

JOIN



CONTENTS

- 1 Understanding Join
- 2 Foreign key table
- 3 Same dimension Main & Sub table
- 4 Non-Equijoin
- 5 Converting SQL subqueries into JOIN

01

Understanding Join

1. Cartesian product

Employee

ID	NAME	DEPT
1	David	1
2	Daniel	2
3	Andrew	1



Department

ID	NAME
1	Sales
2	R&D



ID	NAME	DEPT	ID	NAME
1	David	1	1	Sales
1	David	1	2	R&D
2	Daniel	2	1	Sales
2	Daniel	2	2	R&D
3	Andrew	1	1	Sales
3	Andrew	1	2	R&D

2. Conditional filtering

ID	NAME	DEPT	ID	NAME
1	David	1	1	Sales
4	David	4	2	R&D
2	Daniel	2	4	Sales
2	Daniel	2	2	R&D
3	Andrew	1	1	Sales
3	Andrew	4	2	R&D

Employee.DEPT =
Department.ID



ID	NAME	DEPT	ID	NAME
1	David	1	1	Sales
2	Daniel	2	2	R&D
3	Andrew	1	1	Sales

Employee

ID	NAME	DEPT
1	David	1
2	Daniel	2
3	Andrew	1

JOIN



Department

ID	NAME
1	Sales
2	R&D



Employee	Department
[1, David, 1]	[1, Sales]
[2, Daniel, 2]	[2, R&D]
[3, Andrew, 1]	[1, Sales]

SPL joins two or more sets and takes the binary group composed of set members as members instead of simply expanding the data structure of all sets. SPL is not only more in line with the concepts and intentions of JOIN, but also clearer and more concise than SQL.

Common Types of Equivalent JOIN



**Foreign key
table**



**Same
Dimension
Table**

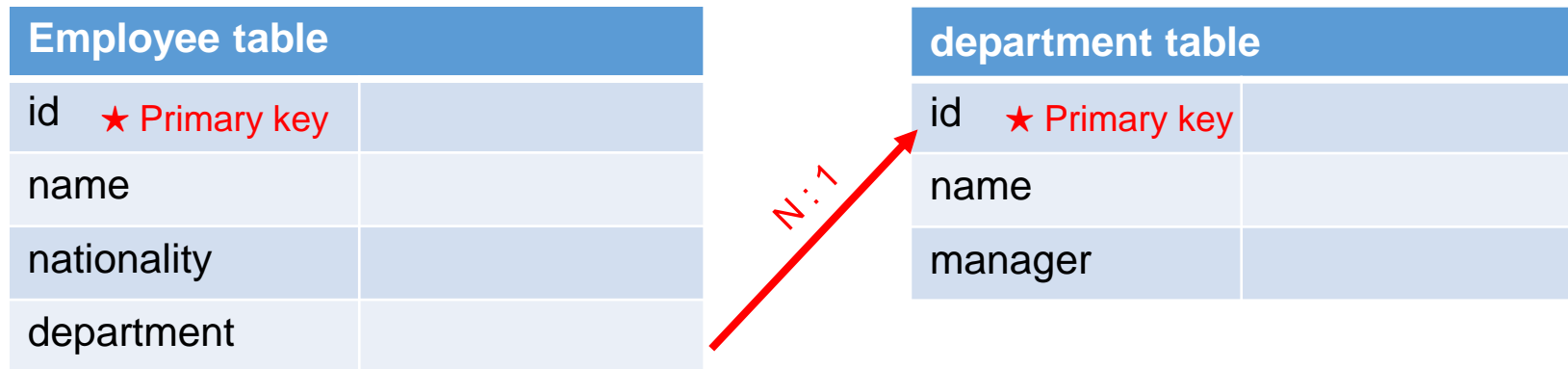


**Main & Sub
table**

In reality, most JOINS are equivalent JOINS, and the above three JOINS have covered most cases of equivalent JOINS.

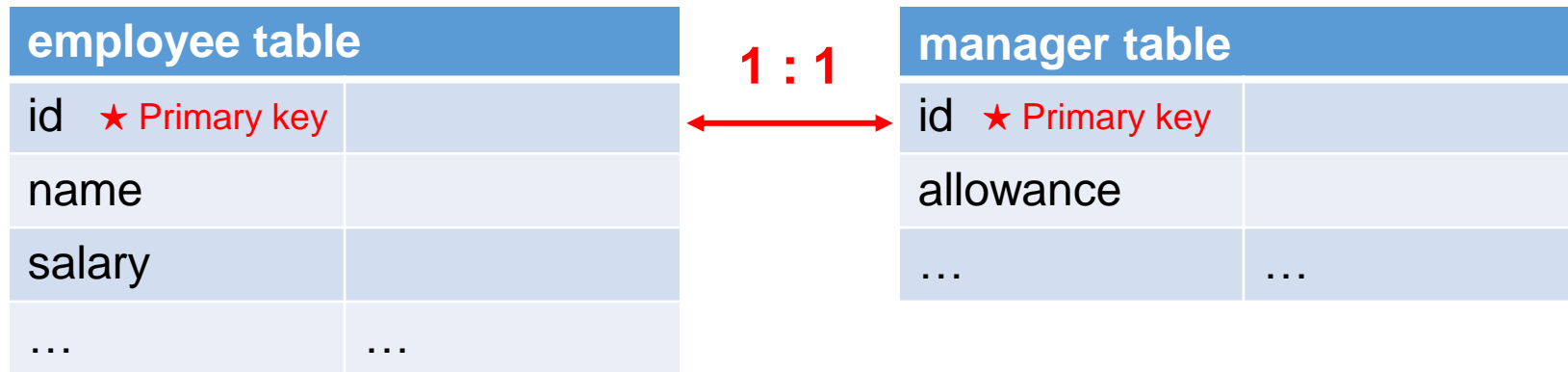
SPL makes full use of these features to create simpler writing formats and achieve more efficient computing performance.

Foreign key table



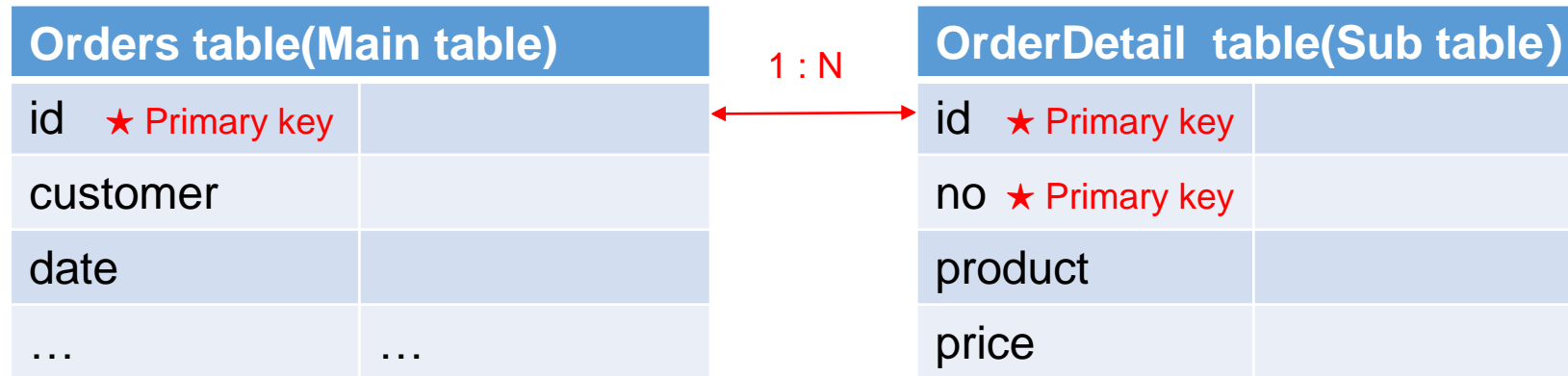
Some fields of Table A are associated with the primary key of Table B. The field associated with the primary key of table B in table A is called the foreign key of A to B, and B is also called the foreign key table of A. The foreign key table is a many-to-one relationship.

Same Dimension Table



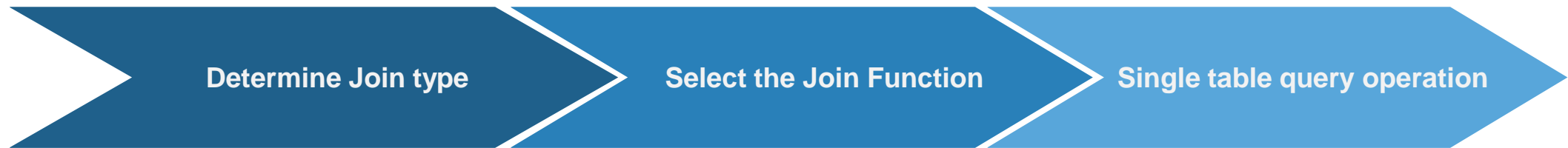
The primary key of table A is associated with the primary key of table B. A and B are called the same dimension tables. The same dimension table is a one-to-one relationship, and the relationship with the same dimension table is equal.

Main & Sub table



The primary key of table A is associated with part of the primary keys of table B. A is called the main table and B is called the sub-table. The main and sub-table is a one-to-many relationship.

Process for using Joins



First, the type of Join is determined.
Common types are: foreign key table,
same dimension table and main &
sub table.

