Unit 3

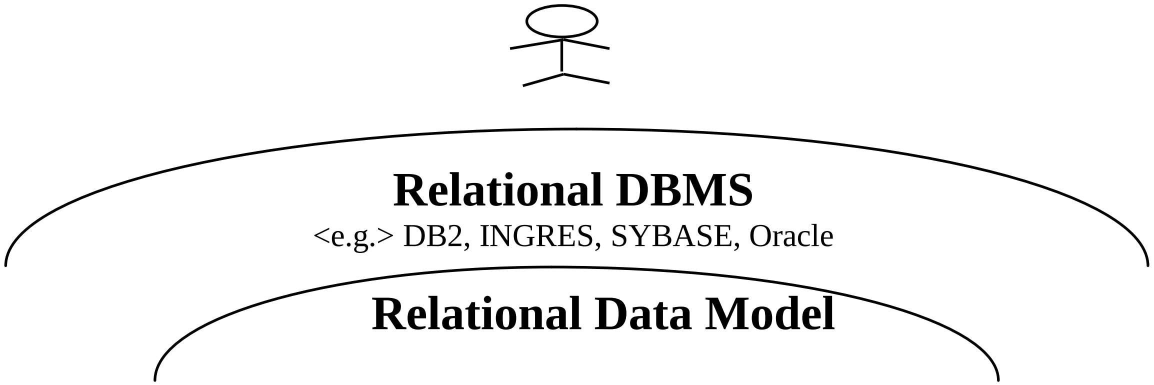
The Relational Model

Outline

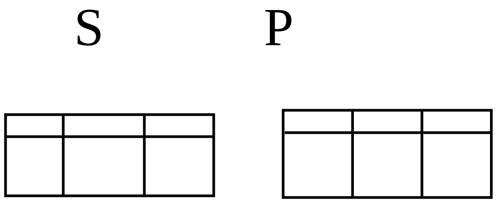
- 3.1 Introduction
- 3.2 Relational Data Structure
- 3.3 Relational Integrity Rules
- 3.4 Relational Algebra
- 3.5 Relational Calculus

3.1 Introduction

Relational Model [Codd '70]



- **A way of** looking at data
- **A prescription for**
  - **representing data**:
    by means of tables
  - **manipulating that representation**:
    by select, join, ...



Relational Model (cont.)

- Concerned with three aspects of data:
  1. Data structure: tables
  2. Data integrity: primary key rule, foreign key rule
  3. Data manipulation: (Relational Operators):
     - Relational Algebra (See Section 3.4)
     - Relational Calculus (See Section 3.5)

- Basic idea: relationship expressed in data values, not in link structure.

<e.g.>

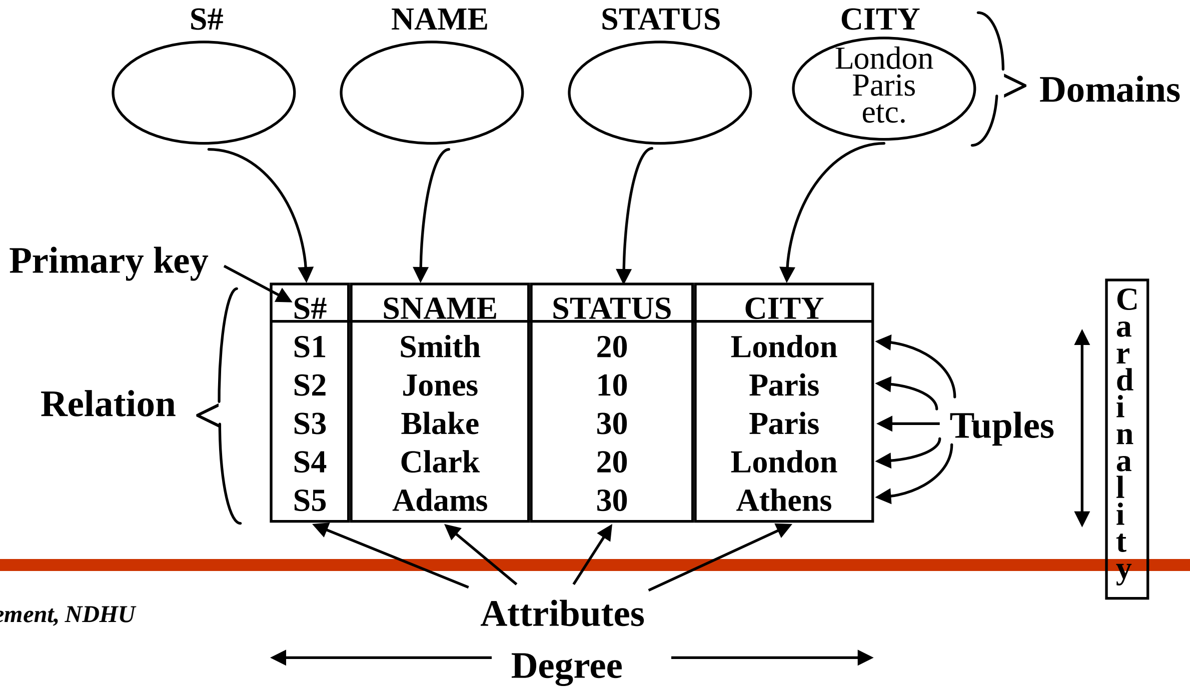
Entity	**Relationship**	**Entity**
Mark	Works_in	Math_Dept

WORKS_IN

Name	Dept
Mark	Math_Dept

Terminologies

- Relation : **so far corresponds to a *table*.**
- Tuple : **a *row* of such a table.**
- Attribute : **a *column* of such a table.**
- Cardinality : **number of tuples.**
- Degree : **number of attributes.**
- Primary key : **an attribute or attribute combination that uniquely identify a tuple.**
- Domain : **a pool of legal values.**



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

3.2 Relational Data Structure

Domain

- **Scalar**: the smallest semantic unit of data, atomic, nondecomposable.
- **Domain**: a set of scalar values with the same type.
- **Domain-Constrained Comparisons**: two attributes defined on the same domain, then comparisons and hence joins, union, etc. will make sense.

  <e.g.>

```
SELECT  P.*,  SP.*              SELECT  P.*, SP.*
FROM     P, SP                  FROM     P, SP
WHERE  P.P#=SP.P#               WHERE   P.Weight=SP.Qty
```

same domain

different domain

- A system that supports domain will prevent users from making silly mistakes.

Domain (cont.)

- Domain should be specified as part of the database definition.

  <e.g.>

```
CREATE              DOMAIN      S#          CHAR(5)
CREATE              DOMAIN      NAME        CHAR(20)
CREATE              DOMAIN      STATUS      SMALLINT;
CREATE              DOMAIN      CITY        CHAR(15)
CREATE              DOMAIN      P#          CHAR(6)
CREATE              TABLE  S
                    (S#       DOMAIN  (S#)  Not Null
                     SNAME   DOMAIN (NAME),
                     .
                     .
CREATE              TABLE  P
                    (P#                 DOMAIN (P#)  Not Null,
                     PNAME              DOMAIN (NAME).
                     .
                     .
CREATE              TABLE  SP
                    (S#     DOMAIN  (S#)  Not Null,
                     P#     DOMAIN  (P#)  Not Null,
```

- **Composite domains**: a combination of simple domains.

<e.g.> DATE = MONTH(1..12) + DAY(1..31) +YEAR(0..9999)

```
CREATE              DOMAIN      MONTH       CHAR(2);
CREATE              DOMAIN      DAY         CHAR(2);
CREATE              DOMAIN      YEAR        CHAR(4);
CREATE              DOMAIN      DATE
                    (MONTH      DOMAIN      (MONTH),
                     DAY        DOMAIN      (DAY),
                     YEAR     DOMAIN      (YEAR));
```

Relations

- Definition : **A relation on domains D_1, D_2, ..., D_n (not necessarily all distinct) consists of a *heading* and a *body*.**

heading

body

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

- ***Heading :*** a fixed set of attributes $A_1,....,A_n$ such that A_j underlying domain D_j (j=1...n) .
- ***Body:*** a time-varying set of tuples.
- ***Tuple:*** a set of attribute-value pairs.

$$\{A_1:Vi_1,\ A_2:Vi_2,...,\ A_n:Vi_n\}, \text{ where } I = 1...m$$

or

$$\left\{t_1, t_2, t_3, \ldots t_m\right\}$$

Properties of Relations

- There are no duplicate tuples: since **relation** is a **mathematical set.**
  - ***Corollary*** : the primary key always exists.

    (at least the combination of all attributes of the relation has the uniqueness property.)
- Tuples are unordered.
- Attributes are unordered.
- All attribute values are atomic.

  i.e. There is only one value, not a list of values at every row-and-column position within the table.

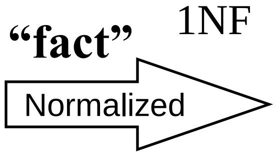
  i.e. Relations do not contain repeating groups.

  i.e. Relations are ***normalized***.

Properties of Relations (cont.)

- Normalization

S#	PQ
S1	{ (P1,300), (P2, 200), (P3, 400), (P4, 200), (P5, 100), (P6, 100) }
S2	{ (P1, 300), (P2, 400) }
S3	{ (P2, 200) }
S4	{ (P2, 200), (P4, 300), (P5, 400) }


“fact”

1NF

Normalized

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

- **degree** : 2
- **domains**:

S# = {S1, S2, S3, S4}

PQ = {<p,q> | p∈{P1, P2, ..., P6}

q ∈{x| 0≤ x ≤1000}}

- **a mathematical relation**

- **degree**: 3
- **domains**:

S# = {S1, S2, S3, S4}

P# = {P1, P2, ..., P6}

QTY = {x| 0≤ x ≤1000}}

- **a mathematical relation**

Properties of Relations (cont.)

