An Introduction to Database System

Chap 7. Relational Algebra

spark@dblab.sogang.ac.kr

7.1 Introduction (1/6)

- Relational Algebra
  - consists of collection of operators, such as join, that take relations as their operands and return relations as their result
- "original" algebra
  - Codd's eight operators

spark@dblab.sogang.ac.kr

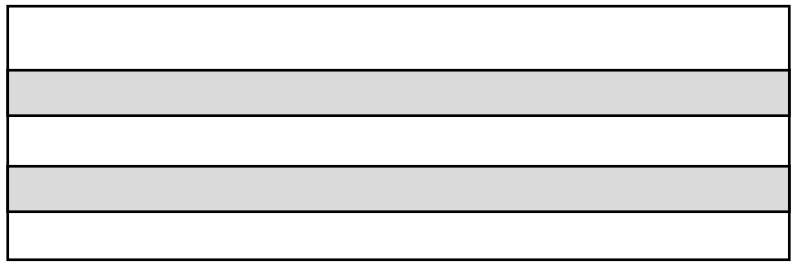
7.1 Introduction (2/6)
An Overview of the Original Algebra

- the relational algebra as defined by Codd
  - the traditional set operations
    - union, intersection, difference, and Cartesian product
  - the special relational operations
    - restrict, project, join, and divide
- Remarks
  - All operators apply to all relations: generic operators
  - Read-only
- Expressive power
  - basis
    - relational algebra
    - relational calculus

spark@dblab.sogang.ac.kr

7.1 Introduction (3/6)
An Overview of the Original Algebra

- simple definition of operators
  - RESTRICT :
    - all tuples from a specified relation that satisfy a specified condition



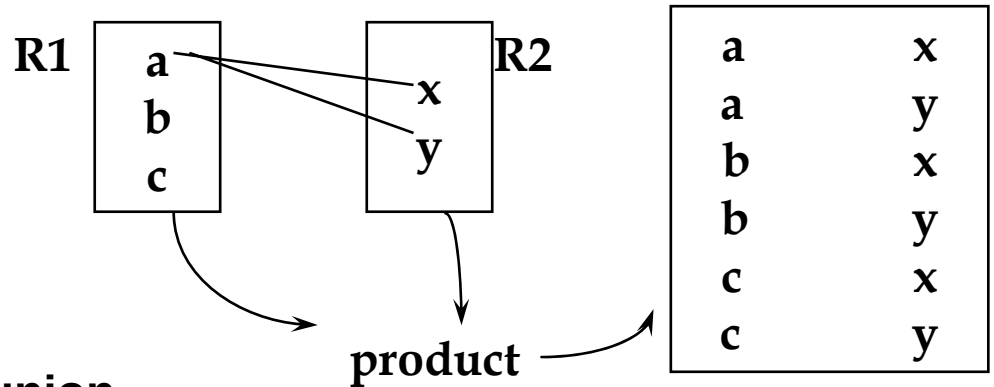
PROJECT :

- all tuples that remains as subtuples in a specified relation after specified attributes have been eliminated

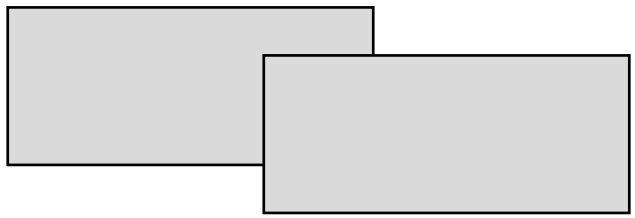
spark@dblab.sogang.ac.kr

7.1 Introduction (4/6)
An Overview of the Original Algebra

- simple definition of operators
  - product (Cartesian product)
    - combination of two tuples