Use the corresponding function
according to the Join type. Detailed
information will be provided in later
chapters.

After joining, members of other
tables can be accessed in the
form of “field. Property”.

We changed the view on JOIN operation, abandoned the idea of Cartesian product, and regarded multi-table Join operation as a slightly more complex single-table operation. In this way, we can basically eliminate the Join from the most common equivalent JOIN operation, and it is much simpler to write and understand.

02

Foreign key table

Foreign key table – Foreign key Objectification

By objectifying the foreign key, the foreign key field can be converted into the corresponding reference in the foreign key table, so that it can be processed as a single table.

(1) Single foreign key

Please select all the employees in the product department.

employee table	
id ★ Primary key	
name	
nationality	
department	

department table	
id ★ Primary key	
name	
manager	



Foreign key table – Foreign key Objectification

Employee table

ID	NAME	NATIONALITY	DEPARTMENT
103	Rudy	American	Product



Foreign key objectification

ID	NAME	NATIONALITY	DEPARTMENT
103	Rudy	American	[11, Product, Robert]

ID	NAME	MANAGER
11	Product	Robert

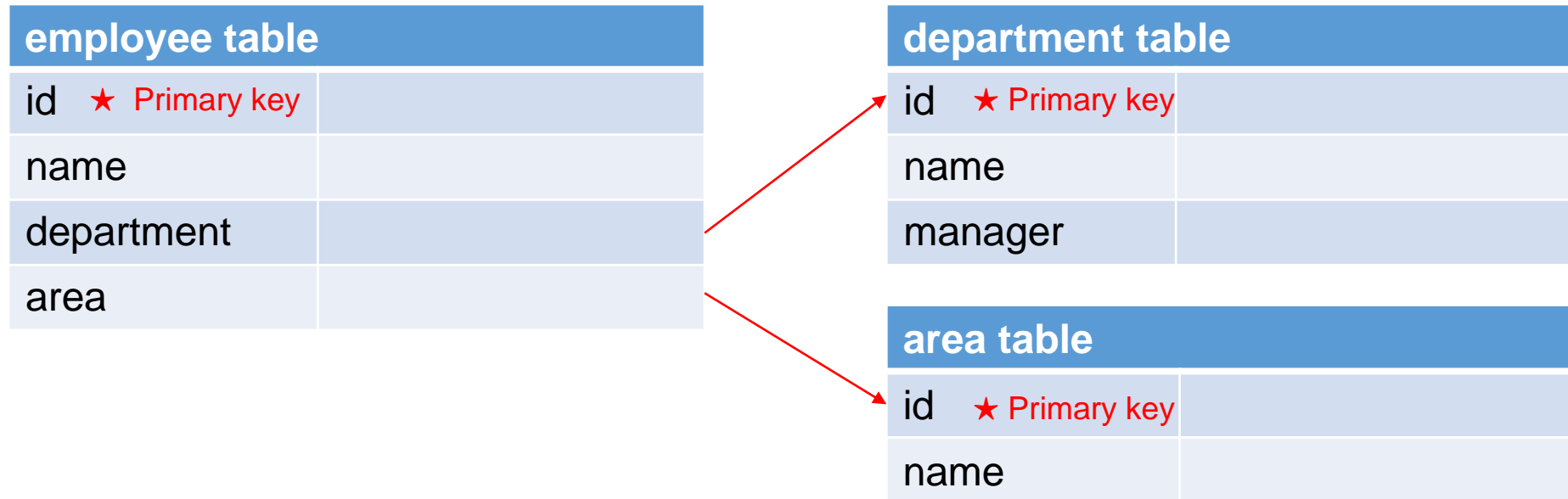
After foreign key objectification, department. name in the employee table is the name of the Department to which it belongs. The complete SPL is as follows:

	A	B
1	=db.query("select * from employee")	=db.query("select * from department")
2	=A1.switch(department, B1:id)	=A2.select(department.name=="Product")

Foreign key table – Foreign key Objectification

(2) Single-layer multiple foreign keys

The area field of the employee table is also a foreign key to the area table. Please select the employees in Beijing's product department.



Foreign key table – Foreign key Objectification

EMPLOYEE

ID	NAME	DEPARTMENT	AREA
103	Rudy	11	101

Foreign key objectification

ID	NAME	DEPARTMENT	AREA
103	Rudy	[11,Product,Robert]	[101, Beijing]

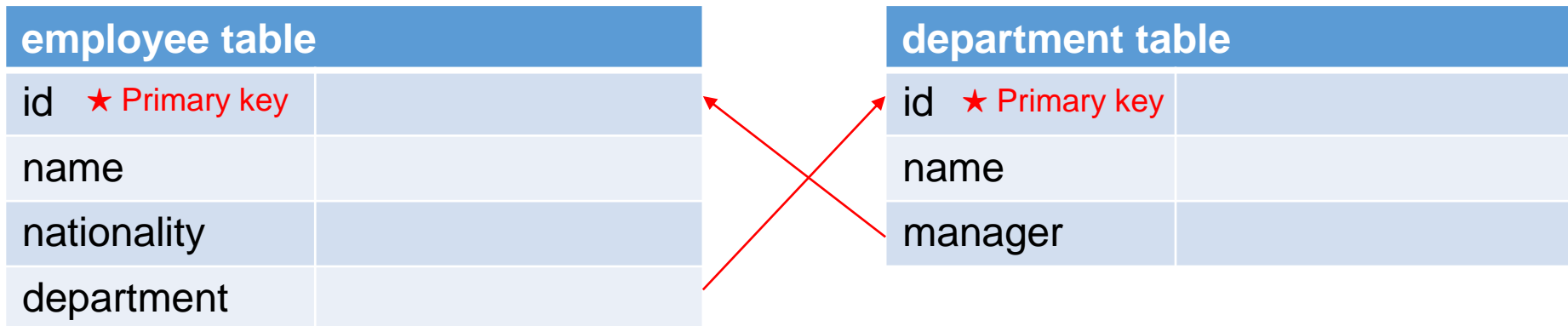
The complete SPL is as follows:

	A	B
1	=db.query("select * from employee")	=db.query("select * from department")
2	=db.query("select * from area")	
3	=A1.switch(department, B1.id; area,A2.id)	=A3.select(department.name=="Product" && area.name=="Beijing")

Foreign key table – Foreign key Objectification

(3) Multi-layer foreign key

Which American employees have a Chinese manager?



Foreign key table – Foreign key Objectification

For multi-layer foreign key join, only need to objectify the foreign key layer by layer.

1

department

ID	NAME	MANAGER
11	Product	Robert

Foreign key objectification



ID	NAME	MANAGER
11	Product	[101, Robert, Chinese, Product]

2

employee

ID	NAME	NATIONALITY	DEPARTMENT
103	Rudy	American	Product

Foreign key objectification



ID	NAME	NATIONALITY	DEPARTMENT
103	Rudy	American	[11,Product,[101,Robert,Chinese,Product]]

Foreign key table – Foreign key Objectification

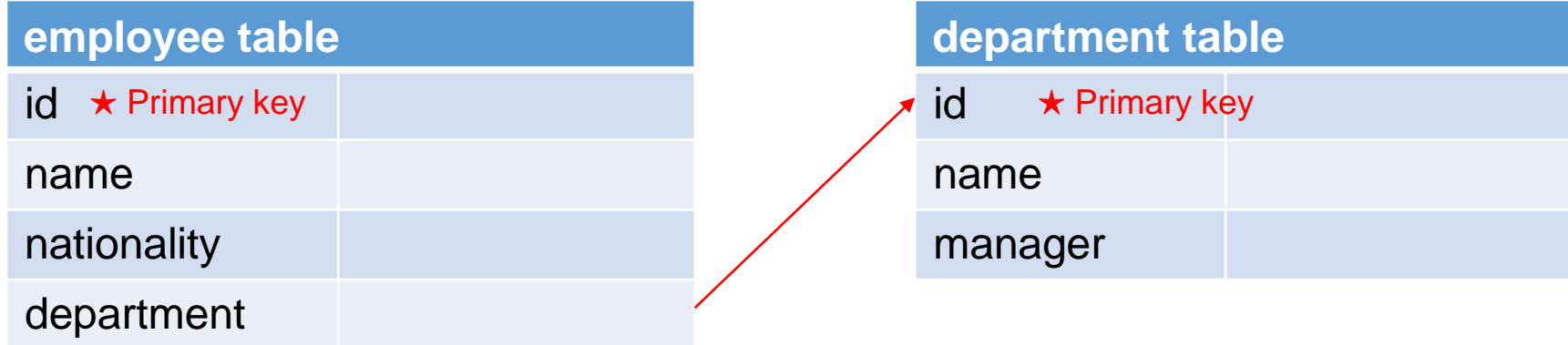
The complete SPL is as follows:

	A	B
1	<code>=db.query("select * from employee")</code>	<code>=db.query("select * from department")</code>
2	<code>=B1.switch(manager, A1:id)</code>	<code>=A1.switch(department, A2:id)</code>
3	<code>=B2.select(nationality=="American" && department.manager.nationality=="Chinese")</code>	

