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 DBMS
<e.g.> DB2, INGRES, SYBASE, Oracle

Relational Data Model
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.

\[<\text{e.g.}>\]

<table>
<thead>
<tr>
<th>Entity</th>
<th>Relationship</th>
<th>Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mark</td>
<td>Works_in</td>
<td>Math_Dept</td>
</tr>
</tbody>
</table>

WORKS_IN

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mark</td>
<td>Math_Dept</td>
</tr>
</tbody>
</table>
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.

```
<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>STATUS</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Smith</td>
<td>20</td>
<td>London</td>
</tr>
<tr>
<td>S1</td>
<td>Jones</td>
<td>10</td>
<td>Paris</td>
</tr>
<tr>
<td>S2</td>
<td>Blake</td>
<td>30</td>
<td>Paris</td>
</tr>
<tr>
<td>S3</td>
<td>Clark</td>
<td>20</td>
<td>London</td>
</tr>
<tr>
<td>S4</td>
<td>Adams</td>
<td>30</td>
<td>Athens</td>
</tr>
</tbody>
</table>
```
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.*
  FROM P, SP
  WHERE P.P# = SP.P#
  ```

  ```
  SELECT P.*, SP.*
  FROM P, SP
  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**: 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.

  \[ \{A_1:V_{i_1}, \ A_2:V_{i_2}, ..., \ A_n:V_{i_n}\}, \text{ where } I = 1...m \]

  or

  \[ \{t_1, t_2, t_3, ..., t_m\} \]
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**

<table>
<thead>
<tr>
<th>S#</th>
<th>PQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>{ (P1,300),</td>
</tr>
<tr>
<td></td>
<td>(P2, 200),</td>
</tr>
<tr>
<td></td>
<td>(P3, 400),</td>
</tr>
<tr>
<td></td>
<td>(P4, 200),</td>
</tr>
<tr>
<td></td>
<td>(P5, 100),</td>
</tr>
<tr>
<td></td>
<td>(P6, 100) }</td>
</tr>
<tr>
<td>S2</td>
<td>{ (P1, 300),</td>
</tr>
<tr>
<td></td>
<td>(P2, 400) }</td>
</tr>
<tr>
<td>S3</td>
<td>{ (P2, 200) }</td>
</tr>
<tr>
<td>S4</td>
<td>{ (P2, 200),</td>
</tr>
<tr>
<td></td>
<td>(P4, 300),</td>
</tr>
<tr>
<td></td>
<td>(P5, 400) }</td>
</tr>
</tbody>
</table>

\[
\text{Normalized}\quad\text{INF}
\]

- **degree**: 2
- **domains**:
  \[
  S\# = \{ S1, S2, S3, S4 \}
  \]
  \[
  PQ = \{ <p,q> | p \in \{ P1, P2, ..., P6 \} \}
  \]
  \[
  q \in \{ x | 0 \leq x \leq 1000 \}
  \]
- a mathematical relation

- **degree**: 3
- **domains**:
  \[
  S\# = \{ S1, S2, S3, S4 \}
  \]
  \[
  P\# = \{ P1, P2, ..., P6 \}
  \]
  \[
  QTY = \{ x | 0 \leq x \leq 1000 \}
  \]
- 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 differences:
  - 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.*
  
  \[
  \text{CREATE SNAPSHOT SC AS SELECT S#, CITY FROM S REFRESH EVERY DAY;}
  \]
  
  - 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:
  - **Uniqueness**: At any time, no two tuples of R have the same value for K.
  - **Minimum**: 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.

- **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.

<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>STATUS</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Smith</td>
<td>20</td>
<td>London</td>
</tr>
<tr>
<td>S4</td>
<td>Clark</td>
<td>20</td>
<td>London</td>
</tr>
</tbody>
</table>

Wei-Pang Yang, Information Management, NDHU
**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.

```
<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>.</th>
<th>.</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>S2</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>S3</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>
```

```
P (R1)
P# | PNAME | .   | .   | .   |
----|-------|-----|-----|-----|
| P1 |       | .   | .   | .   |
| P2 |       | .   | .   | .   |
| P3 |       | .   | .   | .   |
| P4 |       | .   | .   | .   |
```

```
<table>
<thead>
<tr>
<th>S#</th>
<th>P#</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>P2</td>
<td>.</td>
</tr>
<tr>
<td>S1</td>
<td>P4</td>
<td>.</td>
</tr>
<tr>
<td>S2</td>
<td>P1</td>
<td>.</td>
</tr>
<tr>
<td>S2</td>
<td>P2</td>
<td>.</td>
</tr>
<tr>
<td>S2</td>
<td>P4</td>
<td>.</td>
</tr>
</tbody>
</table>
```