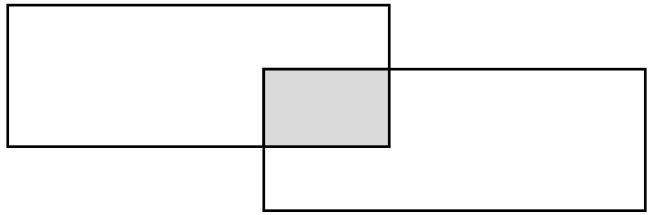
  - union



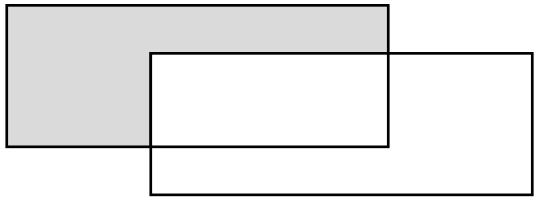
spark@dblab.sogang.ac.kr

7.1 Introduction (5/6)
An Overview of the Original Algebra

- simple definition of operator
  - intersect



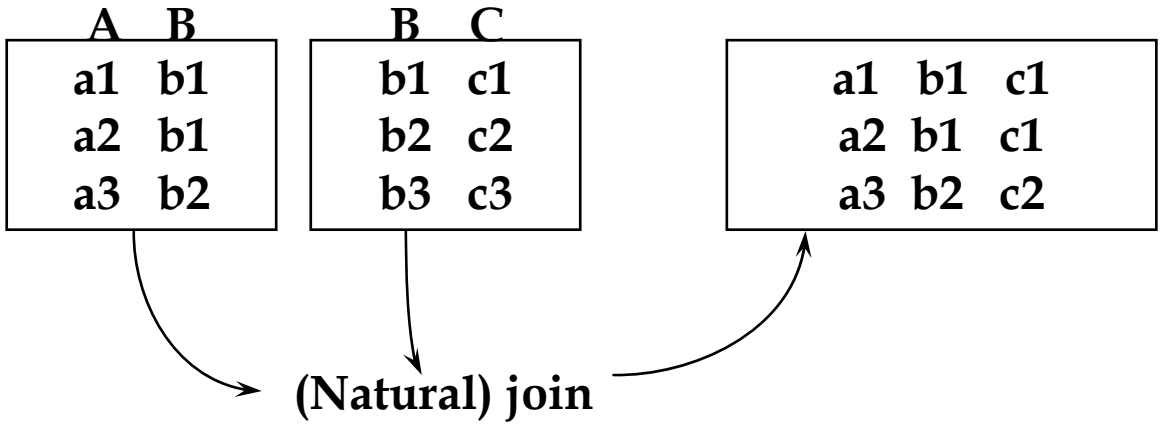
  - difference



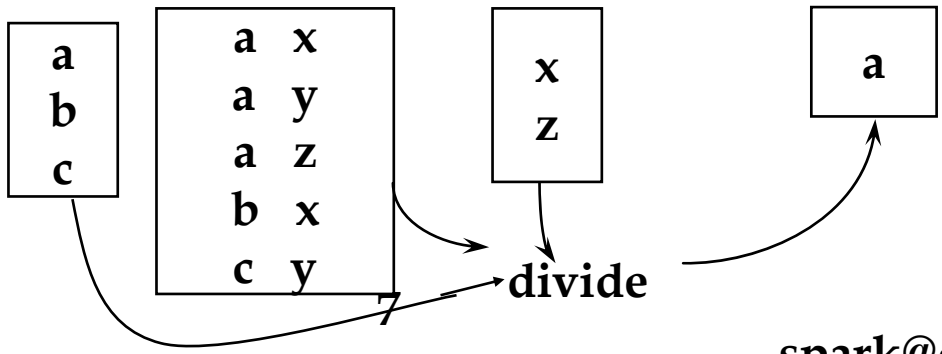
spark@dblab.sogang.ac.kr

7.1 Introduction (6/6)
An Overview of the Original Algebra

- simple definition of operator
  - join



  - divide



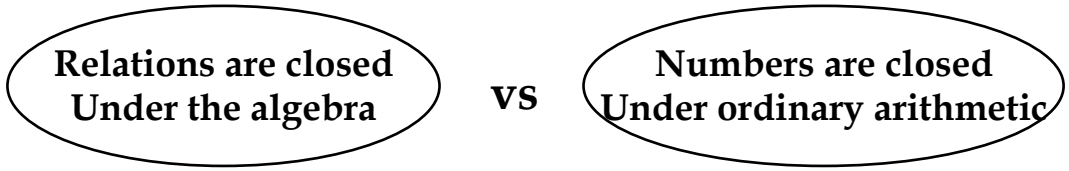
7.2 Closure Revised(1/4)