- **Reason for normalizing a relation :** ***Simplicity!!***

<e.g.> Consider two transactions T1, T2:

Transaction T1 : insert ('S5', 'P6' , 500)

Transaction T2 : insert ('S4', 'P6', 500)

There are difference:

- Un-normalized: two operations (one insert, one append)
- Normalized: one operation (insert)

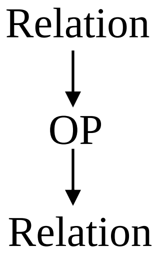
Kinds of Relations

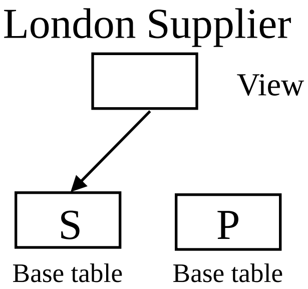
- **Base Relations (Real Relations):** a named, atomic relation; a direct part of the database. e.g. S, P
- **Views (Virtual Relations):** a named, derived relation; purely represented by its definition in terms of other named relations.
- **Snapshots:** a named, derived relation with its ***own stored data.***

  <e.g.>

```
CREATE  SNAPSHOT  SC
AS  SELECT S#,  CITY
FROM    S
REFRESH  EVERY  DAY;
```

  - A read-only relation.
  - Periodically refreshed

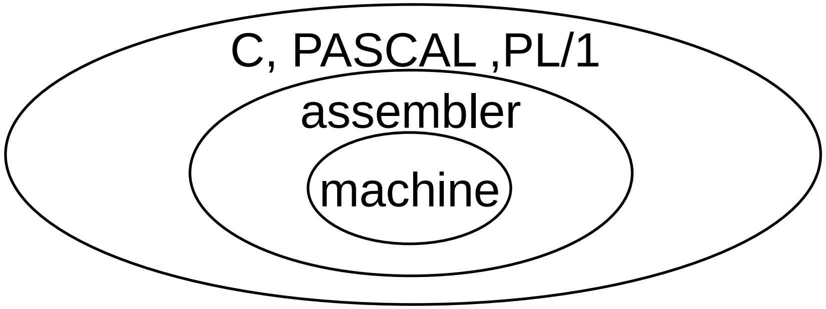


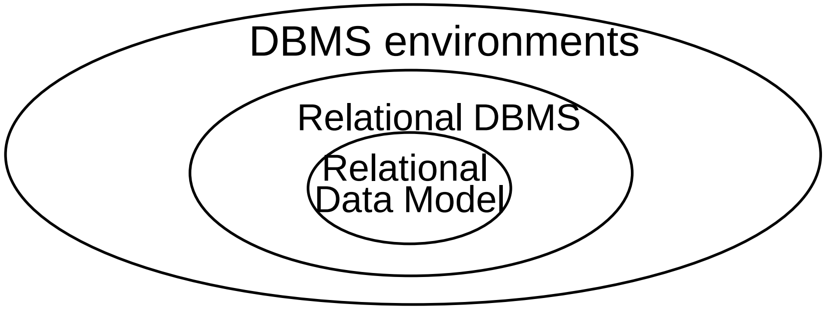


- **Query Results:** may or may not be named, no persistent existence within the database.
- **Intermediate Results:** result of subquery, typically unnamed.
- **Temporary Relations:** a named relation, automatically destroyed at some appropriate time.

Relational Databases

- Definition: A **Relational Database** is a database that is perceived by the users as a collection of *time-varying*, *normalized* **relations.**
  - ***Perceived by the users:*** the relational model apply at the **external** and **conceptual** levels.
  - ***Time-varying:*** the set of tuples changes with time.
  - ***Normalized:*** contains no repeating group (only contains atomic value).
- The **relational model** represents a database system at a level of abstraction that removed from the details of the underlying machine, like **high-level language.**





3.3 Relational Integrity Rules

Purpose:

*to inform the DBMS of certain constraints
in the real world.*

Keys

- Candidate keys: Let R be a relation with attributes $A_1, A_2, ..., A_n$.
  The set of attributes K $(A_i, A_j, ..., A_m)$
  of R is said to be a candidate key iff it satisfies:
  - ***<u>Uniqueness</u>:*** At any time, no two tuples of R have the same value for K.
  - ***<u>Minimum</u>:*** none of $A_i, A_j, ... A_k$ can be discarded from K without destroying the uniqueness property.

    <e.g.> S# in S is a candidate key.
    (S#, P#) in SP is a candidate key.
    (S#, CITY) in S is not a candidate key.

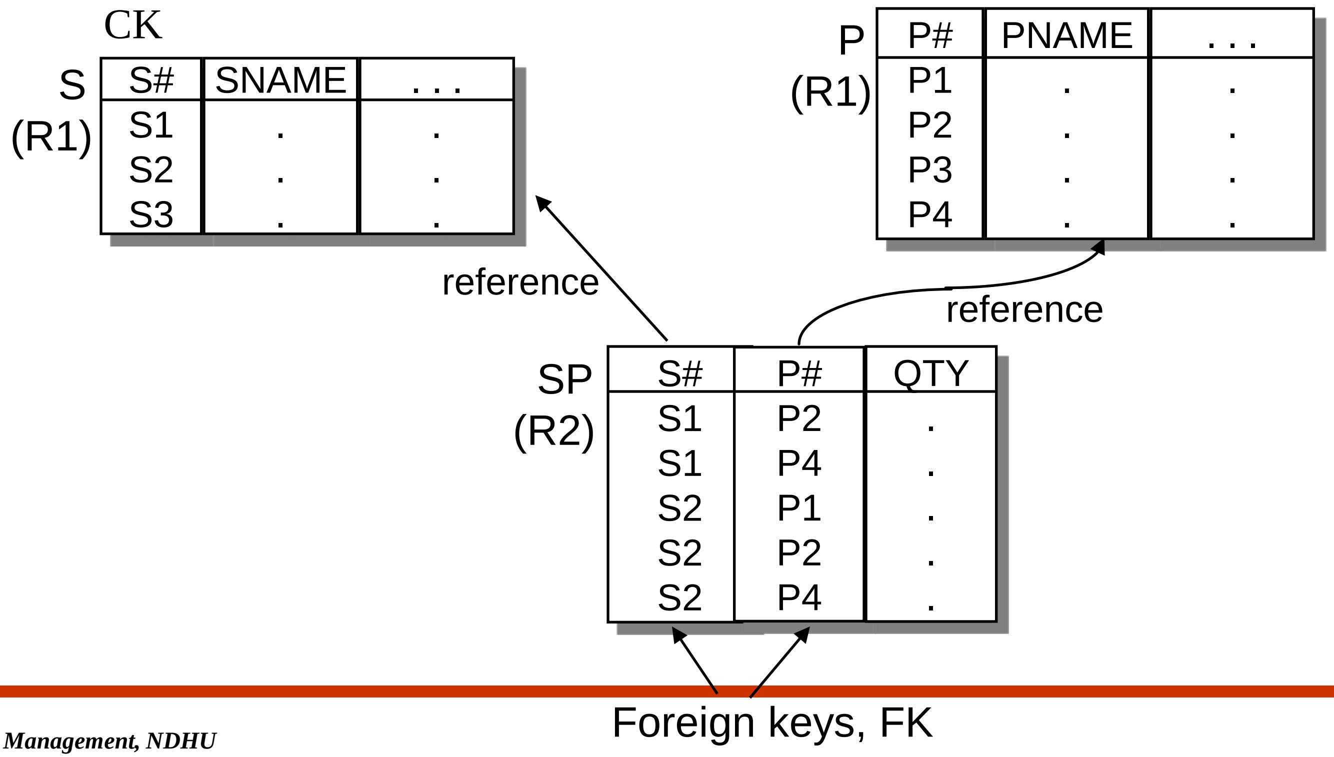
S#	SNAME	STATUS	CITY
S1	Smith	20	London
S4	Clark	20	London

- **Primary key**: one of the candidate keys.
- **Alternate keys:** candidate keys which are not the primary key.

  <e.g.> S#, SNAME: both are candidate keys
  S#: primary key
  SNAME: alternate key.
- **Note:** Every relation has at least one candidate key.

Foreign keys (p.261 of C. J . Date)

- ***Foreign keys*: ** Attribute FK (possibly composite) of base relation R2 is a foreign keys iff it satisfies:
  - 1. There exists a base relation R1 with a candidate key CK, and
  - 2. For all time, each value of FK is identical to the value of CK in some tuple in the current value of R1.



S (R1)

S#	SNAME	. . .
S1	.	.
S2	.	.
S3	.	.

P (R1)

P#	PNAME	. . .
P1	.	.
P2	.	.
P3	.	.
P4	.	.

SP (R2)

S#	P#	QTY
S1	P2	.
S1	P4	.
S2	P1	.
S2	P2	.
S2	P4	.

Two Integrity Rules of Relational Model

- **Rule 1: Entity Integrity Rule**

  No component of the primary key of a base relation is allowed to accept nulls.

- **Rule 2: Referential Integrity Rule**

  The database must not contain any unmatched foreign key values.

Note: *Additional rules which is specific to the database can be given.*

<e.g.> QTY = { 0~1000}

However, they are outside the scope of the relational model.

Referential Integrity Rule

How to avoid against the referential Integrity Rule?

- Delete rule: **what should happen on an attempt to delete/update target of a foreign key reference**
  - *RESTRICTED*
  - *CASCADES*
  - *NULLIFIES*

  <e.g.> **User issues:**

  DELETE FROM S WHERE S#='S1'

  System performs:

  Restricted:

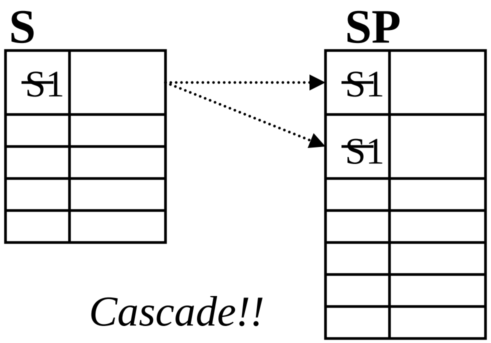
  Reject!

  Cascades:

  DELETE FROM **SP** WHERE S#='S1'

  Nullifies:

  UPDATE **SP** SET S#=Null WHERE S#='S1'



Foreign Key Statement

- Descriptive statements:

```
FOREIGN KEY (foreign key) REFERENCES target
              NULLS  [NOT] ALLOWED
              DELETE OF target effect
              UPDATE OF target-primary-key effect;
```

effect: one of {RESTRICTED, CASCADES, NULLIFIES}

<e.g.1> (p.269)

```
CREATE TABLE SP
(S# S# NOT NULL, P# P# NOT NULL,
QTY QTY NOT NULL,
 PRIMARY KEY (S#, P#),
 FOREIGN KEY (S#) REFERENCE S
                  ON DELETE CASCADE
                  ON UPDATE CASCADE,
 FOREIGN KEY (P#) REFERENCE P
                  ON DELETE CASCADE
                  ON UPDATE CASCADE,
 CHECK (QTY>0 AND QTY<5001));
```

3.4 Relational Algebra

Introduction to Relational Algebra

- The relational algebra consists of a collection of eight high-level operators that **operate on relations**.
- Each operator takes relations (one or two) as operands and produce a relation as result.
  - the important property of **closure.**
  - nested relational expression is possible.