Foreign keys, FK
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\sim 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

<User issues:>
DELETES 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'

![Diagram showing referential integrity rule](image)
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.

  \[
  \langle \text{e.g.} \rangle \quad R3 = \sigma(R1 \bowtie R2) \quad T_1 \leftarrow R_1 \text{ join } R_2 \\
  R_3 \leftarrow T_1 \text{ selection}
  \]

  \[
  \{\{0,1,2,3\},+\}
  \]

  Integer  
  \[
  \begin{array}{cccc}
  + & 0 & 1 & 2 & 3 \\
  0 & 0 & 1 & 2 & 3 \\
  1 & 1 & 2 & 3 & 4 \\
  2 & 2 & 3 & 4 & 5 \\
  3 & 3 & 4 & 5 & 6 \\
  \end{array}
  \]

  \[
  \{\text{relations; OP}_1, \text{OP}_2, \ldots, \text{OP}_8\}
  \]

  \[
  \begin{array}{cccc}
  (\text{OP}_2(\text{OP}_1(A)) \text{ OP}_3 B) & \oplus & 0 & 1 & 2 & 3 \\
  0 & 0 & 1 & 2 & 3 & 0 \\
  1 & 1 & 2 & 3 & 0 & 1 \\
  2 & 2 & 3 & 0 & 1 & 2 \\
  3 & 3 & 1 & 0 & 2 & 3 \\
  \end{array}
  \]

  \[
  \{1,2,3,\ldots\}
  \]

  \[
  \begin{array}{cccc}
  \text{NOT Closure!} & \oplus & 0 & 1 & 2 & 3 \\
  \end{array}
  \]

  \[
  \text{N} = \{1,2,3,\ldots\}
  \]

  \[
  \text{Closure!}
  \]

  \[
  \begin{array}{cccc}
  1+2 = 3 \in N \\
  5+8 = 13 \in N \text{ closure!}
  \end{array}
  \]
Introduction to Relational Algebra (cont.)

- Relational operators: [defined by Codd, 1970]
  
  - **Traditional set operations:**
    - Union (\( \cup \))
    - Intersection (\( \cap \))
    - Difference (\( - \))
    - Cartesian Product / Times (\( x \))

  - **Special relational operations:**
    - Restrict (\( \sigma \)) or Selection
    - Project (\( \Pi \))
    - Join (\( \Join \))
    - Divide (\( \div \))
Relational Operators

Union ($\cup$)

Intersection ($\cap$)

Difference ($\setminus$)
Relational Operators (cont.)

Restrict ($\sigma$)

Project ($\Pi$)

Product ($\times$)

Join (Natural)

Divide ($\div$)
SQL vs. Relational Operators

- A SQL SELECT contains several relational operators.
  
  \[
  \text{SQL: } \begin{align*}
  &\text{SELECT } S\#, \text{ SNAME} \\
  &\text{FROM } S, \text{ SP} \\
  &\text{WHERE } S.S\# = \text{SP.S\#} \\
  &\text{AND } \text{CITY} = 'London' \\
  &\text{AND } \text{QTY} > 200
  \end{align*}
  \]

  \[
  \xrightarrow{\text{SQL}} \text{Language processor} \xrightarrow{\text{algebra (intermediate form)}} \text{Code generator} \xrightarrow{\text{Object code}}
  \]

  \[
  1> S \bowtie_{S\#} \text{SP} \\
  2> \sigma_{\text{CITY}='London', \text{QTY}>200} \\
  3> \Pi_{S\#, \text{SNAME}}
  \]

  \[
  \Pi_{S\#, \text{SNAME}} \left( \sigma_{\text{CITY}='London', \text{QTY}>200} \left( S \bowtie_{S\#} \text{SP} \right) \right)
  \]

- 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:**
  \[A \cup B = B \cup A\]
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**:
  
  \[
  A \cap B = B \cap A
  \]

\[
\begin{array}{|c|c|c|c|}
\hline
S# & SNAME & STATUS & CITY  \\
\hline
S1 & Smith & 20 & London  \\
S4 & Clark & 20 & London  \\
\hline
\end{array}
\]
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!
  
  \[ A - B \neq B - A \]
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
  - **Product Compatibility**: two relations are product-compatible iff their headings are disjoint.