- ***closure* property**
  - **output of any operation is the same kind of object(relations) as the input**
  - **the output from one operation can become input to another**
  - **possible to make *nested expression***

spark@dblab.sogang.ac.kr

7.2 Closure Revised(2/4)

- An analogy between the ability to nested relational expressions in relational algebra and the ability to nest arithmetic expressions in ordinary arithmetic



- Every relation has two parts, a heading and a body
  - A heading for a base relation - well known to the system
  - What about derived relation?

spark@dblab.sogang.ac.kr

7.2 Closure Revised(3/4)

ex) **S JOIN P**

- **We know what the body of the result looks like – but what about heading ?**
- **That result must be of a well-defined relation type**

ex) **(S JOIN P) WHERE CITY = 'Athens'**

- **Relation-type inference rules**
  - If we know the type(s) of the input relation(s) for any given relational operation, we can infer the type of the output from that operation
- **RENAME operator**
  - **Takes a given relation and returns another that is identical to the given one except that at least one of its attributes has a different name**

spark@dblab.sogang.ac.kr

7.2 Closure Revised(4/4)

S RENAME CITY AS SCITY

S#	SNAME	STATUS	SCITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

S RENAME PNAME AS PN, WEIGHT AS WT

P#	PN	COLOR	WT	CITY
P1	Nut	Red	12.0	London
P2	Bolt	Green	17.0	Paris
P3	Screw	Blue	17.0	Rome
P4	Screw	Red	14.0	London
P5	Cam	Blue	12.0	Paris
P6	Cog	Red	19.0	London

spark@dblab.sogang.ac.kr

7.3 Syntax(1/3)

□ Concrete syntax for relational algebra expressions

σ : restriction π : projection ∩ : intersection ⋈ : join

<relational expression>

::= RELATION { < tuple expression commalist > }

| <relvar name>

| <relational operation>

| (<relational expression>)

<relational operation>

::= <project> | <nonproject>

<project>

::= <relational expression>

{ [ALL BUT] <attribute name commalist> }

<nonpoject>

::= <rename>

<union> | <intersect> | <minus> | <times> | <restrict> | <join> |

<divide>

spark@dblab.sogang.ac.kr

7.3 Syntax(2/3)

□ Concrete syntax for relational algebra expressions

```
<rename>
        ::=     <relational expression>
                        RENAME <renaming commalist>
<union>
        ::=     <relational expression>
                        UNION <relational expression>
<intersect>
        ::=     <relational expression>
                        INTERSECT <relational expression>
<minus>
        ::=     <relational expression>
                        MINUS <relational expression>
<times>
        ::=     <relational expression>
                        TIMES <relational expression>
```


spark@dblab.sogang.ac.kr

7.3 Syntax(3/3)

- Concrete syntax for relational algebra expressions

```
<restrict>
        ::=     <relational expression> WHERE <boolean expression>

<join>
        ::=     <relational expression>
                        JOIN <relational expression>
<divide>
        ::=     <relational expression>
                        DIVIDEBY <relational expression> PER <per>
<per>
        ::=     <relational expression> | ( <relational expression>,
                                            <relational expression> )
```


spark@dblab.sogang.ac.kr

7.4 Semantics (1/9)

- If the two relations(input) are of the same type, then the result will also be a relation of the same(relation) type. The closure property will be preserved

A

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S4	Clark	20	London

B

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris

Union (A UNION B)