Foreign key table – Cases unsupported by Foreign key Objectification

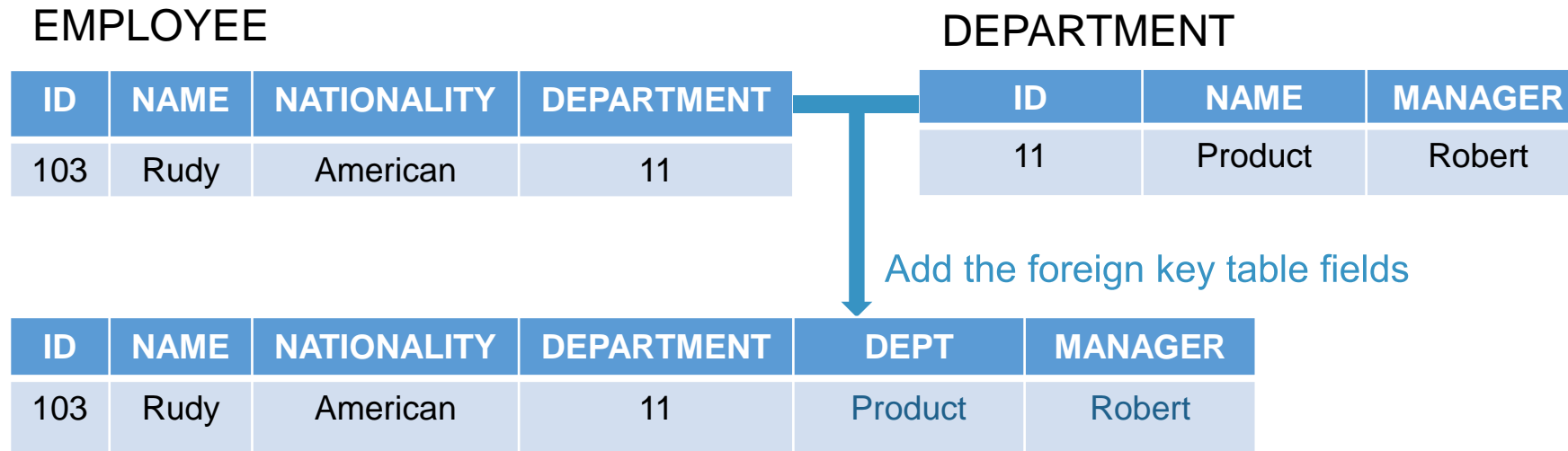
(1) Foreign key pointing to blank record

When the direction of the Department is empty, objectification will cause the code value of the field to be lost.



Foreign key table – Cases unsupported by Foreign key Objectification

Solution 1: Do not use foreign key objectification, add the required foreign key table fields.

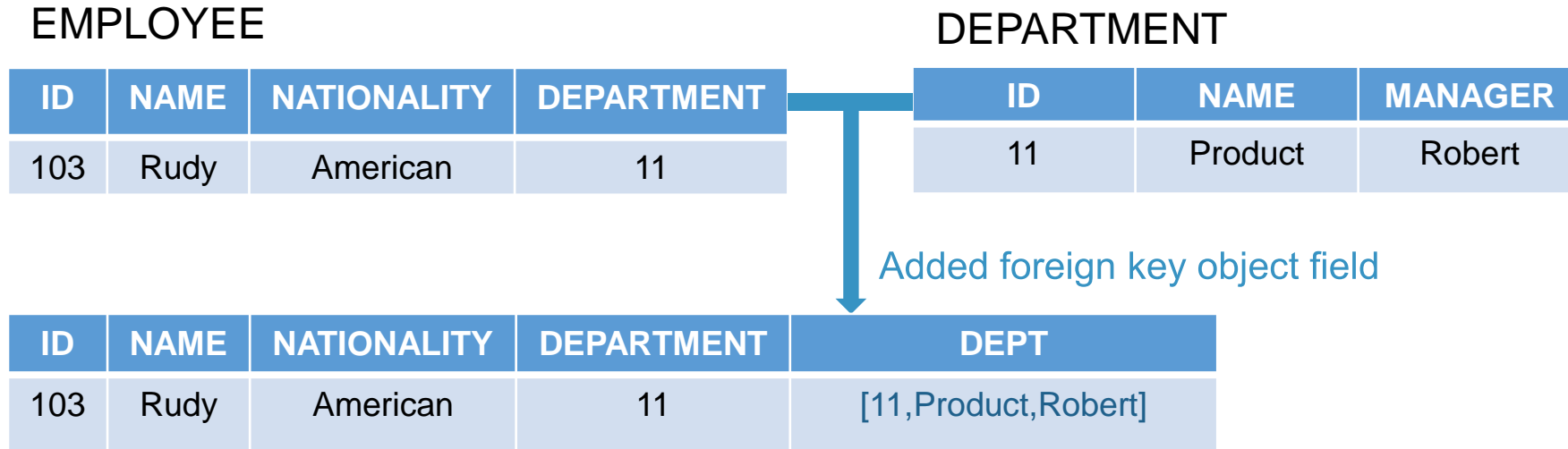


The complete SPL is as follows:

	A	B
1	<code>=db.query("select * from employee")</code>	<code>=db.query("select * from department")</code>
2	<code>=A1.join(department, B1.id, name:dept, manager)</code>	<code>=A2.select(department.name=="Product")</code>

Foreign key table – Cases unsupported by Foreign key Objectification

Solution 2: Foreign keys are still objectified, but foreign key objects are generated on new fields.



The complete SPL is as follows:

	A	B
1	<code>=db.query("select * from employee")</code>	<code>=db.query("select * from department")</code>
2	<code>=A1.join(department, B1.id, ~:dept)</code>	<code>=A2.select(department.name=="Product")</code>

Foreign key table – Cases unsupported by Foreign key Objectification

(2) Multi-field foreign key

Please choose the names of the products whose order payment is greater than 500 in 2018.



Foreign key table – Cases unsupported by Foreign key Objectification

1

OrderDetail

ID	ORDER	PRODUCT	PRICE
10248	1	17	238
10248	2	18	475

Foreign key
objectification

OrderDetail

ID	ORDER	PRODUCT	PRICE
10248	1	[17, cake]	238
10248	2	[18, apple]	475

2

OrderPayment

ID	ORDER	ORDERNO	DATE
101	10248	1	2012-07-26
103	10248	2	2012-08-15

Add the foreign key object

OrderPayment

ID	ORDER	ORDERNO	DATE	DETAIL
101	10248	1	2012-07-26	[10248, 1, [17, cake], 238]
103	10248	2	2012-08-15	[10248, 2, [18, apple], 475]

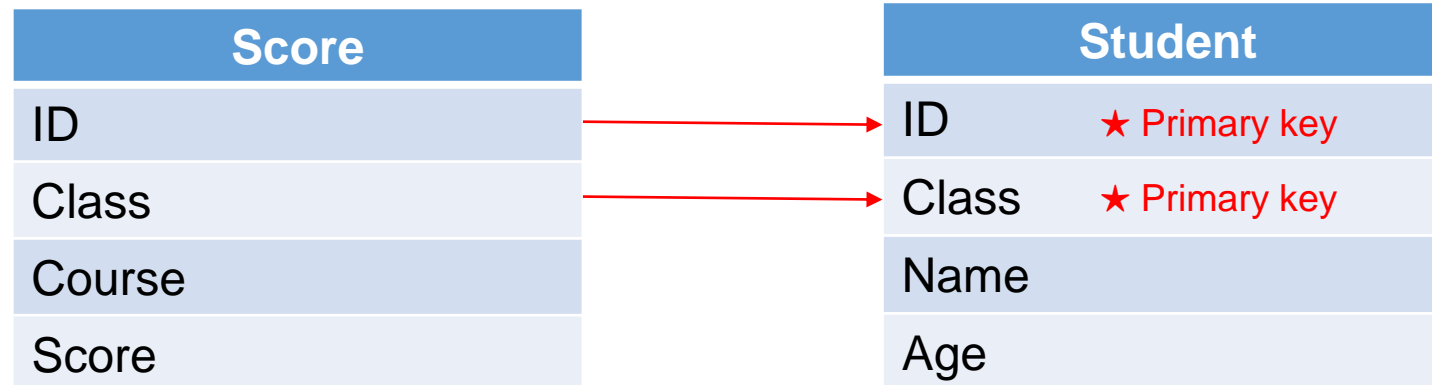
Foreign key table – Cases unsupported by Foreign key Objectification

The complete SPL is as follows:

	A	B
1	=db.query("select * from OrderPayment")	=db.query("select * from OrderDetail")
2	=db.query("select * from Product")	=B1.switch(product, A2:id)
3	=A1.join(order:orderno,B2:id:no,~:detail)	=A3.select(year(date)==2018)
4	=B3.select(detail.price>500)	=A4.new(id,date,detail.product.name:name,detail.price:price)

Foreign key table - Conditional cases of foreign key table

Query the grades of the students in Class One.



Foreign key table - Conditional cases of foreign key table

Solution 1: Filter the foreign key table first, then filter the join.

1

Student

ID	Class	Name	Age
1	Class 1	David	16
2	Class 2	Daniel	17

Filter

Student

ID	Class	Name	Age
1	Class 1	David	16

2

Score

ID	Class	Course	Score
1	Class 1	Math	89
2	Class 2	English	95

Join filtering

Score

ID	Class	Course	Score
1	Class 1	Math	89

Foreign key table - Conditional cases of foreign key table

The complete SPL is as follows:


	A
1	=db.query("select * from Score")
2	=db.query("select * from Student")
3	=A2.select(Class=="Class 1")
4	=A1.join@i(ID, A3:ID)

A3: After filtering classes, multiple primary key Joins (class + student ID) become single primary key Join (student ID).