<e.g.> $R3 = \sigma(R1 \bowtie R2)$

$T_1 \leftarrow R_1$ join R_2

$R_3 \leftarrow T_1$ selection

Integer

{I; +, -, *}

$\uparrow$

objects

{{0,1,2,3},+}

+	0	1	2	3
0	0	1	2	3
1	1	2	3	4
2	2	3	4	5
3	3	4	5	6

NOT Closure!

$(OP_2(OP_1(A))\ OP_3\ B)$

{relations; OP1, OP2, ..., OP8}

2-3 = -1 $\notin$ N not closure!

N = {1,2,3,....}

⊕	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	1	0	2

Closure!

1+2 = 3 $\in$ N

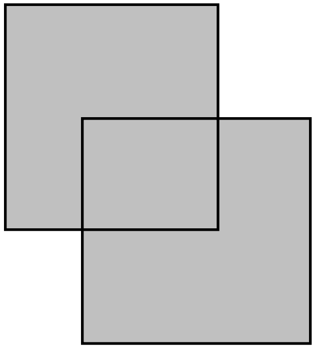
5+8 = 13 $\in$ N closure!

Introduction to Relational Algebra (cont.)

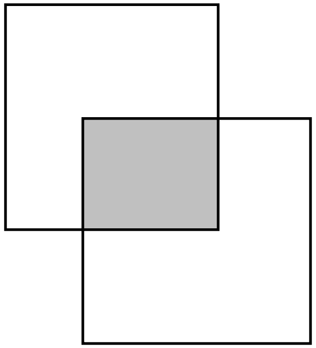
- Relational operators: [defined by Codd, 1970]
  - **Traditional set operations:**
    - Union (∪)
    - Intersection (∩)
    - Difference (–)
    - Cartesian Product / Times (x)
  - **Special relational operations:**
    - Restrict (σ) or Selection
    - Project (Π)
    - Join (⋈)
    - Divide (÷)

Relational Operators

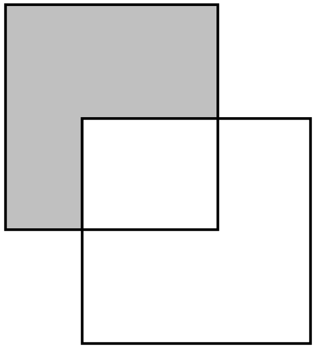
Union (∪)



Intersection (∩)

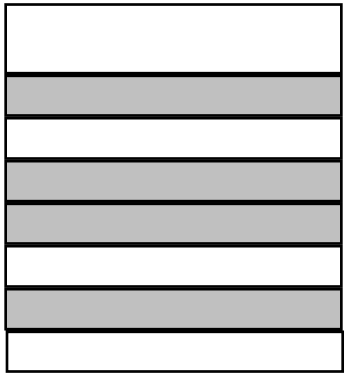


Difference (–)

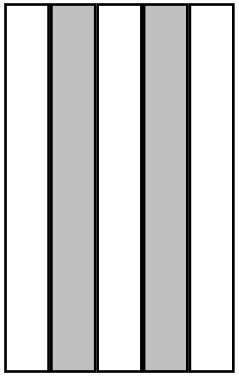


Relational Operators (cont.)

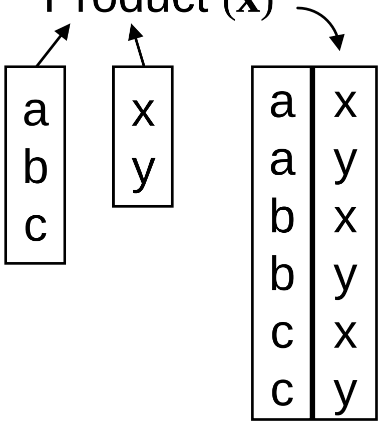
Restrict (σ)



Project (Π)



Product (**x**)

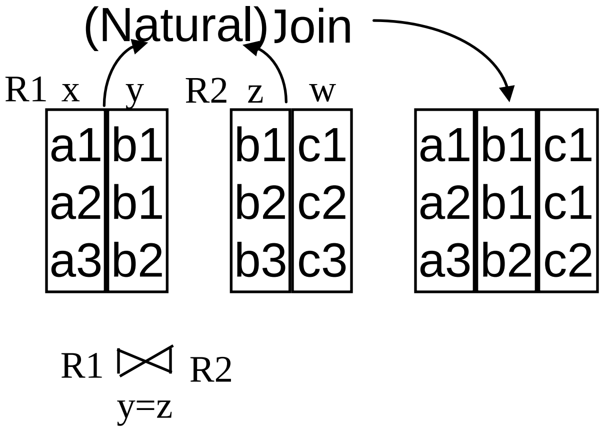


a
b
c

x
y

a	x
a	y
b	x
b	y
c	x
c	y

(Natural) Join



R1

x	y
a1	b1
a2	b1
a3	b2

R2

z	w
b1	c1
b2	c2
b3	c3

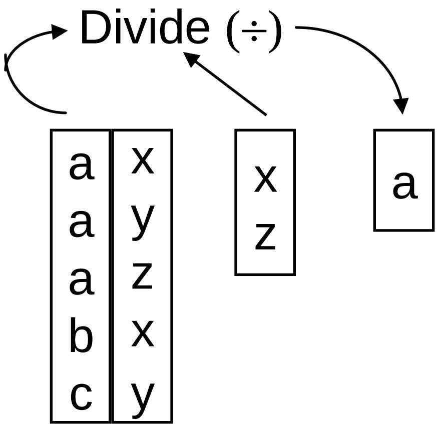
a1	b1	c1
a2	b1	c1
a3	b2	c2

$R1 \underset{y=z}{\bowtie} R2$

R1 x R2

x	y	z	w
a1	b1	b1	c1
a1	b1	b2	c2
a1	b1	b3	c3
a2	b1	b1	c1
⋮	⋮	⋮	⋮

Divide (÷)



a	x
a	y
a	z
b	x
c	y

x
z

a

SQL vs. Relational Operators

- A SQL SELECT contains several relational operators.

<e.g.>

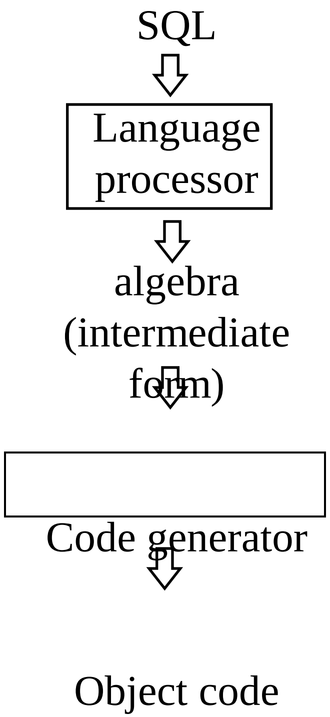
```
SQL:    SELECT      S#, SNAME
        FROM        S, SP
        WHERE       S.S# = SP.S#
        AND         CITY = 'London ‘
        AND         QTY > 200
```

1> $S \bowtie_{S\#} SP$

2> $\sigma_{CITY ='London', QTY>200}$

3> $\Pi_{S\#,SNAME}$

$$\Pi_{S\#, SNAME}\ (\sigma_{CITY='London', QTY>200}\ (S \bowtie_{S\#} SP))$$



SQL

Language processor

algebra (intermediate form)

Code generator

Object code

- BNF (p. 3-44)

Traditional Set Operations

- **Union Compatibility**: two relations are union compatible iff they have identical headings.

  i.e.:

  1. they have same set of attribute name.
  2. corresponding attributes are defined on the same domain.

  - objective: ensure the result is still a relation.

- Union (∪), Intersection (∩) and Difference (–) require Union Compatibility, while Cartesian Product (X) don't.

Traditional Set Operations: UNION

- **A, B:** two union-compatible relations.

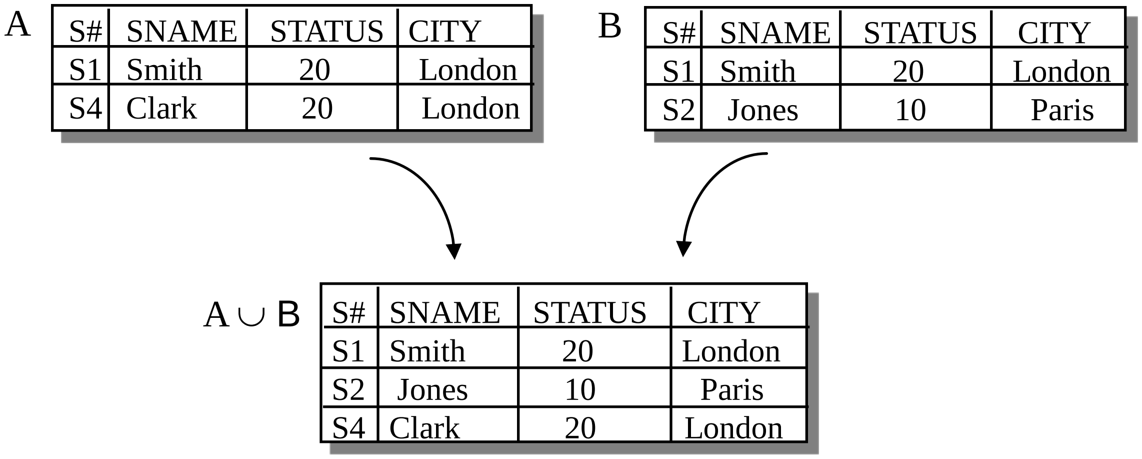
  A : $(X_1,\ldots,X_m)$

  B : $(X_1,\ldots,X_m)$

  - **A UNION B:**
    - **Heading:** $(X_1,\ldots,X_m)$
    - **Body:** the set of all tuples t belonging to either A or B (or both).
  - **Association:**

    $(A \cup B\ ) \cup C = A \cup (\ B \cup C)$
  - **Commutative:**

    $A \cup B\ =\ B \cup A$



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

A ∪ B

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

Traditional Set Operations: INTERSECTION

- **A, B:** two union-compatible relations.