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S4	Clark	20	London
S2	Jones	10	Paris

Intersection (A INTERSECT B)

S#	SNAME	STATUS	CITY
S1	Smith	20	London

Difference (A MINUS B)

S#	SNAME	STATUS	CITY
S4	Clark	20	London

Difference (B MINUS A)

S#	SNAME	STATUS	CITY
S2	Jones	10	Paris

spark@dblab.sogang.ac.kr

7.4 Semantics (2/9)

- If the two relations(input) are of the same type, then the result will also be a relation of the same(relation) type. The closure property will be preserved
- Union
  - Given two relations A and B of the same type, the union A UNION B, is a relation of the same type, with body consisting of all tuples *t* such that *t* appears in A or B or in both
- Intersect
  - Given two relations A and B of the same type, the intersection of those two relations, A INTERSECT B, is a relation of the same type, with body consisting of all tuples *t* such that *t* appears in both A and B
- Difference
  - A MINUS B is a relation of the same type, with body consisting of all types *t* such that *t* appears in A and not in B

spark@dblab.sogang.ac.kr

7.4 Semantics (3/9)

- **Product**
  - **Cartesian product of two relations A and B, A TIMES B, where A and B have no common attribute names, to be a relation with a heading that is the (set theory) union of the headings of A and B and with a body consisting of the set of all tuples *t* such that *t* is the union of a tuple appearing in A and a tuple appearing in B**

(common attributes → RENAME operator)

{A_1 : a_1, A_2 : a_2, …, A_m : a_m}

{B_1 : b_1, B_2 : b_2, …, B_m : b_m}

Product

{A_1 : a_1, …, A_m : a_m, B_1 : b_1, …, B_m : b_m}

- **(n + m) attributes, (i*j) rows**

spark@dblab.sogang.ac.kr

7.4 Semantics (4/9)

- **Restriction**
  - **A WHERE X θ Y**
  - **θ : scalar comparison operator**
    =, ≠, ≤, ≥, <, >
  - **Boolean combination**
    - **A where c1 and c2 = (A where c1) intersect (A where c2)**
    - **A where c1 or c2 = (A where c1) union (A where c2)**
    - **A where NOT c = A minus (A where c)**

S WHERE CITY = 'London'

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S4	Clark	20	London

P WHERE WEIGHT < WEIGHT (14.0)

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12.0	London
P5	Cam	Blue	12.0	Paris

SP WHERE S# = S# ('S6') OR P# = P# ('P7')

S#	P#	QTY

spark@dblab.sogang.ac.kr

7.4 Semantics (5/9)

- **projection**
  - **projection of relation A on X, Y, ..., Z**
  - **A {X, Y, ..., Z}**
  - **duplicate tuples are eliminated**

S { CITY }

CITY
London
Paris
Athens

P { COLOR, CITY }

COLOR	CITY
Red	London
Green	Paris
Blue	Rome
Blue	Paris

(S WHERE CITY = 'Paris') { S# }

S#
S2
S3

spark@dblab.sogang.ac.kr

7.4 Semantics (6/9)

- **natural join**
  - **relation A : { X_1, X_2, ..., X_m, Y_1, Y_2, ..., Y_n}**
  - **relation B : { Y_1, Y_2, ..., Y_n , Z_1, Z_2, ..., Z_p }**
  - **A join B : { X_1, X_2, ..., X_m , Y_1, Y_2, ..., Y_n ,Z_1, Z_2, ..., Z_p }**
  - **if A and B have no common attributes, then A join B is equivalent to A TIMES B.**