A4: The join@i option is used to delete the record when the corresponding value is not found for join filtering.

Foreign key table - Conditional cases of foreign key table

Solution 2: If the condition is for the primary key, you can also use multi-field join filtering.

Score				Join filtering	Student			
ID	Class	Course	Score		ID	Class	Name	Age
1	Class 1	Math	89		1	Class 1	David	16
1	Class 1	English	94		2	Class 2	Daniel	17
2	Class 2	Math	92	
2	Class 2	English	95					
...					

Join fields are the two fields of student ID + class, which can specify the class as a constant condition (class 1) when joining, thus realizing join filtering.

Foreign key table - Conditional cases of foreign key table

The complete SPL is as follows:

	A
1	=db.query("select * from Score")
2	=db.query("select * from Student")
3	=A1.join@i(ID:"Class 1", A2:ID:Class)

Direct multi-primary key join can keep the index of student table, thus ensuring the speed of join.

03

Same dimension Main & Sub table



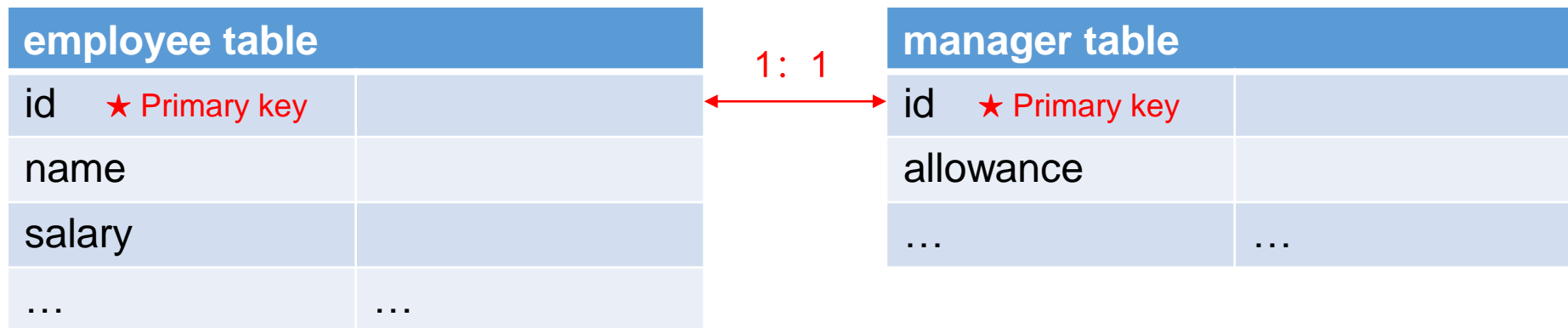
Same dimension table and Main & Sub table

There are many similarities between the same dimension table and the main & sub table, so they are explained in the same chapter.

The relationship between the same dimension tables is equal. Once the main & sub tables are transformed into the same dimension tables, they can be operated as single table.

3.1 Same dimension table

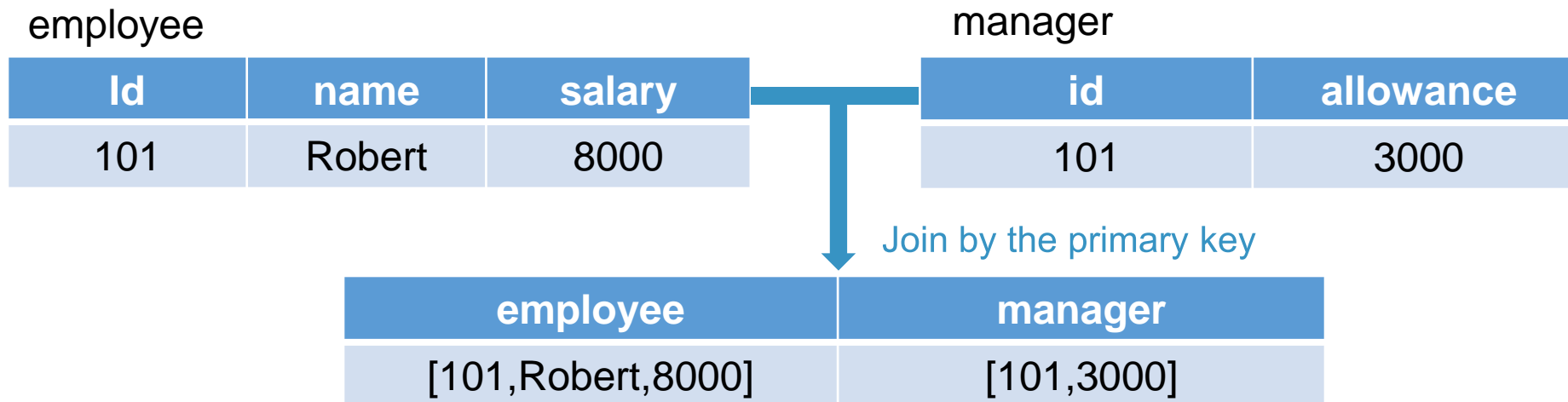
The total income (plus allowances) of all employees (including managers) should be counted.



Employee table is used to store employee information. Managers are also employees. Managers have more attributes than ordinary employees and use a manager table to save them. The two tables share the same employee number.

Same dimension Main & Sub table

In case of the same dimension table type, Join directly according to the primary key.



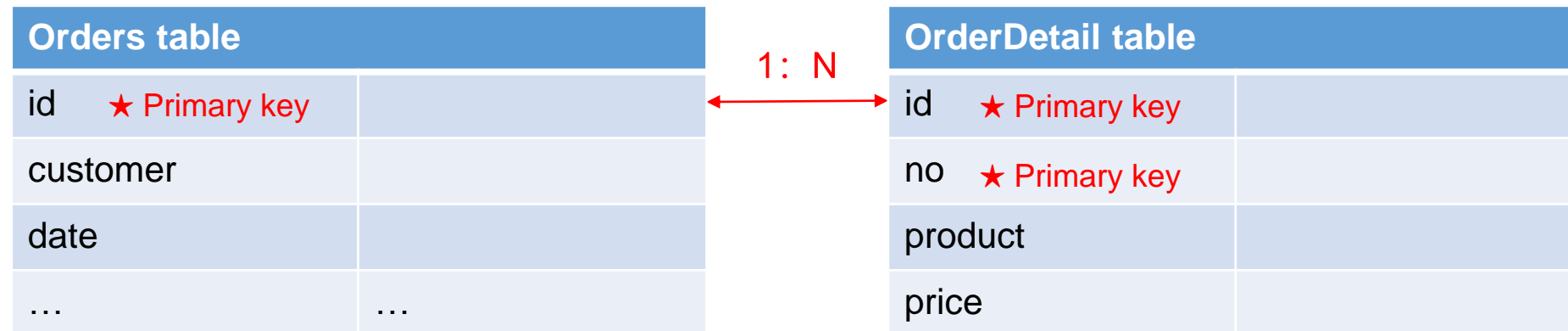
The complete SPL is as follows:

	A	B
1	<code>=db.query("select * from employee")</code>	<code>=db.query("select * from manager")</code>
2	<code>join(A1:employee,id;B1:manager,id)</code>	<code>=A2.new(employee.id,employee.name,employee.salary+manager.allowance)</code>

3.2 Single Main & Sub table

Example: The primary key of the order table is id, and the primary key of the order detail table is ID and No. The primary key of the former is part of the latter.