  $A : (X_1,\ldots,X_m)$

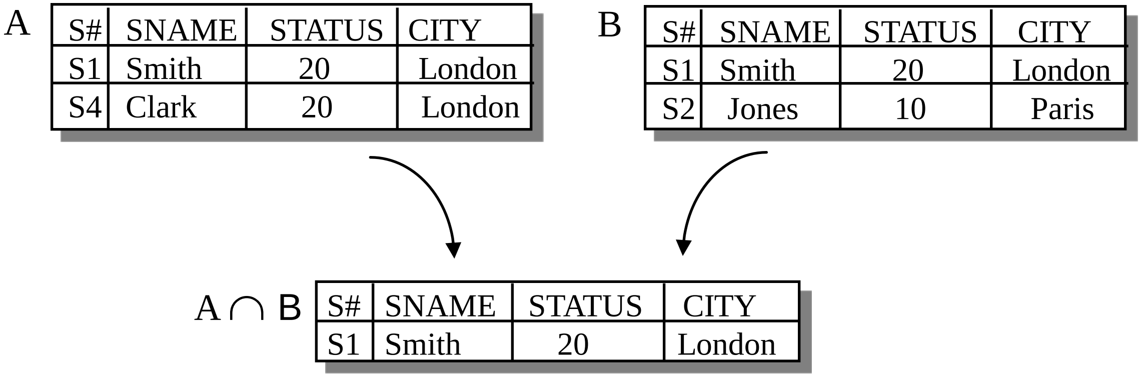
  $B : (X_1,\ldots,X_m)$

- **A INTERSECT B:**
  - **Heading:** $(X_1,\ldots,X_m)$
  - **Body:** the set of all tuples t belonging to **both** A and B.
- **Association:**

  $(A \cap B) \cap C = A \cap (B \cap C)$

- **Commutative:**

  $A \cap B = B \cap A$



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

$A \cap B$

S#	SNAME	STATUS	CITY
S1	Smith	20	London

Traditional Set Operations: DIFFERENCE

- **A, B:** two union-compatible relations.

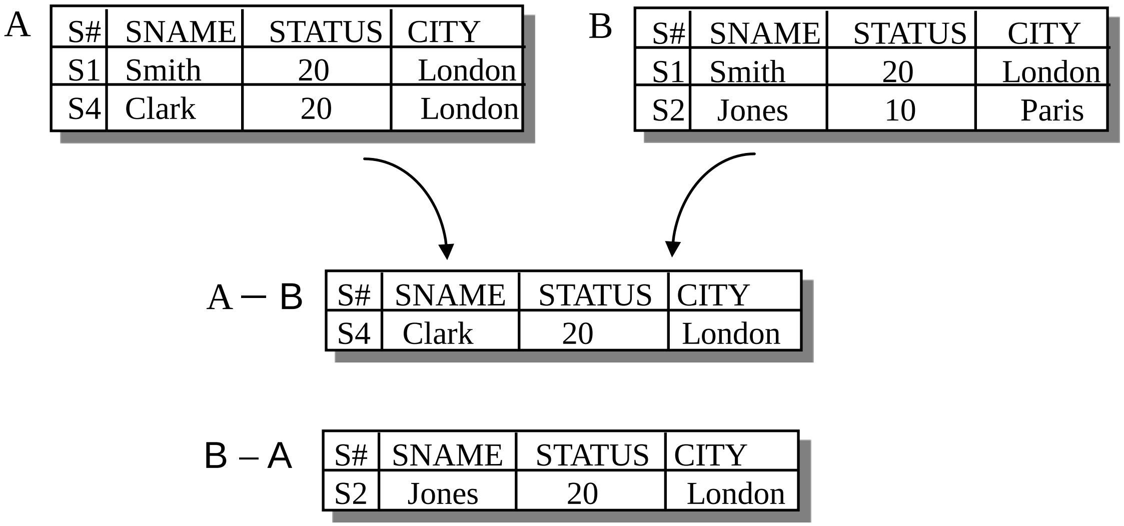
  A : $(X_1,\ldots,X_m)$

  B : $(X_1,\ldots,X_m)$

- **A MINUS B:**
  - **Heading:** $(X_1,\ldots,X_m)$
  - **Body:** the set of all tuples t belonging to A and not to B.
- **Association:** No!

  $(A - B) - C \neq A - (B - C)$
- **Commutative:** No!

  $A - B \neq B - A$



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

A – B

S#	SNAME	STATUS	CITY
S4	Clark	20	London

B – A

S#	SNAME	STATUS	CITY
S2	Jones	20	London

Traditional Set Operations: TIMES

- Extended Cartesian Product (**x**):

  Given:

  $A = \{ a \mid a = (a_1,\ldots,a_m)\}$

  $B = \{ b \mid b = (b_1,\ldots,b_n)\}$

  - Mathematical Cartesian product:

    $A \times B = \{ t \mid t = ((a_1,\ldots,a_m),(b_1,\ldots,b_n))\}$

  - Extended Cartesian Product:

    $A \times B = \{ t \mid t = (a_1,\ldots,a_m,b_1,\ldots,b_n)\}$

    ⇧

    Coalescing

math.

$A = \{x, y\}$

$B = \{y, z\}$

$A \times B = \{(x,y),(x,z),(y,y),(y,z)\}$

- **<u>Product Compatibility:</u>** two relations are product-compatible iff their ***<u>headings are disjoint.</u>***

<e.g.1> A (S#, SNAME)

B (P#, PNAME, COLOR)

⇩ A x B (S#, SNAME, P#, PNAME, COLOR)

A and B are product compatible!

Traditional Set Operations: TIMES (cont.)

<e.g.2> S (S#, SNAME, STATUS, CITY)

P (P#, PNAME, COLOR, WEIGHT, **CITY**)

⇩ S x P (S#, ..., CITY, ..., **CITY**)

S and P are ***not*** product compatible!

⇩

P RENAME CITY AS **PCITY**;

S x P (S#, ..., CITY, ..., **PCITY**)

Traditional Set Operations: TIMES (cont.)

- A, B: two product-compatible relations.

  $A : (X_1,\ldots,X_m),\ A = \{ a \mid a = (a_1,\ldots,a_m)\}$

  $B : (Y_1,\ldots,Y_n),\ B = \{ b \mid b = (b_1,\ldots,b_n)\}$

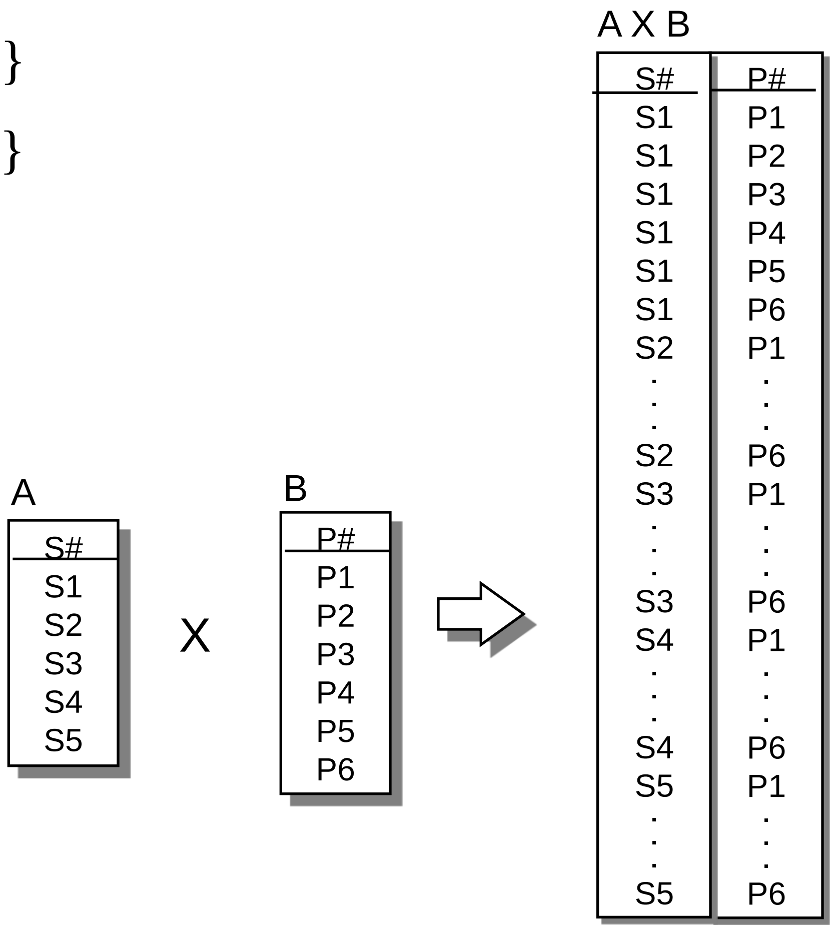
- **A TIMES B: (A x B)**
  - **Heading:** $(X_1,\ldots,X_{m,}Y_1,\ldots,Y_n)$
  - **Body:** $\{ c \mid c = (a_1,\ldots,a_m,b_1,\ldots,b_n)\}$

- **Association:**

  (A x B) x C = A x (B x C)

- **Commutative:**

  A x B = B x A



A

S#
S1
S2
S3
S4
S5

X

B

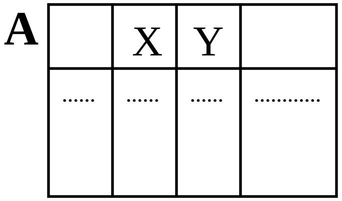
P#
P1
P2
P3
P4
P5
P6

A X B

S#	P#
S1	P1
S1	P2
S1	P3
S1	P4
S1	P5
S1	P6
S2	P1
.	.
.	.
.	.
S2	P6
S3	P1
.	.
.	.
.	.
S3	P6
S4	P1
.	.
.	.
.	.
S4	P6
S5	P1
.	.
.	.
.	.
S5	P6

Special Relational Operations: Restriction

- Restriction: a unary operator or monadic
  - Consider: A: a relation, X,Y: attributes or literal
  - **theta-restriction** (or abbreviate to just 'restriction'):



A WHERE X theta Y   or $\sigma_{X \text{ theta } Y}(A)$

(By Date) (θ) (By Ullman)

theta : =, <>, >, >=, <, <=, etc.