S#	SNAME	STATUS	CITY	P#	PNAME	COLOR	WEIGHT
S1	Smith	20	London	P1	Nut	Red	12.0
S1	Smith	20	London	P4	Screw	Red	14.0
S1	Smith	20	London	P6	Cog	Red	19.0
S2	Jones	10	Paris	P2	Bolt	Green	17.0
S2	Jones	10	Paris	P5	Cam	Blue	12.0
S3	Blake	30	Paris	P2	Bolt	Green	17.0
S3	Blake	30	Paris	P5	Cam	Blue	12.0
S4	Clark	20	London	P1	Nut	Red	12.0
S4	Clark	20	London	P4	Screw	Red	14.0
S4	Clark	20	London	P6	Cog	Red	19.0

spark@dblab.sogang.ac.kr

7.4 Semantics (7/9)

- θ - join
  - θ - join of relation A on attribute X with relation B on attribute Y
    - (A TIMES B) where X θ Y
  - corresponding to θ
    - greater than join

```
( ( S RENAME CITY AS SCITY ) TIMES
  ( p RENAME CITY AS PCITY ) )
        WHERE SCITY > PCITY
```

S#	SNAME	STATUS	CITY	P#	PNAME	COLOR	WEIGHT	PCITY
S2	Jones	10	Paris	P1	Nut	Red	12.0	London
S2	Jones	10	Paris	P4	Screw	Red	14.0	London
S2	Jones	10	Paris	P6	Cog	Red	19.0	London
S3	Blake	30	Paris	P1	Nut	Red	12.0	London
S3	Blake	30	Paris	P4	Screw	Red	14.0	London
S3	Blake	30	Paris	P6	Cog	Red	19.0	London

spark@dblab.sogang.ac.kr

7.4 Semantics (8/9)

- **equijoin**
  - include two attributes with the property that the values in those two attributes are equal
  - if one of those two attributes is eliminated, the result is the natural join.
- **natural join is not primitive operation**
  - a projection of a restriction of a product

S join P is equivalent to

((S TIMES (P RENAME CITY AS PCITY)) WHERE CITY=PCITY)

{ ALL BUT PCITY }

{ S#, SNAME, STATUS, CITY, P#, PNAME, COLOR, WEIGHT}

spark@dblab.sogang.ac.kr

7.4 Semantics (9/9)

- **Divide**
  - **division of A by B**
  - **A divideby B**

DEND

S#
S1
S2
S3
S4
S5

MED

S#	P#
S1	P1
S1	P2
S1	P3
S1	P4
S1	P5
S1	P6
...	...

...	...
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4
S4	P5

DOR

P#
P1

DOR

P#
P2
P4

DOR

P#
P1
P2
P3
P4
P5
P6

DEND DIVIDEBY DOR PER MED

S#
S1
S2

S#
S1
S4

S#
S1

spark@dblab.sogang.ac.kr

7.5 Examples

6.5.1 Get supplier names for suppliers who supply part P2.

- **((SP JOIN S) WHERE P# = P# ('P2‘)) {SNAME}**

6.5.2 Get supplier names for suppliers who supply at least one red part.

- **(((P WHERE COLOR = COLOR ('Red’)) {P#} JOIN SP) JOIN S) {SNAME}**

6.5.3 Get supplier names for suppliers who supply all parts.

- **((S {S#} DIVIDEBY P{P#}) PER SP {S#, P#}) JOIN S) {SNAME}**

6.5.4 Get supplier numbers for suppliers who supply at least all those parts supplied by supplier S2.

- **S {S#} DIVIDEBY (SP WHERE S# = S# (‘S2’)) {P#} PER SP {S#, P#}**

6.5.5 Get all pairs of supplier numbers such that the two suppliers concerned are “colocated” (i.e., located in the same city).