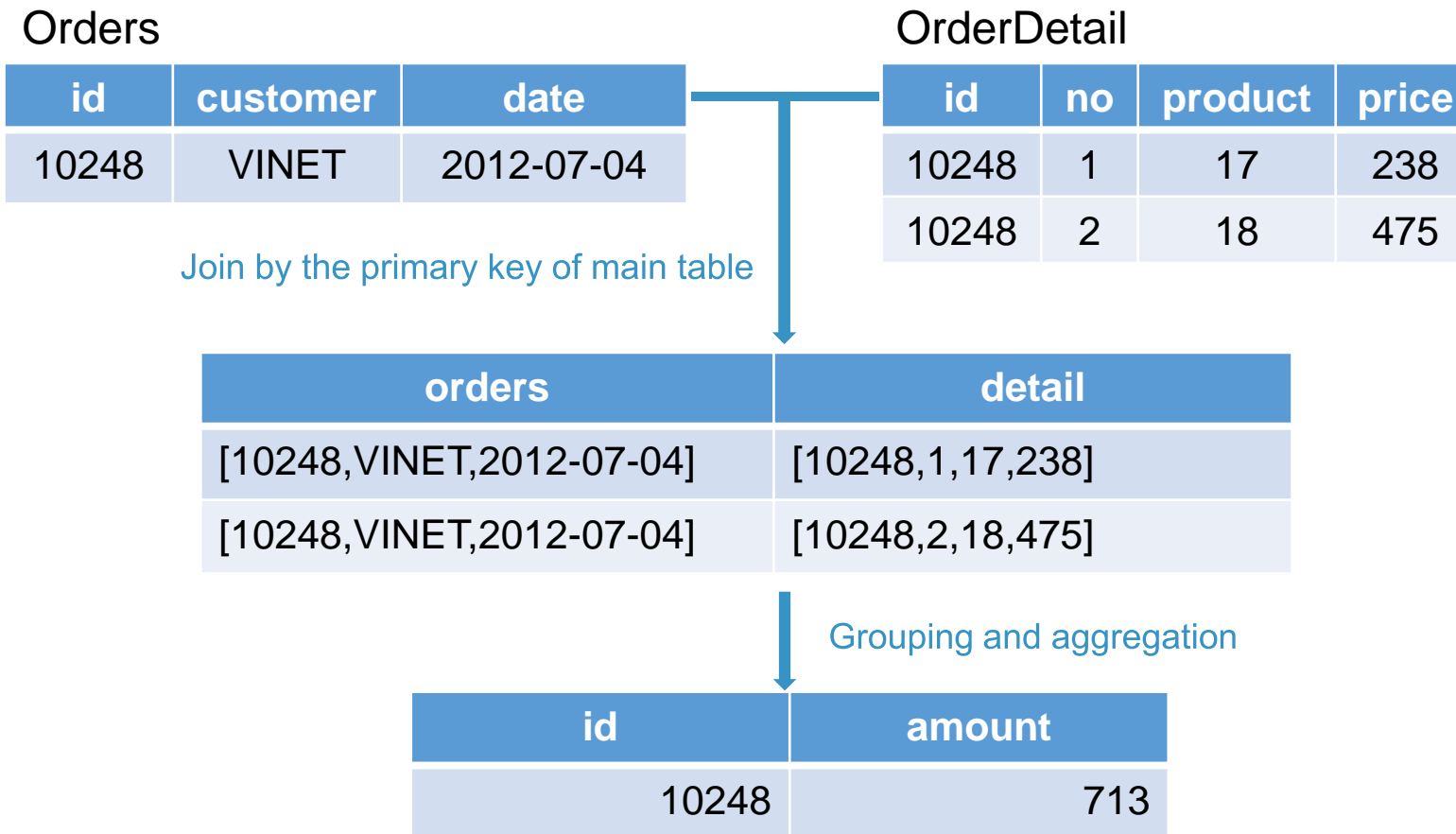
Now we need to calculate the total amount of each order.



The relationship between main and sub table is unequal, and the reference of main table from sub table is similar to that of foreign key table.

Here we mainly introduce how to refer to sub tables from the main table.

Same dimension Main & Sub table



Same dimension Main & Sub table

The complete SPL is as follows:

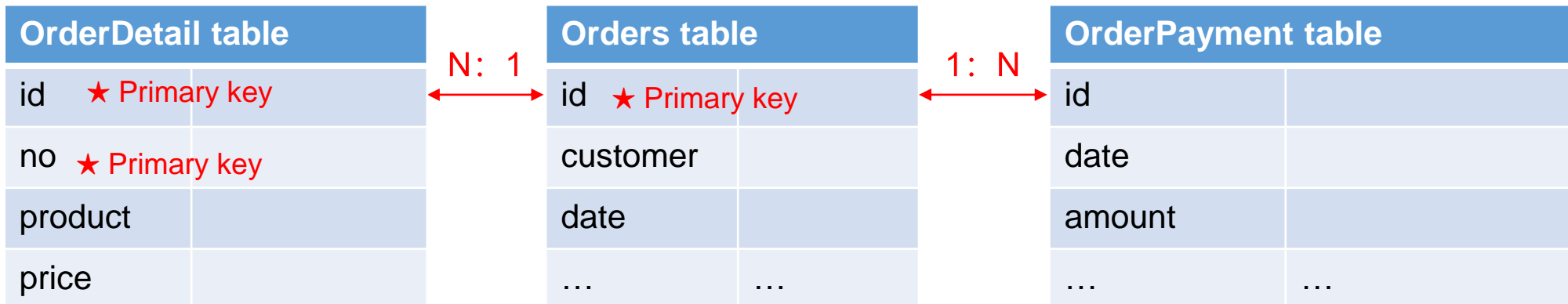
	A	B
1	=db.query("select * from Orders")	=db.query("select * from OrderDetail")
2	=join(A1:Order,id;B1:Detail,id)	=A2.groups(Order.id:order,Order.customer:customer; sum(Detail.price):price)

If the order and order details are orderly for id, we can use join@m to merge them in order to improve the speed and performance of join.

	A
2	=join@m(A1:Order,id;B1:Detail,id)

3.3 Multiple Main & Sub tables

If the order table also has a sub table to record the repayment. Now we want to know which orders haven't been paid back, that is, orders whose cumulative amount of payment is less than the total amount of the order.



It is not right to simply join the three tables. The order details table and the order payment table will have a many-to-many relationship.

Id is not the only primary key in the order detail table and the order payment table. How to turn them into the unique factual primary key?

Same dimension Main & Sub table

1

OrderDetail

id	no	product	price
10248	1	17	238
10248	2	18	475

Sub table
grouping by id

member
[[10248,1,17,238],[10248,2,18,475]]

Id becomes the
primary key

OrderPayment

id	date	amount
10248	2012-07-26	238
10248	2012-08-15	475

Sub table
grouping by id

member
[[10248,2012-07-26,238],[10248,2012-08-15,475]]

Id becomes the
primary key

2

Orders

id	customer	date
10248	VINET	2012-07-04

Join by the primary key id of the main table

order	detail	payment
[10248,VINET, 2012-07-04]	[[10248,1,17,238],...]	[[10248,2012-07-26,238],...]

Same dimension Main & Sub table

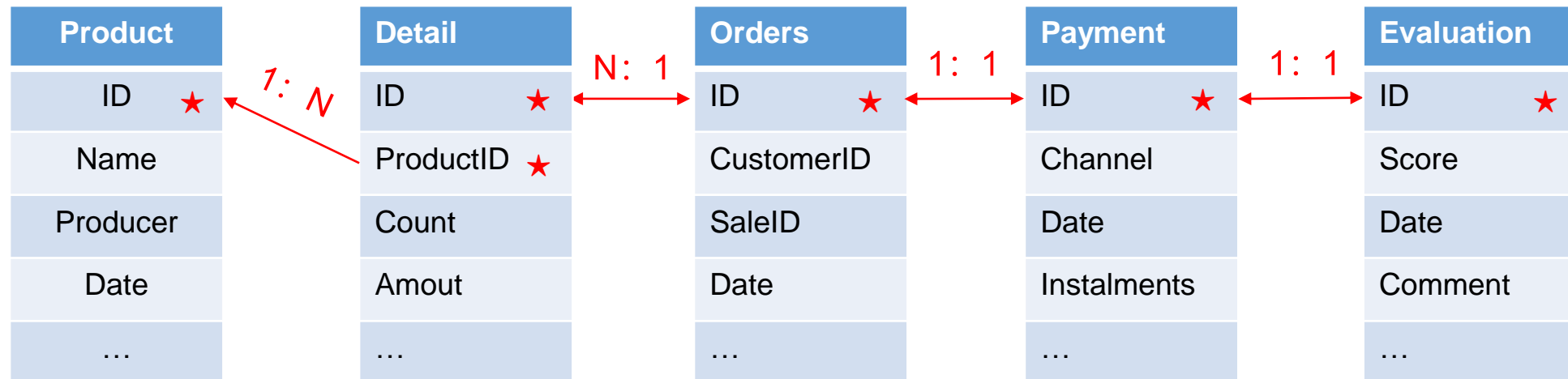
The complete SPL is as follows:

	A	B
1	=db.query("select * from Orders")	=db.query("select * from OrderDetail")
2	=db.query("select * from OrderPayment")	
3	=B1.group(id)	=A2.group(id)
4	=join(A1:Orders,id; A3:Detail, id; B3:Payment, id)	=A4.new(Orders.id:id,Orders.product:pr oduct,Detail.sum(price):price,Payment. sum(amount):amount)
5	=B4.select(amount<price)	

When sub tables are grouped by the primary key id of the main table, the id becomes the primary key in fact. Then we can treat the same dimension main & sub table as the same dimension table.

3.4 The combining of foreign key table, same dimension table and main & sub table

Query the order information that includes "water" in the name of the commodity, the order was placed in January 2019, the total amount of the order was more than 200 yuan, no instalment was used, and the evaluation was five stars.



This example contains both the same dimension table and the main & sub table, as well as the foreign key table. The problem needs to be solved by disassembling: first, the foreign keys are objectified, then the sub tables are grouped according to the primary key of the main table, and finally, join the same dimension tables.

Same dimension Main & Sub table

1

ID	ProductID	Count	Amout
10248	1	17	238

Foreign key
objectification

ID	ProductID	Count	Amout
10248	[1, cake,...]	17	238

2

ID	ProductID	Count	Amout
10248	[1,cake,...]	17	238
10248	[2,apple,...]	18	475

Grouping by id
for sub table

member			
[[10248,[1,...],17,238],[10248,[2,...],18,475]]			

3



Join by primary key of main table

Orders	Detail	Payment	Evaluation
[10248,VINET, 2012-07-04]	[[10248,[1,...],17,238],[10248,[2,...],18,475]]	[10248,3,2012-07-04,0]	[10248,5,2012-07-16,"Good"]

Same dimension Main & Sub table

The complete SPL is as follows:

	A	B
1	=db.query("select * from Orders")	=A1.select(year(Date)==2019&&month(Date)==1)
2	=db.query("select * from Product")	=A2.select(like(Name, "*water*"))
3	=db.query("select * from Detail")	=A3.switch @i(ProductID,B2:ProductID)
4	=B3.group(ID)	=A4.select(sum(Amount)>=200)
5	=db.query("select * from Payment")	=A3.select(Instalments==0)
6	=db.query("select * from Evaluation")	=A4.select(Score==5)
7	=join(B1:Orders,ID;B4:Detail,ID;B5:Payment, ID;B6:Evaluation,ID)	

04

Cross Join

4.1 Non-Equijoin

Please list the age groups of community residents.