  <e.g.1>  
  A (S#, SNAME)  
  B (P#, PNAME, COLOR)  
  \[ A \times B (S#, SNAME, P#, PNAME, COLOR) \]
  A and B are product compatible!
Traditional Set Operations: TIMES (cont.)

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

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

\[ \downarrow \]

\( S \times P \ (S\#, \ ..., \ CITY, \ ..., \ CITY) \)

S and P are *not* product compatible!

\[ \downarrow \]

\( P \ \text{RENAME} \ \text{CITY} \ \text{AS} \ \text{PCITY}; \)

\[ \downarrow \]

\( S \times P \ (S\#, \ ..., \ \text{CITY}, \ ..., \ \text{PCITY}) \)
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:**
  
  \[(A \times B) \times C = A \times (B \times C)\]

- **Commutative:**
  
  \[ A \times B = B \times A \]
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'):
    \[ A \text{ WHERE } X \theta Y \text{ or } \sigma_{X \theta Y}(A) \]
    (By Date) \quad \text{(By Ullman)}

- The restriction condition \((X \theta Y)\) can be extended to be any Boolean combination by including the following equivalences:
  1. \[ \sigma_{C_1 \text{ and } C_2}(A) = \sigma_{C_1}(A) \cap \sigma_{C_2}(A) \]
  2. \[ \sigma_{C_1 \text{ or } C_2}(A) = \sigma_{C_1}(A) \cup \sigma_{C_2}(A) \]
  3. \[ \sigma_{\text{not } C}(A) = A - \sigma_C(A) \]

- **<e.g.>** \(S \text{ WHERE } \text{CITY}='\text{London}'? \text{ or } \sigma_{\text{CITY}='\text{London'}}(S)\)
Special Relational Operations: Projection

- Projection: a unary operator.
  - Consider:
    \[ A : \text{a relation} \]
    \[ X,Y,Z : \text{attributes} \]
  - \[ A[X,Y,Z] \quad \text{or} \quad \Pi_{X,Y,Z}(A) \]
  - **Identity projection:**
    \[ A = A \quad \text{or} \quad \Pi(A) = A \]
  - **Nullity projection:**
    \[ A[\ ] = \emptyset \quad \text{or} \quad \Pi_{\emptyset}(A) = \emptyset \]

<e.g.> \[ P[\text{COLOR,CITY}] \]

<table>
<thead>
<tr>
<th>COLOR</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>London</td>
</tr>
<tr>
<td>Green</td>
<td>Paris</td>
</tr>
<tr>
<td>Blue</td>
<td>Rome</td>
</tr>
<tr>
<td>Blue</td>
<td>Paris</td>
</tr>
</tbody>
</table>
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 \( \bowtie \) 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.)

\[ \text{S JOIN P \ or \ S } \bowtie \text{P} \]
\[ \text{S.city = P.city} \quad \text{S.city = P.city} \]

\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\text{S#} & \text{SNAME} & \text{STATUS} & \text{CITY} & \text{P#} & \text{PNAME} & \text{COLOR} & \text{WEIGHT} \\
\hline
S1 & Smith & 20 & London & P1 & Nut & Red & 12 \\
S1 & Smith & 20 & London & P4 & Screw & Red & 14 \\
S1 & Smith & 20 & London & P6 & Cog & Red & 19 \\
S2 & Jones & 10 & Paris & P2 & Bolt & Green & 17 \\
S2 & Jones & 10 & Paris & P5 & Cam & Blue & 12 \\
S3 & Blake & 30 & Paris & P2 & Bolt & Green & 17 \\
S3 & Blake & 30 & Paris & P5 & Cam & Blue & 12 \\
S4 & Clark & 20 & London & P1 & Nut & Red & 12 \\
S4 & Clark & 20 & London & P4 & Screw & Red & 14 \\
S4 & Clark & 20 & London & P6 & Cog & Red & 19 \\
\hline
\end{array}
\]
Special Relational Operations: Theta Join

