, only the field of its corresponding dimension table record can be referenced.

For example, the DQL in the above example cannot directly reference score yc to participate in the calculation or to be taken out, and can only further use the score.yc.teacher.

Modifying the DQL syntax rule 1 as follows can make the syntax stricter.

1. A generalized field that is not the composite foreign key can, as ordinary field, be referenced during the operation of table itself;

If the SQL capability is extended to allow set and record to be used as field value, this restriction will be unnecessary.

4. Dimension alignment syntax

A further alignment operation can be performed to the query results of DQL.

Let's go back to the table structure example at the beginning of the article. We want to count the order amount and the payment amount by date. Write it in SQL is like this:

```
SELECT NVL(o.date,p.date) date,o.amount o_amount, p.amount p_amount
FROM
    (SELECT date, sum(amount) amount FROM order
    GROUP BY date ) o
    FULL JOINSELECT date,sum(amount) amount FROM payment
    GROUP BY date ) p
```

Obviously, the JOIN here has business meaning. So, which of the three types of JOINs mentioned above does it belong to?

It is the JOIN made between two subqueries, and each subquery has a GROUP BY. As we know that the grouped result set will naturally take the grouping key as the primary key. In other words, both the subqueries here will take the date as primary key. If each subquery is regarded as a table, then they are in homo-dimension relationship, and the JOIN here is the homo-dimension association. As a result, it does not exceed the scope of the above three JOINs.

However, this homo-dimension relationship is instantly calculated and cannot be described in the metadata in advance. Therefore, the aforementioned DQL syntax cannot be directly applied to simplify it into a JOIN-free syntax.

To solve this kind of operation, we add a DQL syntax rule:

3. The FROM clause allows multiple tables to be aligned by the same dimensions (after grouping and aggregation); The expressions in the SELECT clause can reference the tables involved in the FROM clause, but mixing expression is prohibited.

Based on this rule, the above query can be written in DQL:

```
SELECT order.SUM(amount) o_amount, payment.SUM(amount) ON date
```

```
FROM order GROUP BY date FULL JOIN payment GROUP BY date
```

This statement continues to use the existing keywords of SQL, ON is used to describe the target dimension for aligning, and JOIN is used to conjoin multiple tables. The JOIN and ON used here are very different from their meanings in SQL, and what is continued to be used is only the two words themselves rather than their meanings in SQL.

JOIN in SQL is a verb that describes some kind of operation between two tables. When writing, attention needs to be given to the relationship between tables. If the association table is missing, the association operation cannot be written. When reducing the number of tables or modifying the table, the corresponding association table needs to be taken into consideration at the same time. Consequently, both the understanding difficulty and error probability are relatively high.

Unlike the JOIN in SQL, the JOIN in DQL is essentially a conjunction, which is only used to separate multiple tables to be aligned. There is no direct association between multiple tables. The operations on each table can be written independently, and there is no need to pay attention to the relationship between tables, even there is no need to consider whether other tables exist. This writing form not only avoids the subquery and makes the statement simpler, but also reduces the probability of errors.

In SQL, aligning multiple tables can also be performed using UNION. The above example can be written as:

```
SELECT date,sum(o.amount), sum(p.amount)
FROM
    (SELECT date, amount o_amount, null p_amount FROM order
    UNION ALL
    SELECT date,null o_amount, amount p_amount FROM payment )
GROUP BY date
```

We use the following statement to further improve DQL syntax:

4. The FROM clause allows for performing a union operation on multiple tables (after being grouped and aggregated into homo-dimension tables). The expressions in SELECT clause can reference the tables involved in FROM, but mixing expression is prohibited;

Based on this rule, the above query can be written in DQL:

```
SELECT order.sum(o.amount), payment.sum(p.amount) ON date
FROM order GROUP BY date UNION ALL payment GROUP BY date
```

For the application scenarios of DQL, there is little difference between JOIN syntax and UNION syntax in description ability, we can choose either one.

Epilogue

DQL changes SQL's understanding of JOIN operation. Using DQL syntax can greatly simplify the association operation in SQL, and convert most multi-table association queries to single-table query, thereby reducing the difficulty of understanding and writing, and the probability of errors. With the support of DQL, BI software is able to easily work out an interface understandable by business personnel, so as to perform quite complex association query, and make the on-line analysis no longer rely often on technical professionals to model on the data structure.

The design goal of DQL is only to provide the back-end data support for the multidimensional analysis interface, and it cannot replace SQL to perform more complex and flexible operations. To use DQL, it needs to define the metadata describing the relationship between the primary key and foreign key in advance, and parse out the dimension structure information from the metadata. Only in this way can the association operation be correctly and automatically implemented. For tables whose primary and foreign key relationships are not set in the metadata, the association cannot be automatically implemented in DQL. Moreover, it’s not allowed in DQL to establish the association instantly as arbitrary and flexible as SQL because these scenarios are not what DQL was designed for.

We have implemented DQL engine. One version implements rules 1/2/3 based on SQL, and can translate DQL statement to SQL and work on any database that supports SQL; Another version implements rules 1/2/4 without relying on SQL, and can directly execute DQL and obtain better computing performance.