Community

ID	Name	Age
1	David	28
2	Daniel	15
3	Andrew	65

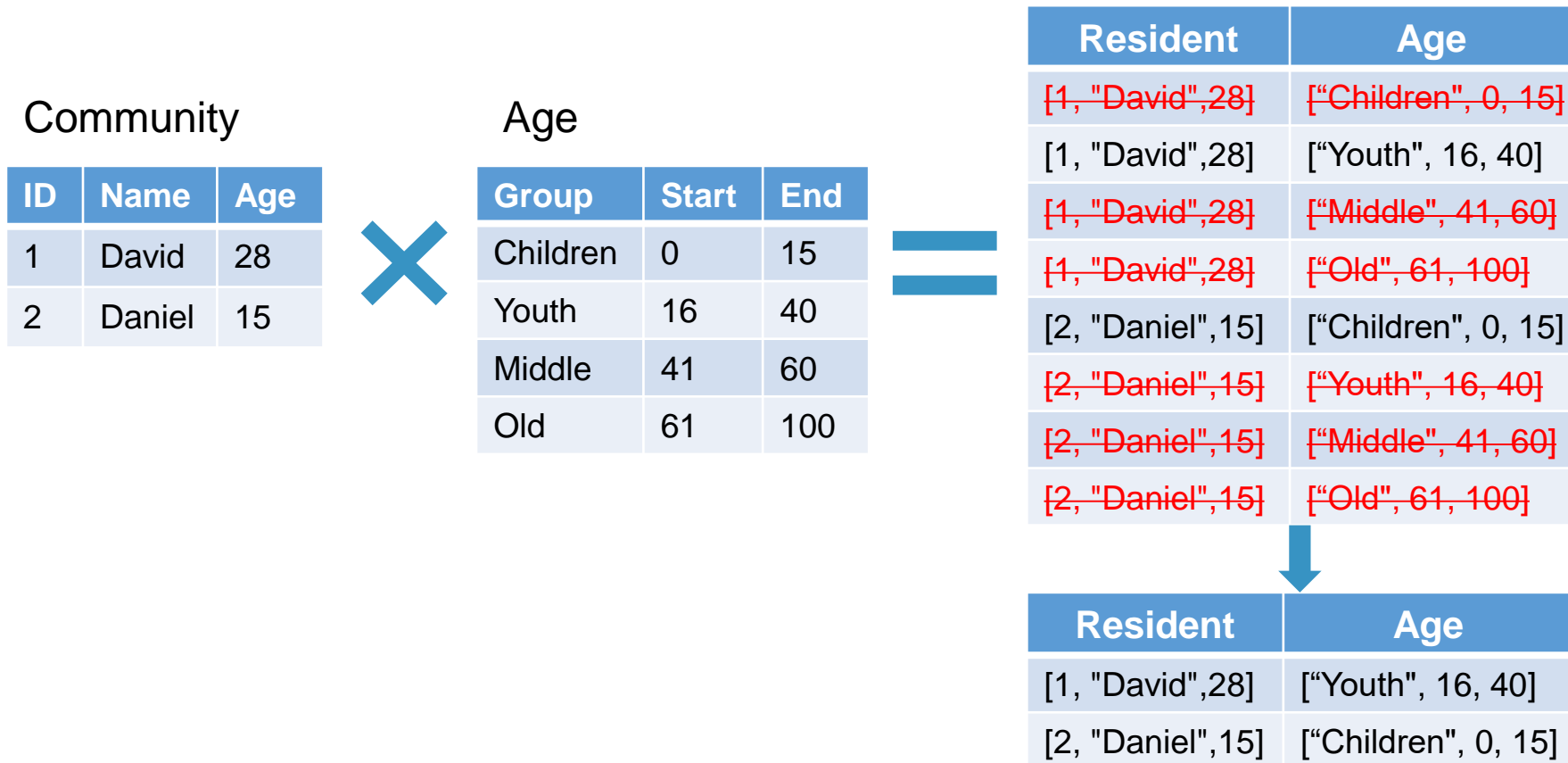
Age

Group	Start	End
Children	0	15
Youth	16	40
Middle	41	60
Old	61	100

There is no direct correlation between the two tables. We need to multiply the two tables by cross and then compare the age groups according to the starting and ending ages.

Cross Join

The two tables were cross multiplied and filtered according to the age range (starting = < age <= ending).



The complete SPL is as follows:

	A	B
1	<code>=db.query("select * from Community")</code>	<code>=db.query("select * from Age")</code>
2	<code>=xjoin(A1:Resident; B1:Age, B1.Start<=A1.Age && B1.End>=A1.Age)</code>	<code>=A2.new(Resident.ID:ID, Resident.Name:Name,Resident.Age:Age,A ge.Grouop:Group)</code>

The xjoin function supports filtering conditions and can filter during join.

4.2 Equijoin

The data structure of the matrix table is as follows. The product of two matrices is needed.

Matrix	
row	
col	
value	

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Here we assume that in matrix A*B, the number of columns in matrix A equals the number of rows in matrix B.

1. Two matrix tables are cross multiplied and filtered at the same time (records with column numbers of A equal to row numbers of B are selected).

MatrixA

row	col	value
1	1	1
1	2	2
1	3	3
2	1	4
2	2	5
2	3	6



MatrixB

row	col	value
1	1	1
1	2	4
2	1	2
2	2	5
3	1	3
3	2	6



A	B
[1, 1, 1]	[1, 1, 1]
[1, 1, 1]	[1, 2, 4]
[1, 1, 1]	[2, 1, 2]
[1, 1, 1]	[2, 2, 5]
[1, 1, 1]	[3, 1, 3]
[1, 1, 1]	[3, 2, 6]
...	...

2. Sum up the product of values.

row	col	sum(A.value*B.value)
1	1	14
1	2	32
2	1	32
2	2	77

The complete SPL is as follows:

	A	B
1	=db.query("select * from MatrixA")	=db.query("select * from MatrixB")
2	=xjoin(A1:A; B1:B,A1.col==B1.row)	=A2.groups(A.row,B.col;sum(A.value * B.value))

The mathematical formula for this example is as follows:

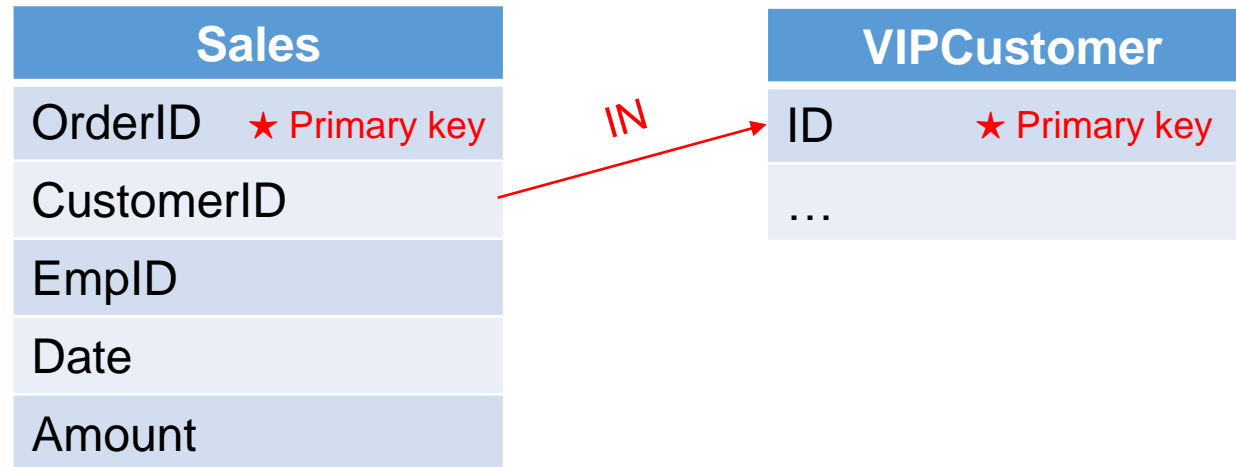
$$C = AB = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} = \begin{pmatrix} 1 \times 1 + 2 \times 2 + 3 \times 3 & 1 \times 4 + 2 \times 5 + 3 \times 6 \\ 4 \times 1 + 5 \times 2 + 6 \times 3 & 4 \times 4 + 5 \times 5 + 6 \times 6 \end{pmatrix} = \begin{pmatrix} 14 & 32 \\ 32 & 77 \end{pmatrix}$$

05

Converting SQL subqueries into JOIN

5.1 IN subquery

Query the sales information of VIP customers.



The SQL statement is as follows:

```
select * from Sales where CustomerID in ( select ID from VIPCustomer )
```

Converting SQL subqueries into JOIN

Sales

OrderID	CustomerID	EmpID	Date	Amout	Join filtering	VIPCustomer
4	VINET	2	2018-07-04	2440		CENTC
2	CENTC	1	2018-07-05	3730	→	KNEOE
3	OTTIK	5	2018-07-08	1863		...
...		