- **(((S RENAME S# AS SA) {SA, CITY} JOIN**
  (S RENAME S# AS SB) {SB, CITY})
  WHERE SA < SB) {SA, SB}

6.6.6 Get supplier names for suppliers who do not supply part P2.

- **((S {S#} MINUS (SP WHERE P#=P# ('P2‘)) {S#}) JOIN S) {SNAME}**

spark@dblab.sogang.ac.kr

7.6 What is the Algebra For (1/3)

- primitive operation
  - minimal set of Codd's operations
  - restriction, projection, product, union, difference
  - other three operations
    - join, intersection, divideby
    - can be defined in terms of the other five operations
  - five primitive operations
    - can not be defined in terms of the other four operations
  - A join B
    - (A TIMES B) where X θ Y
  - A intersect B
    - A minus (A minus B)

spark@dblab.sogang.ac.kr

7.7 What is the Algebra For ? (2/3)

- A divideby B,
  - R : set of attributes on A, S : set of attributes on B

A

S#	P#
S1	P1
S1	P2
S1	P3
S2	P1

B

P#
P1
P2
P3

A [R-S] minus ((A[R-S] times B) minus A) [R-S]

S#
S1
S1
S1
S2

S#	P#
S1	P1
S1	P2
S1	P3
S2	P1
S2	P2
S2	P3

S#	P#
S2	P2
S2	P3

S#
S2

Result :

S#
S1

spark@dblab.sogang.ac.kr

7.7 What is the Algebra For ? (3/3)

- The fundamental intent of the algebra : the writing of relational expression
  - defining a scope for retrieval and update
  - defining derived relvars
  - defining security rules
  - defining stability operations
  - defining integrity rules
- The expressions serve as a high-level, symbolic-representation of the user's intent
  - Transformation rules → a basis for optimization

    ((SP JOIN S) WHERE P# = P# ('P2')) {SNAME}

    → ((SP WEHRE P# = P# ('P2')) JOIN S) {SNAME}
- relationally complete
  - if it is at least as powerful as the algebra
  - measure of relational languages

spark@dblab.sogang.ac.kr

7.7 Further Points

- Associativity and Commutativity
  - Associative
    - UNION
      - (A UNION B) UNION C ≡ A UNION (B UNION C)
    - INTERSECT, TIMES, JOIN, but not MINUS
  - Commutative
    - UNION
      - A UNION B ≡ B UNION A
    - INTERSECT, TIMES, JOIN, but not MINUS
- Some Equivalence
- Some Generalization

spark@dblab.sogang.ac.kr

7.8 Additional Operators(1/12)

<semijoin>
::= <relational expression>
SEMIJOIN <relational expression>
<semiminus>
::= <relational expression>
SEMIMINUS <relational expression>
<extend>
::= EXTEND <relational expression>
ADD <extend add commalist>
<extend add>
::= SUMMARIZE <relational expression>
PER <relational expression>
ADD <summarize add commalist>
<summarize add>
::= <summary type> [(<scalar expression>)] AS <attribute name>
<summarize type>
::= COUNT | SUM | AVG | MAX | MIN | ALL | ANY |
COUNTED | SUMD | AVGD
<tclose>
::= TCLOSE <relational expression>

7.8 Additional Operators(2/12)

- **Semijoin**
  - **a SEMIJOIN b ≡ (a join b) { X, Y }**
  - **The semijoin of a with b is the join of a and b, projected over the attributes of a. (the tuples of a that have a counterpart in b)**
  - **EX) Get S#, SNAME, STATUS, CITY for suppliers who supply part P2.**
    - **S SEMIJOIN (SP WHERE P# = P# (‘P2’))**
- **Semidifference**
  - **a SEMIMINUS b ≡ a MINUS (a semijoin b)**
  - **The tuples of a that have no counterpart in b**
  - **EX) Get S#, SNAME, STATUS, CITY for suppliers who do not supply part P2.**
  - **S SEMIMINUS (SP WHERE P# = P# (‘P2’))**

spark@dblab.sogang.ac.kr

7.8 Additional Operators(3/12)

- **Extend**
  - We would like to be able to retrieve the value of an arithmetic expression such as WEIGHT * 454, or to refer to such a value in a WHERE clause
  - EXTEND takes a relation and returns another relation that is identical to the given one except that it includes an additional attribute,
  - EXTEND a ADD exp AS Z
    - A relation defined as follows:
    - - the heading of the result consists of the heading of a extended with attribute Z
    - -the body of the result consists of all tuples t such that t is atuple of a extended with a value for attribute Z, computed by evaluating exp on that tuple of a
    - EX) EXTEND P ADD (WEIGHT * 454) AS GMWT
  - EX) ((EXTEND P ADD (WEIGHT * 454) AS GMWT)
    - WHERE GMWT > WEIGHT (10000.0)) { ALL BUT GMWT }