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

\(<\text{e.g.}>\) a greater-than join

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

\[\sigma_{\text{CITY>P_CITY}}(S \times (P \text{ RENAME CITY AS PCITY}))\]
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

\(<\text{e.g.}>\ "\text{Get supplier numbers for suppliers who supply all parts.}"\)

<table>
<thead>
<tr>
<th>S#</th>
<th>P#</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>P1</td>
</tr>
<tr>
<td>S1</td>
<td>P2</td>
</tr>
<tr>
<td>S1</td>
<td>P3</td>
</tr>
<tr>
<td>S1</td>
<td>P4</td>
</tr>
<tr>
<td>S1</td>
<td>P5</td>
</tr>
<tr>
<td>S1</td>
<td>P6</td>
</tr>
<tr>
<td>S2</td>
<td>P1</td>
</tr>
<tr>
<td>S2</td>
<td>P2</td>
</tr>
<tr>
<td>S3</td>
<td>P2</td>
</tr>
<tr>
<td>S4</td>
<td>P2</td>
</tr>
<tr>
<td>S4</td>
<td>P4</td>
</tr>
<tr>
<td>S4</td>
<td>P5</td>
</tr>
</tbody>
</table>

\(A ÷ B\)
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: $S \bowtie P$
     $s.c\text{\textit{ity}} = p.c\text{\textit{ity}}$

      $\Pi s# , sname, status, city, p# , p\text{\textit{name}}, color, weight \ (\sigma_{\text{\textit{city}}} = p\text{\textit{city}} \ (S \times (P \ \text{RENAME} \ \text{\textit{city}} \ \text{\textit{AS}} \ p\text{\textit{city}}))))$
Special Relational Operations: primitive (cont.)

\( \Ω \text{INTERSECT: } A \cap B = A - (A - B) \)

\[ \begin{align*}
A & \quad B \\
\cap & \quad \cap \\
A - B & \quad A - (A - B)
\end{align*} \]
Special Relational Operations: primitive (cont.)

3. **DIVIDE:** \( A \div B = A[X] - (A[X] \times B - A)[X] \)
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

(e.g. 1. S [S#, SNAME]
    term attri-commalist

(e.g. 2) S Join P
term term
dyadic
   exp

(Back to p. 3-27)
BNF Grammars for Relational Operator

(continued)

e.g. S JOIN P

\[
\text{exp} \quad \begin{array}{c}
\text{dyadic-expression} \\
\text{projection} \\
\text{term} \\
\text{relation} \\
S
\end{array}
\begin{array}{c}
1 \\
7 \\
6 \\
4 \\
7
\end{array}
\begin{array}{c}
\text{dyadic-operation} \\
\text{expression} \\
\text{JOIN} \\
\text{monadic-expression} \\
\text{projection} \\
\text{term} \\
\text{relation} \\
P
\end{array}
\begin{array}{c}
7 \\
8 \\
1 \\
2 \\
6 \\
4 \\
7
\end{array}
\]
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'
    ```
  
  - **Relational algebra:**
    
    $$((S \bowtie SP) \text{ WHERE } P# = 'P2') [\text{SNAME}]$$
    
    or
    
    $$\Pi_{\text{SNAME}} (\sigma_{P#='P2'} (S \bowtie SP))$$
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!*

\[
\begin{align*}
\text{e.g. } & \quad (( S \text{ JOIN } SP ) \text{ WHERE } P#='P2')[\text{SNAME}] \\
& \quad = (S \text{ JOIN } ( SP \text{ WHERE } P#='P2')) [\text{SNAME}]
\end{align*}
\]

(p.544; p.11-12)
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.> \( \text{SP.P#}, \text{S.CITY} \) WHERE \( \text{SP.S#} = \text{S.S#} \)

- 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:**

\[
S.X.S\# \text{ WHERE } S.X.CITY='Paris' \text{ and } S.X.STATUS>20
\]

- tuple
- attribute
- WFF (Well-Formed Formula)
- variable
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"

<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>STATUS</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Smith</td>
<td>20</td>
<td>London</td>
</tr>
<tr>
<td>S2</td>
<td>Jones</td>
<td>30</td>
<td>Paris</td>
</tr>
<tr>
<td>S3</td>
<td>Clerk</td>
<td>10</td>
<td>Athens</td>
</tr>
</tbody>
</table>
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
  
  \[
  \text{EXISTS SPX (SPX.S#=SX.S# and SPX.P#='P2')} \]
  
  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
  
  \[
  \text{FORALL PX(PX.COLOR = 'Red')} \]
  
  i.e. For all P tuples, the color is red.