In the VIP customer table, the customer ID is unique, and the two tables can join by the customer ID. When the two tables join, the non-corresponding customers (non-VIP customers) need to be deleted.

Converting SQL subqueries into JOIN

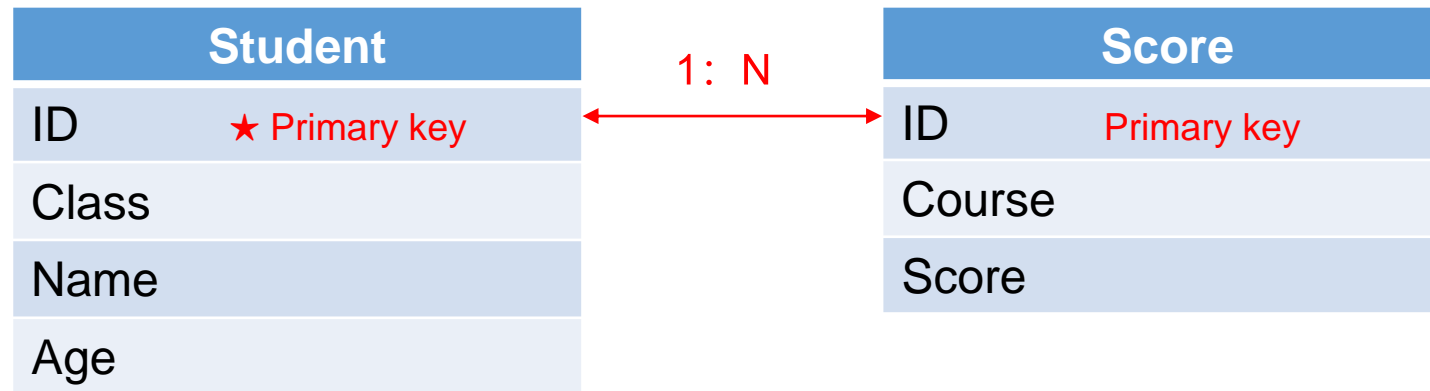
The complete SPL is as follows:

	A	B
1	<code>=db.query("select * from Sales")</code>	<code>=db.query("select * from VIPCustomer")</code>
2	<code>=A1.join@i(CustomerID, B1:ID)</code>	

A2: The join@i option is used to delete the record when the corresponding value is not found for join filtering.
If you need to use foreign key table objects after join, you can use switch@i to objectify foreign keys and filter.

5.2 NOT IN subquery

Look for students who don't have exam results.



The SQL statement is as follows:

```
select * from Student t1 where ID not in ( select ID from Score t2 where t1.ID = t2.ID )
```

Converting SQL subqueries into JOIN



Student ID is not unique in the student's score table, and can not join directly. The student ID needs to be grouped and de-duplicated first, and then the student ID is equivalent to the primary key, so join can be performed.

Converting SQL subqueries into JOIN

The complete SPL is as follows:

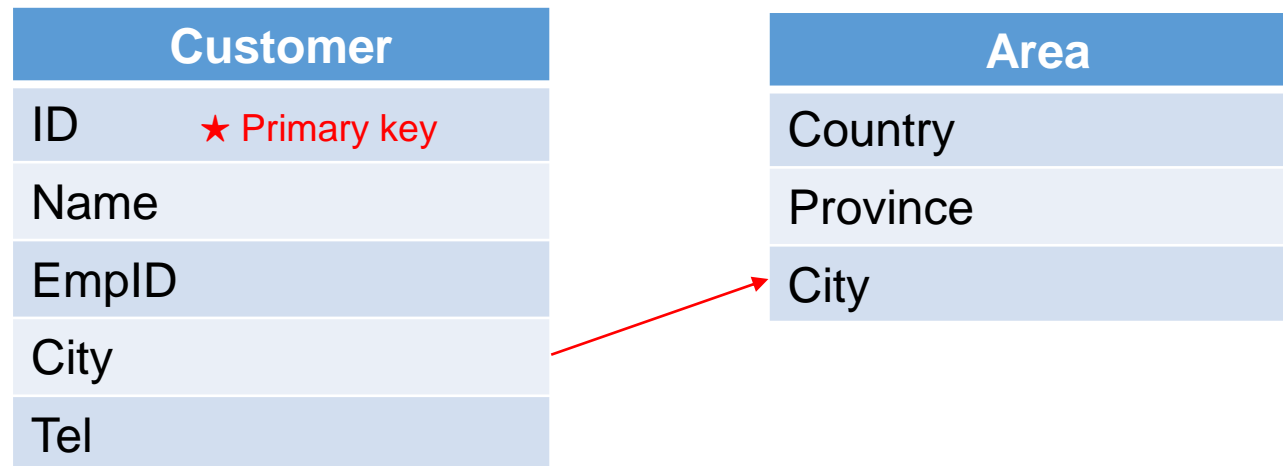
	A	B
1	=db.query("select * from Student")	=db.query("select * from Score")
2	=B1.group@1(ID)	=A1.join@d(ID, A2:ID)

A2: The join@d option is used to keep the record (as opposed to the i option) when the corresponding value is not found for join filtering.

In IN and EXISTS statements, before joining, it is necessary to determine whether the join field is actually the primary key in the sub-query (which is not the primary key field, and may become the de facto primary key after conditional filtering). If it is not the primary key, you need to group according to the specified field before join.

5.3 EXISTS subquery

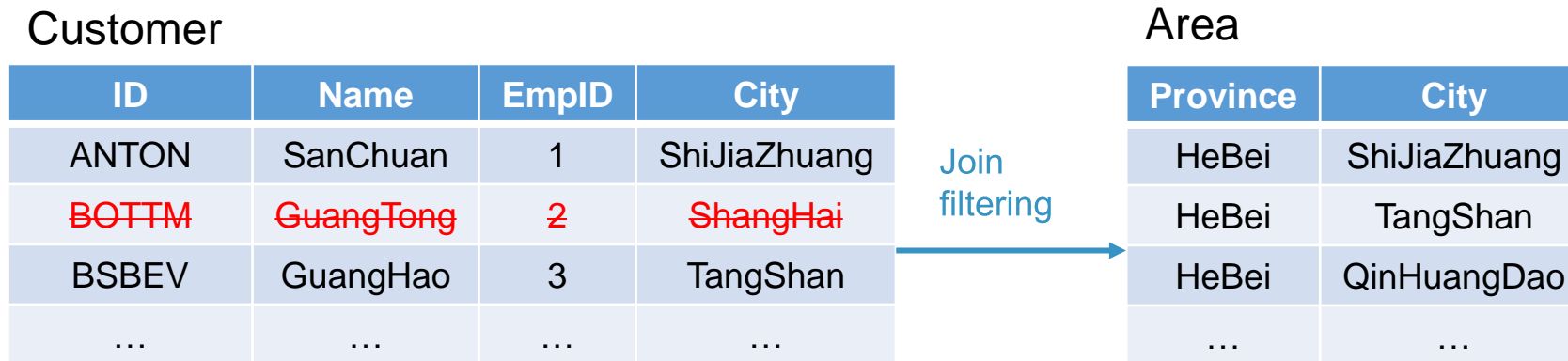
Example 1: Query the customer information of Hebei Province.



The SQL statement is as follows:

```
select * from Customer t1 where exists ( select City from Area t2 where Province = 'HeBei'
and t1.City=t2.City)
```

Converting SQL subqueries into JOIN



In the sub-query where conditions, besides the city field, the join field also has a constant condition: province = Hebei. For EXISTS sub-queries with constant conditions, we can consider them as multi-field joins (province + cities).

Converting SQL subqueries into JOIN

The complete SPL is as follows:

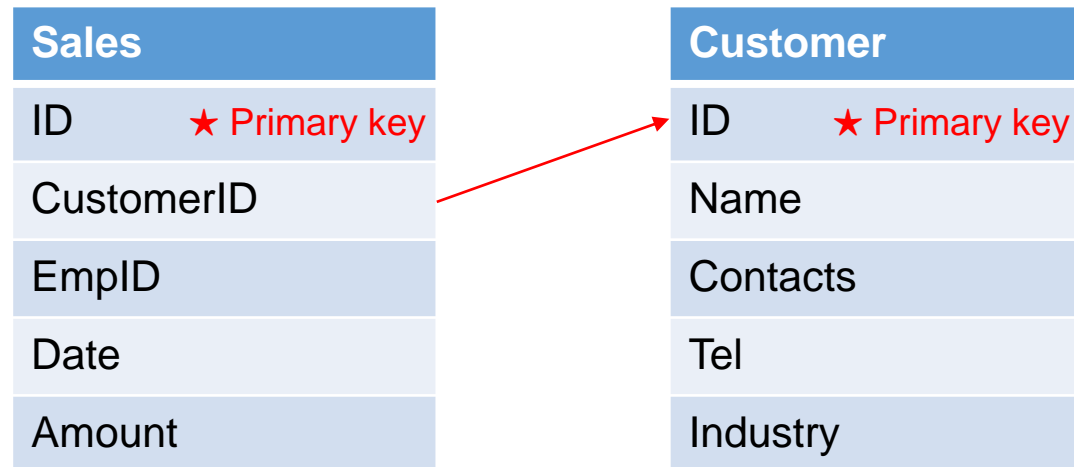
	A	B
1	<code>=db.query("select * from Customer")</code>	<code>=db.query("select * from Area")</code>
2	<code>=A1.join@i("HeBei":City, A2:Province:City)</code>	

If there is no such constant condition (province = Hebei), `join@i()` join filtering can be used directly (it may need to be grouped and de-duplicated first).