  - The restriction condition (X theta Y) can be extended to be any Boolean combination by including the following equivalences:

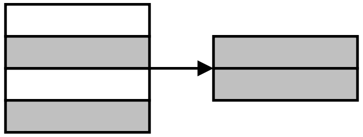
(1) $\sigma_{C1 \text{ and } C2}(A) = \sigma_{C1}(A) \cap \sigma_{C2}(A)$;   (2) $\sigma_{C1 \text{ or } C2}(A) = \sigma_{C1}(A) \cup \sigma_{C2}(A)$;   (3) $\sigma_{\text{not } C}(A) = A - \sigma_{C}(A)$

  - <e.g.> S WHERE CITY='London'? or $\sigma_{CITY='London'}(S)$

S'

S →

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



Special Relational Operations: Projection

- Projection: a unary operator.
  - Consider:

    A   : a relation

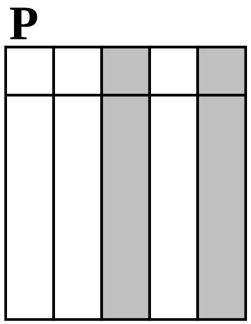
    X,Y,Z : attributes

  - A[X,Y,Z]   or $\Pi_{X,Y,Z}(A)$
  - **Identity projection:**

    A =A   or $\Pi(A) = A$
  - **Nullity projection:**

    A[ ] = ∅   or $\Pi_{\emptyset}(A) = \emptyset$

<e.g.> P[COLOR,CITY]



COLOR	CITY
Red	London
Green	Paris
Blue	Rome
Blue	Paris

Special Relational Operations: Natural Join

- Natural Join: a binary operator.
  - Consider:

    $A : (X_1,\ldots,X_m, Y_1,\ldots,Y_n)$

    $B : (Y_1,\ldots,Y_n, Z_1,\ldots,Z_p)$
  - A JOIN B (or A⋈B): common attributes appear only once. e.g. CITY

    $(X_1,\ldots,X_m, Y_1,\ldots,Y_n, Z_1,\ldots,Z_p)$;
  - **Association:**

    $(A \bowtie B) \bowtie C = A \bowtie (B \bowtie C)$
  - **Commutative:**

    $A \bowtie B = B \bowtie A$
  - if A and B have no attribute in common, then

    $A \bowtie B = A \times B$

Special Relational Operations: Natural Join (cont.)

<e.g.> S **JOIN** P or S ⋈P

S.city = P.city S.city = P.city

S P

S#	SNAME	STATUS	CITY	P#	PNAME	COLOR	WEIGHT
S1	Smith	20	London	P1	Nut	Red	12
S1	Smith	20	London	P4	Screw	Red	14
S1	Smith	20	London	P6	Cog	Red	19
S2	Jones	10	Paris	P2	Bolt	Green	17
S2	Jones	10	Paris	P5	Cam	Blue	12
S3	Blake	30	Paris	P2	Bolt	Green	17
S3	Blake	30	Paris	P5	Cam	Blue	12
S4	Clark	20	London	P1	Nut	Red	12
S4	Clark	20	London	P4	Screw	Red	14
S4	Clark	20	London	P6	Cog	Red	19

CITY
London

Special Relational Operations: Theta Join

- **A, B:** product-compatible relations, A: $(X_1,...,X_m)$, B: $(Y_1,...,Y_n)$
- theta : =, <>, <, >,.....
- $A \underset{X\ theta\ Y}{\bowtie} B = \sigma_{X\ theta\ Y}(A\ x\ B)$
- If theta is '=', the join is called ***equijoin***.

<e.g.> a greater-than join

```
SELECT S.*, P.*
FROM     S, P
WHERE  S.CITY > P.CITY
```

⇩

$\sigma_{CITY>PCITY}$(S x (P RENAME CITY AS PCITY))

⇩

S#	SNAME	STATUS	CITY	P#	PNAME	COLOR	WEIGHT	PCITY
S2	Jones	10	Paris	P1	Nut	Red	12	London
S2	Jones	10	Paris	P4	Screw	Red	14	London
S2	Jones	10	Paris	P6	Cog	Red	19	London
S3	Blake	30	Paris	P1	Nut	Red	12	London
S3	Blake	30	Paris	P4	Screw	Red	14	London
S3	Blake	30	Paris	P6	Cog	Red	19	London

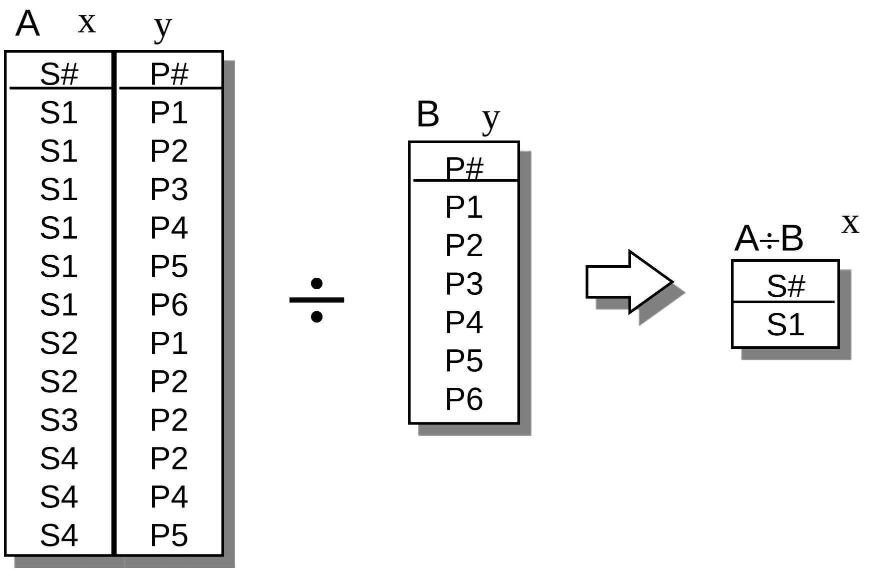
Special Relational Operations: Division

- Division:
  - **A, B:** two relations.

    A : $(X_1,...,X_m, Y_1,...,Y_n)$

    B : $(Y_1,...,Y_n)$
  - A DIVIDEBY B (or A ÷ B):
    - **Heading:** $(X_1,...,X_m)$
    - **Body:** all (X:x) s.t. (X:x,Y:y) in A for all (Y:y) in B

<e.g.> "Get supplier numbers for suppliers who supply all parts."



A x y

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

÷

B y

P#
P1
P2
P3
P4
P5
P6

A÷B x

S#
S1

Special Relational Operations: primitive

- Which of the eight relational operators are primitive?

  1. UNION
  2. DIFFERENCE
  3. CARTESIAN PRODUCT
  4. RESTRICT
  5. PROJECT

- How to define the non-primitive operators by those primitive operators?

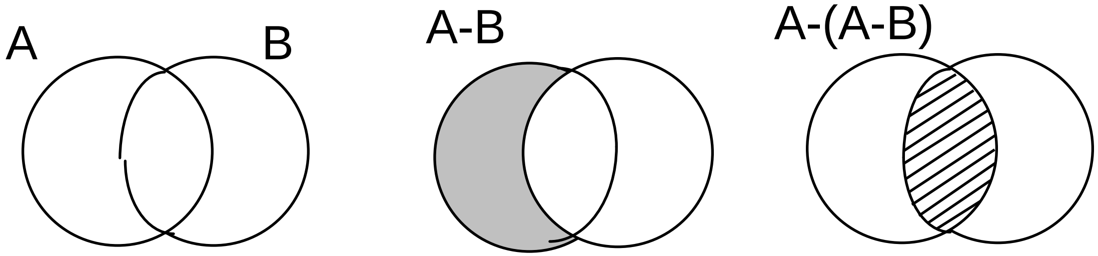
  ① Natural Join: $S \underset{s.city\ =\ p.city}{\bowtie} P$

  ⇩

  $\Pi_{S\#,SNAME,STATUS,CITY,P\#,PNAME,COLOR,WEIGHT}\ (\sigma_{CITY=PCITY}(S\ X\ (P\ RENAME\ CITY\ AS\ PCITY)))$

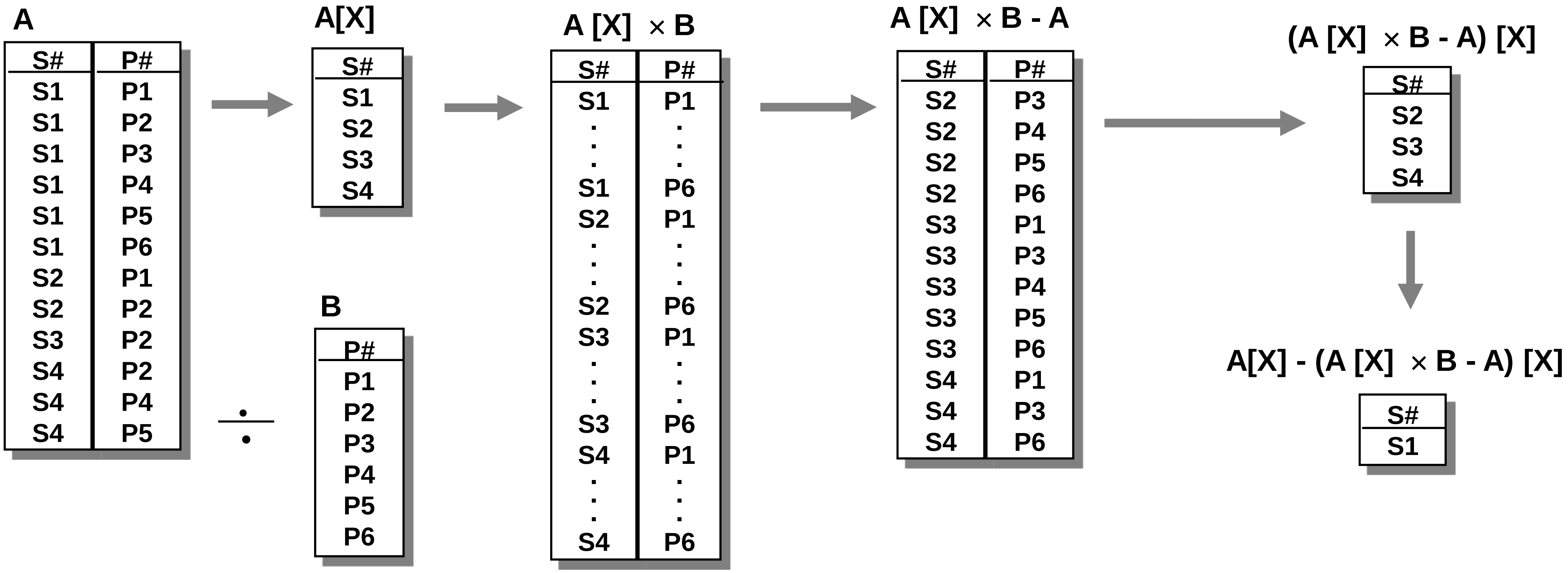
Special Relational Operations: primitive (cont.)