spark@dblab.sogang.ac.kr

7.8 Additional Operators (4/12)

- **Extend**
  - **EXTEND P ADD (WEIGHT * 454) AS GMWT**
  - **GMWT : additional attribute**
  - **GMWT can be used in projection, restriction, etc.**

P#	PNAME	COLOR	WEIGHT	CITY	GMWT
p1	Nut	Red	12.0	London	5448.0
p2	Bolt	Green	17.0	Paris	7718.0
p3	Screw	Blue	17.0	Rome	7718.0
p4	Screw	Red	14.0	London	6356.0
p5	Cam	Blue	12.0	Paris	5448.0
p6	Cog	Red	19.0	London	8626.0

spark@dblab.sogang.ac.kr

7.8 Additional Operators (5/12)

- **Extend**
  - **Examples**
    1. **EXTEND S ADD 'Supplier' AS TAG**
    2. **EXTEND (P JOIN SP) ADD (WEIGHT * QTY) AS SHIPWT**
    3. **(EXTEND S ADD CITY AS SCITY) { ALL BUT CITY }**
    4. **EXTEND P ADD WEIGHT*454 AS GMWT, WEIGHT*16 AS OZWT**
    5. **EXTEND S ADD COUNT ((SP RENAME S# AS X) WHERE X = S#) AS NP**

S#	SNAME	STATUS	CITY	NP
S1	Smith	20	London	6
S2	Jones	10	Paris	2
S3	Blake	30	Paris	1
S4	Clark	20	London	3
S5	Adams	30	Athens	0

spark@dblab.sogang.ac.kr

7.8 Additional Operators (6/12)

- **Aggregate Operator**
  - To derive a single scalar value from the values appearing in some specified attribute of some specified relation
  - COUNT, SUM, AVG, MAX, MIN, ALL, ANY
  - <op name> (<relational expression> [, <attribute name>])

Ex)

SUM (SP WHERE S# = S# ('S1'), QTY)

SUM ((SP WHERE S# = S# ('S1')) {QTY})

spark@dblab.sogang.ac.kr

7.8 Additional Operators (7/12)

- **Summarize**
  - **extend operator – “horizontal” or row-wise computations**
  - **summarize operator – “vertical” or column-wise computations**

ex)

SUMMARIZE SP PER SP {P#} ADD SUM (QTY) AS TOTQTY

P#	TOTQTY
P1	600
P2	1000
P3	400
P4	500
P5	500
P6	100

spark@dblab.sogang.ac.kr

7.8 Additional Operators(8/12)

- **Summarize**
  - **The extend operator provides a way of incorporating "horizontal" or "tuple-wise" computations into the relational algebra. The summarize operator performs the analogous function for "vertical" or "attribute-wise" computations**
  - **SUMMARIZE SP PER P { P# } ADD SUM (QTY) AS TOTQTY**
    - **SP is conceptually partitioned into groups or set of tuples (one group for each P#), and then each such group is used to generate one tuple in the overall result**
  - **SUMMARIZE a PER b ADD summary AS Z**

spark@dblab.sogang.ac.kr

7.8 Additional Operators (9/12)

- **Summarize**

SUMMARIZE SP PER SP {P#} ADD SUM (QTY) AS TOTQTY

- B must be of the same type as some projection of A

- the heading of the result : A1, A2, …, An, Z

(Let B be A1, A2, …, An)