In addition, except for null values and constants, IN sub-queries can be converted to EXISTS sub-queries, and SPL processes the same for both.

5.4 NOT EXISTS subquery

Search for new Internet customers (customers not included in the customer list).

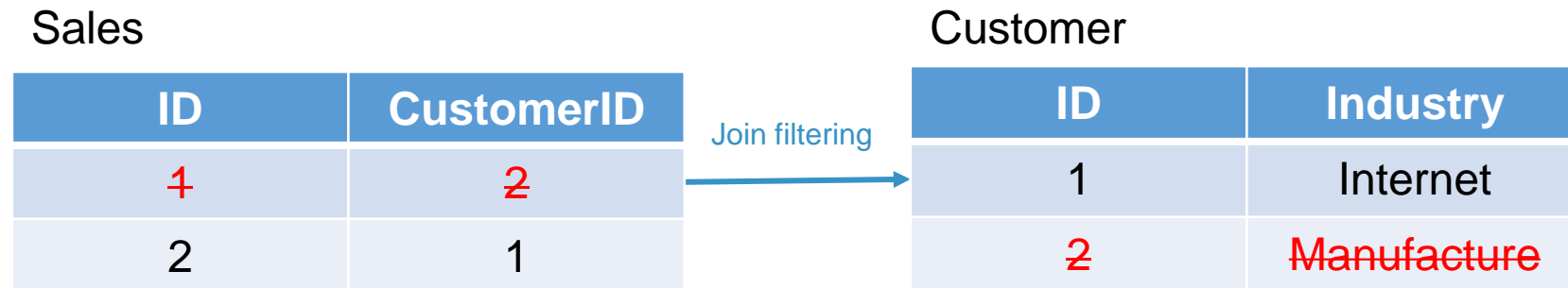


The SQL statement is as follows:

```
select * from Sales t1 where not exists ( select * from Customer t2 where t1.CustomerID=t2.  
ID and Industry = 'Internet' )
```

Converting SQL subqueries into JOIN

Join filtering, customers that cannot be found in the customer table will be deleted.

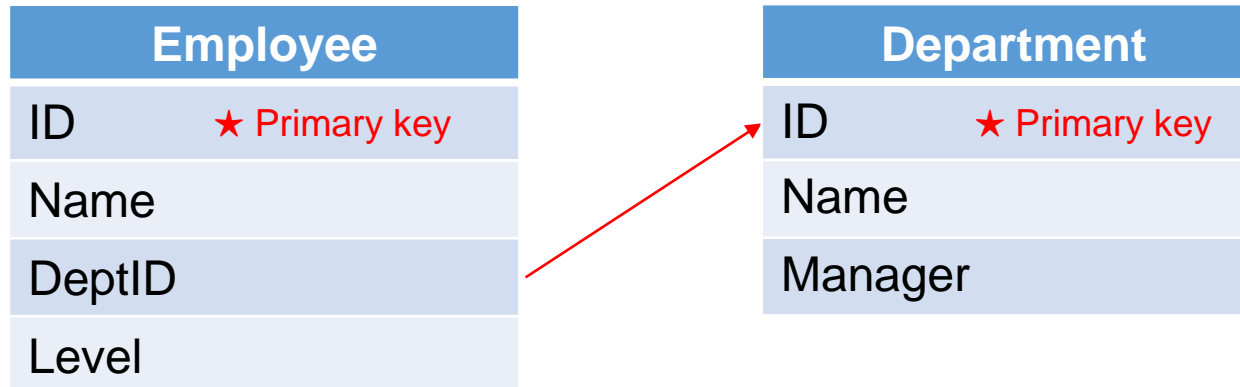


The complete SPL is as follows:

	A	B
1	=db.query("select * from Sales")	=db.query("select * from Customer")
2	=B1.select(Industry="Internet")	=A1.join@i(CustomerID, A2:ID)

5.5 SELECT subquery

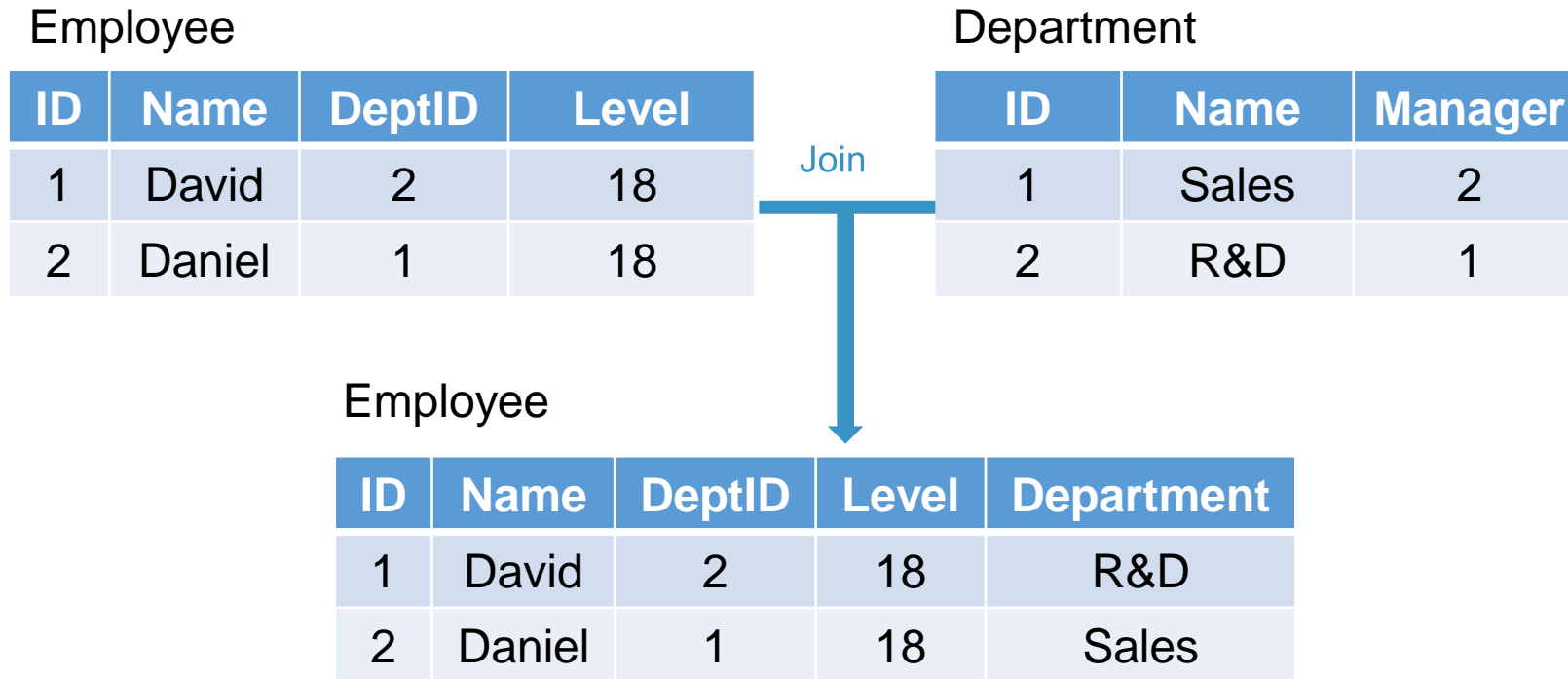
Query the Department name of the employee.



The SQL statement is as follows:

```
select ID, Name, ( select Name from Department where Employee.DeptID=ID ) Department
from Employee
```

Converting SQL subqueries into JOIN



Observing where condition in select sub-query, department ID is used as join condition, which is equivalent to foreign key join and direct join.

Converting SQL subqueries into JOIN

The complete SPL is as follows:

	A	B
1	<code>=db.query("select * from Employee")</code>	<code>=db.query("select * from Department")</code>
2	<code>=A1.join(DeptID, B1:ID, Department)</code>	

5.6 WHERE subquery

Look for employees whose post salaries are higher than 8000.



The SQL statement is as follows:

```
select * from Employee t1 where ( select PostSalary from PostSalary t2 where t1. Level=t2.Level) > 8000
```

Converting SQL subqueries into JOIN

Employee

ID	Name	DeptID	Level
1	David	2	18
2	Daniel	1	18
3	Andrew	4	16

Join
filtering

PostSalary

Level	Salary	Allowance
16	5000	1000
17	7000	2000
18	9000	3000

In the subquery where condition, only the post level can be used as the join field, which can be directly joined and filtered.

Converting SQL subqueries into JOIN

The complete SPL is as follows:

	A
1	=db.query("select * from Employee")
2	=db.query("select * from PostSalary")
3	=A2.select(Salary>8000)
4	=A1.join@i(Level, A3:Level)

SELECT and WHERE subqueries are similar, always using the conditions of the main table, which is equivalent to foreign key join. Multiple conditions are equivalent to multiple field foreign key join.



THANKS