➁ INTERSECT: $A \cap B = A - (A - B)$



Special Relational Operations: primitive (cont.)

③DIVIDE: A ÷ B = A[X] - (A[X] × B - A)[X]



A

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

A[X]

S#
S1
S2
S3
S4

÷

B

P#
P1
P2
P3
P4
P5
P6

A [X] × B

S#	P#
S1	P1
.	.
.	.
.	.
S1	P6
S2	P1
.	.
.	.
.	.
S2	P6
S3	P1
.	.
.	.
.	.
S3	P6
S4	P1
.	.
.	.
.	.
S4	P6

A [X] × B - A

S#	P#
S2	P3
S2	P4
S2	P5
S2	P6
S3	P1
S3	P3
S3	P4
S3	P5
S3	P6
S4	P1
S4	P3
S4	P6

(A [X] × B - A) [X]

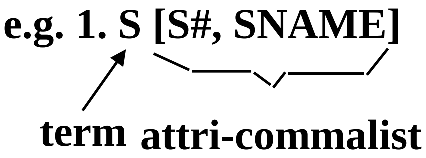
S#
S2
S3
S4

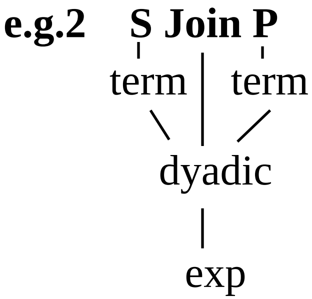
A[X] - (A [X] × B - A) [X]

S#
S1

BNF Grammars for Relational Operator

1. expression ::= monadic-expression | dyadic-expression
2. monadic-expression ::= renaming | restriction | projection
3. renaming ::= term RENAME attribute AS attribute
4. term ::= relation | (expression)
5. restriction ::= term WHERE condition
6. Projection ::= attribute | term [attribute-commalist]
7. dyadic-expression ::= projection dyadic-operation expression
8. dyadic-operation ::= UNION | INTERSECT | MINUS | TIMES | JOIN | DIVIDEBY



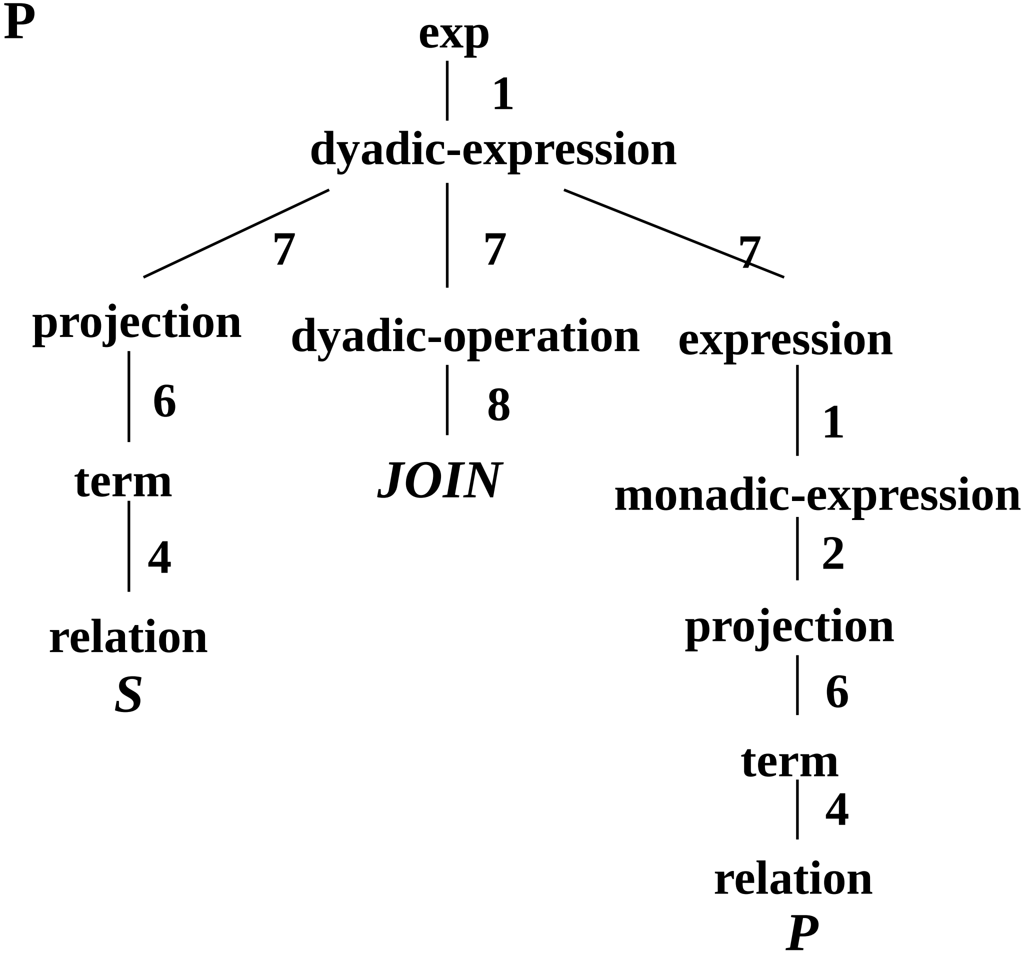


(Back to p. 3-27)

BNF Grammars for Relational Operator

(cont.)

e.g. S JOIN P



exp

1

dyadic-expression

7 7 7

projection dyadic-operation expression

6 8 1

term *JOIN* monadic-expression

4 2

relation projection

S

6

term

4

relation

P

Relational Algebra v.s. Database Language:

- Example : Get supplier name for suppliers who supply part P2.
  - **SQL**:

```
SELECT S.SNAME
FROM   S, SP
WHERE  S.S#  = SP.S#
AND      SP.P#  =  'P2'
```

S#	SNAME	STATUS	CITY	S#	P#	QTY
S1	Smith	20	London	S1	P1	300
S1	Smith	20	London	S1	P2	200
S1	Smith	20	London	S1	P3	400
S1	Smith	20	London	S1	P4	200
S1	Smith	20	London	S1	P5	100
S1	Smith	20	London	S1	P6	100
S2	Jones	10	Paris	S2	P1	300
S2	Jones	10	Paris	S2	P2	400
S3	Blake	30	Paris	S3	P2	200
S4	Clark	20	London	S4	P2	200
S4	Clark	20	London	S4	P4	300
S4	Clark	20	London	S4	P5	400

  - **Relational algebra**:

((S JOIN SP) WHERE P# = 'P2') [SNAME]

or

$$\Pi_{SNAME}\,(\sigma_{P\#='P2'}\,(S \bowtie SP))$$

What is the Algebra for?

(1) Allow writing of expressions which serve as a high-level (SQL) and symbolic representation of the users intend.

(2) Symbolic transformation rules are possible.

A convenient basis for optimization!

e.g. ((S JOIN SP) WHERE P#='P2')[SNAME]

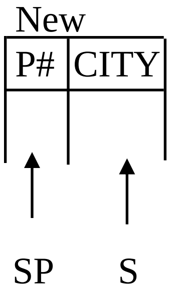
= (S JOIN (SP WHERE P#='P2')) [SNAME]

(p.544; p.11-12)

Back to p.3-66

3.5 Relational Calculus

Introduction to Relational Calculus



- A notation for expressing the definition of some new relations in terms of some given relations.

  <e.g.> SP.P#, S.CITY WHERE SP.S# = S.S#

  definition predicate

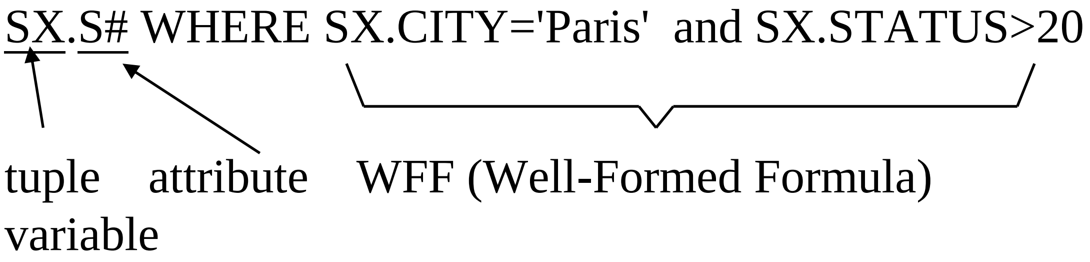
- Based on first order predicate calculus (a branch of mathematical logic).
  - Originated by Kuhn for database language (1967).
  - Proposed by Codd for relational database (1972)
  - ALPHA: a language based on calculus, never be implemented.
  - QUEL: query language of INGRES, influenced by ALPHA.
- Two forms :
  - ***Tuple calculus:*** by Codd..
  - ***Domain calculus:*** by Lacroix and Pirotte.

Tuple Calculus

- BNF Grammar:

<e.g.> "Get supplier number for suppliers in Paris
with status > 20"

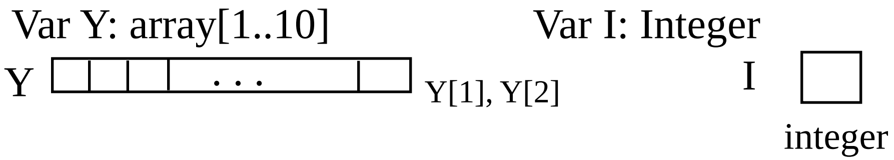
Tuple calculus expression:



SX.S# WHERE SX.CITY='Paris' and SX.STATUS>20

tuple variable attribute WFF (Well-Formed Formula)

Tuple Calculus (cont.)

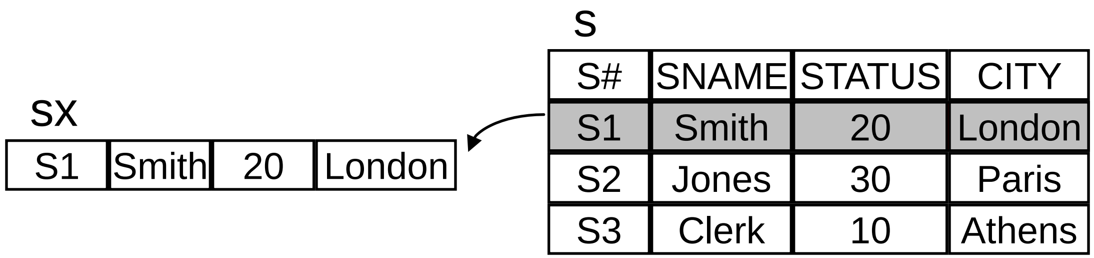


- Tuple variable (or Range variable):
  - A variable that "range over" some named relation.