- the body of the result : all tuples *t* such that *t* is a tuple of B extended with a value for the new attribute Z. The new Z value is computed by evaluating summary over all tuples of A that has the same values for A1, A2, …, An

ex) SUMMARIZE (P JOIN SP) PER P {CITY} ADD COUNT AS NSP

CITY	NSP
London	5
Paris	6
Rome	1

spark@dblab.sogang.ac.kr

7.8 Additional Operators (10/12)

- **Points**
  1. **multiple SUMMARIZEs**

```
SUMMARIZE SP PER P {P#} ADD SUM (QTY) AS TOTQTY,
                        AVG (QTY) AS AVGQTY
```

  2. **general from**

```
SUMMARIZE <relational expression>
          PER <relational expression>
              ADD <summarize add commalist>
<summarize add>
          <summary type> [ ( <scalar expression> ) ] AS <attribute name>
Summary type : COUNT, SUM, AVG, MAX, MIN, ANY, ALL,
               COUNTD, SUMD, AVGD (D-distinct)
```


spark@dblab.sogang.ac.kr

7.8 Additional Operators (11/12)

- **Points**

3. **not primitive operator**

```
SUMMARIZE SP PER S {S#} ADD COUNT AS NP
≡( EXTEND S {S#} ADD ( ( SP RENAME S# AS X ) WHERE X=S# ) AS Y,
                        COUNT ( Y ) AS NP ) {S#, P#}
or equivalently :
WITH ( S {S#} ) AS T1,
        ( SP RENAME S# AS X ) AS T2,
        ( EXTNED T1 ADD ( T2 WHERE X = S# ) AS Y ) AS T3,
        ( EXTEND T3 ADD COUNT ( Y ) AS NP ) AS T4 :
T4 {S#, P#}
```

4. **per a relation that has no attributes at all**

```
SUMMARIZE SP PER SP{ } ADD SUM (QTY) AS GRANDTOTAL
SUMMARIZE SP ADD SUM (QTY) AS GRANDTOTAL
```


spark@dblab.sogang.ac.kr

7.8 Additional Operators(12/12)

- **Tclose**
  - **Let A be a binary relation with X and Y, both of the same type T.**
  - **The transitive closure of A, TCLOSE A is a relation A^+ with heading the same as that of A and body a superset of that of A.**
  - **The tuple { X : x, Y : y } appears in A^+ iff it appears in A or there exists a sequence of values z1, z2, …, zn, all of type T, such that the tuples { X : x, Y : z1 }, { X : z1, Y : z2 }, … , { X : zn , Y : y } all appear in A.**

spark@dblab.sogang.ac.kr

7.9 Grouping and Ungrouping(1/3)

- **The fact that we can relations with attributes whose values are relations in turn leads to the desirability of certain additional relational operators → group, ungroup**

ex) SP GROUP (P#, QTY) AS PQ

(group SP by S#)

S#	PQ
S1	P# / QTY P1 300 P2 200 P3 400 P4 200 P5 100 P6 100
S2	P# / QTY P1 300 P2 400

S#	PQ
S3	P# / QTY P2 200
S4	P# / QTY P2 200 P4 300 P5 400

Heading : { S# S#, PQ relation {P# P#, QTY QTY}

spark@dblab.sogang.ac.kr

7.9 Grouping and Ungrouping(2/3)

- Ungroup
  - Let SPQ be the relation in Fig. 6.12

    SPQ UNGROUP PQ → SP

    - heading : { S# S#, P# P#, QTY QTY}
- nest, unest capabilities - NF^2 relations
- Reversibility of the GROUP and UNGROUP

two

A	RVX
1	X: a, b
1	X: a, b

three

A	X
1	a
1	b
1	c

one

A	RVX
1	X: a, b, c

spark@dblab.sogang.ac.kr

7.9 Grouping and Ungrouping(3/3)

- If relation r has a relation-valued attribute RVX, then r is reversible ungroupable (w.r.t. RVX) iff the following one both true :
  - No tuple of *r* has an empty relation as its RVX value
  - RVX is functionally dependent on the combination of all other attributes of *r*. (There must be some candidate key of *r* that does not include RVX as a component)

spark@dblab.sogang.ac.kr