    <e.g.>:

    In QUEL: (Ingres)

    - RANGE OF SX IS S;
    - RETRIEVE (SX.S#) WHERE SX.CITY = "London"



SX

S1	Smith	20	London

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	30	Paris
S3	Clerk	10	Athens

Tuple Calculus (cont.)

- Implicit tuple variable:

  <e.g.>

  In SQL:

  SELECT S.S# FROM S WHERE S.CITY = 'London‘

  In QUEL:

  RETRIEVE (SX.S#) WHERE SX.CITY='London'

Tuple Calculus: BNF

```
1. range-definition
   ::= RANGE OF variable IS range-item-commalist
2. range-item
   ::= relation | expression
3. expression
    ::= (target-item-commalist)  [WHERE wff]
4. target-item
    ::= variable | variable . attribute [ AS attribute ]
5. wff
    ::= condition
        | NOT wff
        | condition AND wff
        | condition OR wff
        | IF condition THEN wff
        | EXISTS variable (wff)
        | FORALL variable (wff)
        | (wff)
```

Tuple Calculus: BNF - Well-Formed Formula (WFF)

(a) Simple comparisons:

- SX.S# = 'S1'
- SX.S# = SPX.S#
- SPX.P# <> PX.P#

(b) Boolean WFFs:

- NOT SX.CITY='London'
- SX.S#=SPX.S# AND SPX.P#<>PX.P#

(c) Quantified WFFs:

- **EXISTS:** existential quantifier

  <e.g.>

  EXISTS SPX (SPX.S#=SX.S# and SPX.P#= 'P2')

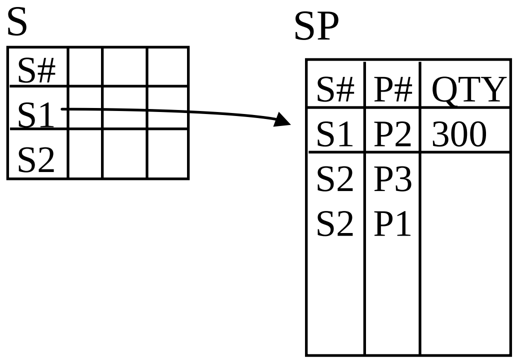
  i.e. There exists an SP tuple with S# value equals to the value of SX.S# and P# value equals to 'P2'

- **FORALL:** universal quantifier

  <e.g.>

  FORALL PX(PX.COLOR = 'Red')

  i.e. For all P tuples, the color is red.



S

S#			
S1			
S2			

SP

S#	P#	QTY
S1	P2	300
S2	P3	
S2	P1	

<Note>: *FORALL x(f) = NOT EXISTS X (NOT f)*

Tuple Calculus: EXAMPLE 1

[Example 1]: Get Supplier numbers for suppliers in Paris with status > 20

- **SQL:**

```
SELECT S#
FROM    S
WHERE CITY = 'Paris' AND STATUS >20
```

- **Tuple calculus:**

SX.S# WHERE SX.CITY= 'Paris‘ AND SX.STATUS > 20

- **Algebra:**

$$\Pi_{S\#}\ (\sigma_{CITY='Paris',\ and\ STATUS>20}(S))$$

Tuple Calculus: EXAMPLE 2

[Example 2]: Get all pairs of supplier numbers such that the two suppliers are located in the same city.

Rename S FIRST, SECOND

- **SQL:** (S.S#) (S.S#)

```
SELECT FIRST.S#, SECOND.S#
FROM S FIRST, S SECOND
WHERE FIRST.CITY = SECOND.CITY AND FIRST.S# < SECOND.S#;
```

- **Tuple calculus:**

```
FIRSTS#=SX.S#, SECONDS# =SY.S#
WHERE SX.CITY=SY.CITY AND  SX.S# < SY.S#
```

- **Algebra:**

$$\Pi_{\text{FIRSTS\#,SECONDS\#}} (\sigma_{\text{FIRSTS\#<SECONDS\#}}$$
$$((\Pi_{\text{FIRSTS\#,CITY}} (\text{S RENAME S\# AS FIRSTS\#})) \bowtie_{\text{city=city}}$$
$$(\Pi_{\text{SECONDS\#,CITY}} (\text{S RENAME S\# AS SECONDS\#}))))$$

{S1, S1}
{S1, S4}
{S4, S1}
{S4, S4}

Output:

{S1,S4}{S2,S3}

Tuple Calculus: EXAMPLE 3

[Example 3]: Get supplier names for suppliers who supply all parts.

- **SQL:**

```
SELECT SNAME
FROM S
WHERE NOT EXISTS
          ( SELECT *   FROM P
           WHERE NOT EXISTS
              ( SELECT *  FROM SP
              WHERE S# = S.S# AND  P# = P.P# ));
```

- **Tuple calculus:**

```
SX.SNAME
WHERE FORALL PX                P1, P2, ..., P6 ∈ PX
               (EXISTS SPX     S1
                  ( SPX.S# = SX.S# AND SPX.P# = PX.P#))
```

- **Algebra:**

$$\Pi_{SNAME}\ (\underbrace{((\Pi_{S\#,P\#}\, \underset{A}{SP}) \div (\Pi_{P\#}\, \underset{B}{P}))}_{S1} \bowtie S)$$

(P3-43)

SX

S1	Smith	....

S

S#			
S1			

P

P#			
P1			

SP

S#	P#	QTY
S1	P1	

Tuple Calculus: EXAMPLE 4 [參考用]

[Example 4]: Get part numbers for parts that either weigh more than 16 pounds or are supplied by supplier S2, or both.

- **SQL:**

```
 SELECT P#   FROM P
 WHERE    WEIGHT > 16
UNION
 SELECT P#    FROM SP
 WHERE S# = 'S2'
```

- **Tuple calculus:**

```
 RANGE OF PU IS
(PX.P# WHERE PX.WEIGHT>16),
(SPX.P# WHERE SPX.S#='S2');
 PU.P#;
```

- **Algebra:**

$$(\Pi_{P\#}\ (\sigma_{WEIGHT>16}\ P)) \cup (\Pi_{P\#}\ (\sigma_{S\#='S2'}\ SP))$$

Relational Calculus v.s. Relational Algebra.

Algebra	**Calculus**
Provides explicit operations [e.g.JOIN, UNION, PROJECT,...] to ***build*** desired relation from the given relations.	Only provide a notation for ***formulate*** the definition of that desired relation in terms of those given relation.
<e.g.> Get supplier numbers and cities for suppliers who supply part P2.	
1> JOIN S with SP on S# 2> RESTRICT the result with P# = 'P2' 3> PROJECT the result on S# and CITY	SX.S#, SX.CITY WHERE EXISTS SPX (SPX.S#=SX.S# AND SPX.P#= 'P2')
Prescriptive (how?)	descriptive (what ?)
Procedural	non-procedural

("expressive power")

Relational Calculus ≡ Relational Algebra

- Codd's reduction algorithm:

  1. Show that any calculus expression can be reduced to an algebraic equivalent.

  ⇩

  Algebra ⊇ Calculus

  2. show that any algebraic expression can be reduced to a calculus equivalent

  ⇩

  Calculus ⊇ Algebra

⇩

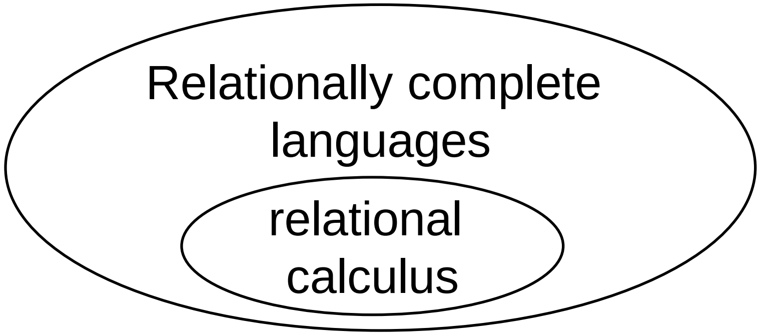
Algebra ≡ Calculus

Relationally Complete

- Def : **A language is said to be** *relationally complete* **if it is at least as powerful as the relational calculus.**

  i.e. **if any relation definable via a** *single expression* **of the calculus is definable via a single expression of the language.**

  <e.g.> SQL,QUEL



- Show a language L is relationally complete

⇩

Show that L includes analogs of the five primitive algebraic operation.

⇩

Easier than show L is at least as powerful as relational calculus.

Domain Calculus
(Domain-Oriented Relational Calculus)

- Distinctions between domain calculus and tuple calculus:
  - Variables range over ***domain*** instead of relation.
  - Support an additional form of comparison:

    the membership condition

<e.g.1> SP(S#:'S1', P#:'P1')

True iff exists a tuple in SP with S#='S1‘ and P# = 'P1'

<e.g.2> SP(S#: SX, P#:PX)

True iff exists a tuple in SP with

S#=current value of domain var. SX.

P#=current value of domain var. PX.

Var. SX PX

S5 P9

S

S#			
S1			
S2			
S3			
S4			

e.g.: S# Domain
={S1, S2, ..., S100}

S# Range
={S1, S2, S3, S4}

SP

S#	P#	QTY

Domain Calculus:
attributes WHERE membership_condition

Tuple Calculus:
term WHERE wff

Domain Calculus:
term WHERE m-c

- Domain Calculus expressions:

e.g.1 SX

(i.e. all possible values of supplier number)

e.g. {S1, ..., S100}

e.g.2 SX WHERE $\underbrace{S(S\#:SX)}_{\text{condition}}$

(i.e. all S# in relation S)

e.g. {S1, ..., S4}

e.g.3 SX WHERE S(S#:SX, CITY:'London')

(i.e. subset of S# in S for which city is 'London')

SQL:

```
Select   S#
From     S
Where    City = 'London'
```

QBE

S	S#	SNAME	STATUS	CITY
	P.			'London'

print

e.g.4

SX, CITYX

WHERE S(S#:SX, CITY:CITYX) AND SP(S#: SX,P#: 'P2')

(i.e. subset of S# and CITY in S for the suppliers who supply P2)

Query-by-Example (QBE)

- **An attractive realization of the domain calculus**
- **Simple in syntax**
- e.g. Get supplier numbers for suppliers in Paris with status > 20
  - **Tuple calculus:**

```
SX.S#
WHERE SX.CITY= 'Paris'
   AND SX.STATUS  > 20
```

  - **Domain calculus:**

```
 SX
WHERE EXISTS STATUSX
   (STATUSX >20) AND
   S(S#:SX, STATUS:STATUSX, CITY:'Paris')
```

  - **QBE:**

S	S#	SNAME	STATUS	CITY
	P.		>20	“Paris”

P. : print or present

Query-by-Example (cont.)

[Example]: Get all pairs of supplier numbers such that the two suppliers are located in the same city.

- **SQL**: SELECT FIRST.S#, SECOND.S#
  FROM S FIRST, S SECOND
  WHERE FIRST.CITY = SECOND.CITY AND FIRST.S# < SECOND.S#;
- **Tuple calculus**:
  FIRSTS# = SX.S#, SECONDS# = SY.S#
  WHERE SX.CITY = SY.CITY AND SX.S# < SY.S#
- **Domain calculus:**
  FIRSTS# = SX, SECONDS# = SY
  WHERE EXISTS CITYZ
  (S(S#:SX,CITY:CITYZ) AND S(S#.SY,CITY:CITYZ) AND SX<SY)

{S1, S4}
{S2, S3}

- **QBE:**

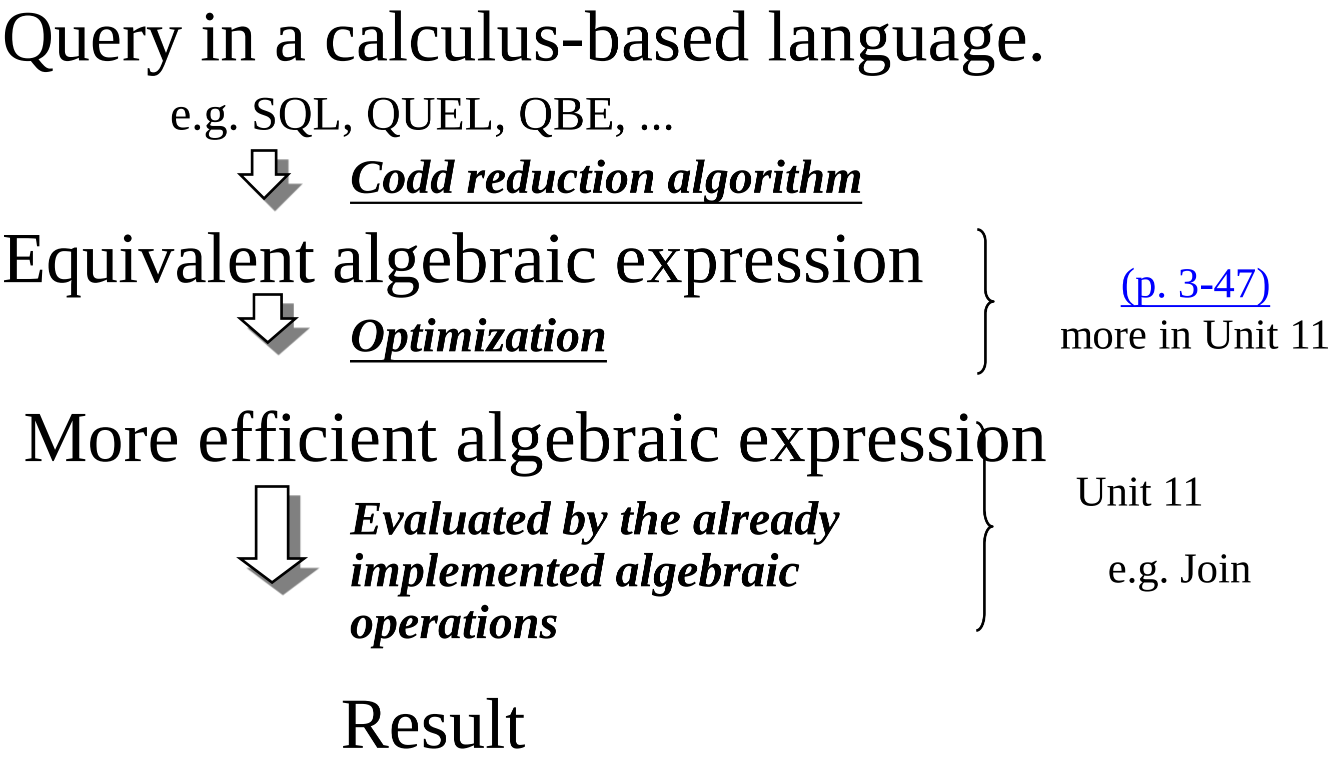
S	S#	CITY
	-SX	-CZ
	-SY	-CZ

P.	-SX	-SY

_SX, _SY, _CZ are ***examples***.

Concluding Remarks

- Relational algebra provide a convenient target language as a vehicle for a possible implementation of the calculus.



Query in a calculus-based language.

e.g. SQL, QUEL, QBE, ...

Codd reduction algorithm

Equivalent algebraic expression

Optimization

(p. 3-47)
more in Unit 11

More efficient algebraic expression

Evaluated by the already implemented algebraic operations

Unit 11

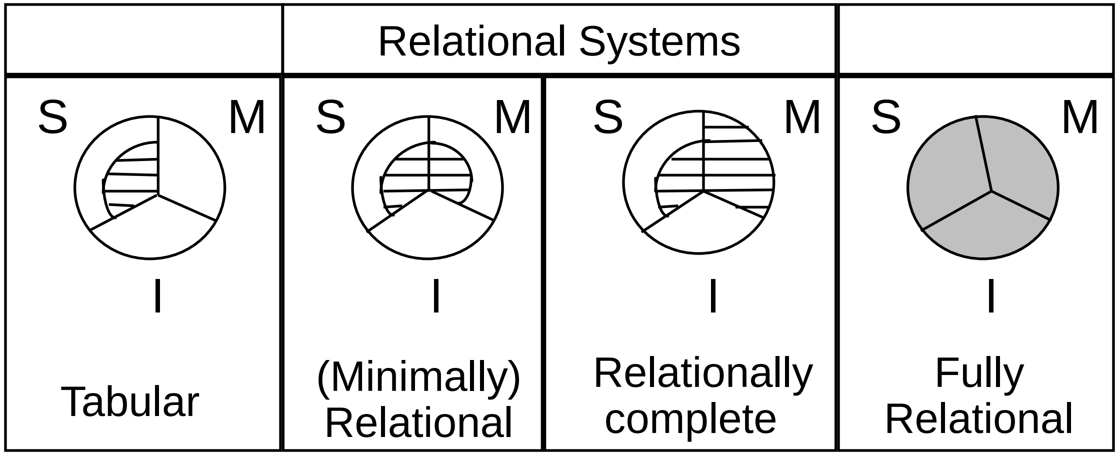
e.g. Join

Result

Concluding Remarks (cont.)

- A spectrum of data management system:

S: Structure (Table)
M: Manipulative
I: Integrity



Foreign Key Statement

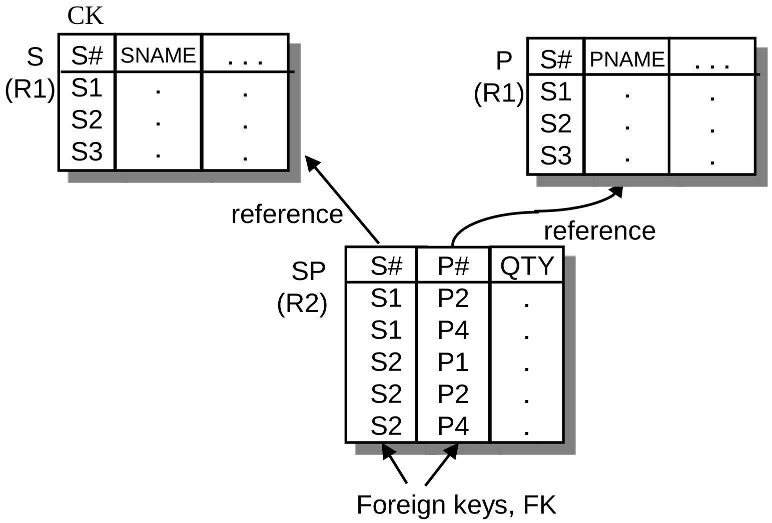
- Descriptive statements:

```
FOREIGN KEY (foreign key) REFERENCES target
                   NULLS  [NOT] ALLOWED
                   DELETE OF target effect
                   UPDATE OF target-primary-key effect;
```

effect: one of {RESTRICTED, CASCADES, NULLIFIES}

<e.g.1> (p.269)

```
CREATE TABLE SP
(S# S# NOT NULL, P# P# NOT NULL,
QTY QTY NOT NULL,
 PRIMARY KEY (S#, P#),
 FOREIGN KEY (S#) REFERENCE S
               ON DELETE CASCADE
               ON UPDATE CASCADE,
 FOREIGN KEY (P#) REFERENCE P
               ON DELETE CASCADE
               ON UPDATE CASCADE,
 CHECK (QTY>0 AND QTY<5001));
```



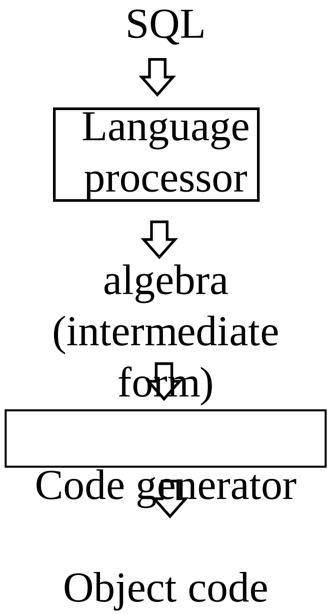
S#	SNAME	. . .
S1	.	.
S2	.	.
S3	.	.

S#	PNAME	. . .
S1	.	.
S2	.	.
S3	.	.

S#	P#	QTY
S1	P2	.
S1	P4	.
S2	P1	.
S2	P2	.
S2	P4	.

SQL vs. Relational Operators

- A SQL SELECT contains several relational operators.


SQL

Language processor

algebra (intermediate form)

Code generator

Object code

<e.g.>

```
SQL:    SELECT    S#, SNAME
        FROM      S, SP
        WHERE     S.S# = SP.S#
        AND       CITY = 'London ‘
        AND       QTY > 200
```

1> $S \bowtie_{S\#} SP$

2> $\sigma_{CITY ='London', QTY>200}$

3> $\Pi_{S\#,SNAME}$

= $\Pi_{S\#, SNAME} (\sigma_{CITY='London', QTY>200} (S \bowtie_{S\#} SP))$

- BNF (p. 3-44)