<Note>: **FORALL x(f) = NOT EXISTS X (NOT f)**
[Example 1]: Get Supplier numbers for suppliers in Paris with status > 20

- **SQL:**
  ```sql
  SELECT S#
  FROM S
  WHERE CITY = 'Paris' AND STATUS > 20
  ```

- **Tuple calculus:**
  ```latex
  SX.S# \text{ WHERE SX.CITY} = \text{'Paris'} \text{ AND SX.STATUS} > 20
  ```

- **Algebra:**
  ```latex
  \Pi_{S#} (\sigma_{\text{CITY}='Paris', \text{ and STATUS}>20} (S))
  ```
[Example 2]: Get all pairs of supplier numbers such that the two suppliers are located in the same city.

Rename \( S \) FIRST, SECOND

- **SQL:**
  
  \[
  \begin{aligned}
  &\text{SELECT FIRST.S#, SECOND.S#} \\
  &\text{FROM S FIRST, S SECOND} \\
  &\text{WHERE FIRST.CITY = SECOND.CITY AND FIRST.S# < SECOND.S#;} \\
  \end{aligned}
  \]

- **Tuple calculus:**
  
  \[
  \begin{aligned}
  \text{FIRSTS#} &= SX.S#, \text{SECONDS#} = SY.S# \\
  \text{WHERE SX.CITY} &= \text{SY.CITY AND SX.S#} < \text{SY.S#} \\
  \end{aligned}
  \]

- **Algebra:**
  
  \[
  \begin{aligned}
  &\Pi_{\text{FIRSTS#,SECONDS#}} (\sigma_{\text{FIRSTS#} < \text{SECONDS#}} \\
  &\left( (\prod_{\text{FIRSTS#,CITY}} (S \text{ RENAME S# AS FIRSTS#})) \bowtie_{\text{city}=\text{city}} \\
  &\left( (\prod_{\text{SECONDS#,CITY}} (S \text{ RENAME S# AS SECONDS#}))) \right)) \\
  \end{aligned}
  \]

Output:

\[
\begin{align*}
\{S1, S1\} \\
\{S1, S4\} \\
\{S4, S1\} \\
\{S4, S4\}
\end{align*}
\]
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
  (EXISTS SPX
  (SPX.S# = SX.S# AND SPX.P# = PX.P#))
  ```

- **Algebra:**
  
  \[
  \Pi_{\text{SNAME}} \left( \left( (\Pi_{\text{S#,P#}} \text{SP}) \div (\Pi_{\text{P#}} \text{P}) \right) \bowtie S \right)
  \]
  
  \[\text{A} \quad \text{B} \]
  
  \begin{tabular}{|c|c|}
  \hline
  S1 & \text{Smith} \\
  \hline
  \end{tabular}
[Example 4]: Get part numbers for parts that either weigh more than 16 pounds or are supplied by supplier S2, or both.

- **SQL:**
  ```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:**
  $$\left( \Pi_{P#} (\sigma_{\text{WEIGHT}>16} P) \right) \cup \left( \Pi_{P#} (\sigma_{S#='S2'} SP) \right)$$
# Relational Calculus v.s. Relational Algebra.

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

**Relational Calculus ≡ Relational Algebra**

- Codd's reduction algorithm:
  1. Show that any calculus expression can be reduced to an algebraic equivalent.

      \[ \downarrow \]

      Algebra \supseteq \text{Calculus} 

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

      \[ \downarrow \]

      Calculus \supseteq \text{Algebra} 

\[ \downarrow \]

Algebra \equiv \text{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. $PX$.

<table>
<thead>
<tr>
<th>Var.</th>
<th>SX</th>
<th>PX</th>
</tr>
</thead>
<tbody>
<tr>
<td>S5</td>
<td></td>
<td>P9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S#</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
</tr>
<tr>
<td>S2</td>
</tr>
<tr>
<td>S3</td>
</tr>
<tr>
<td>S4</td>
</tr>
</tbody>
</table>

  e.g.: $S\#$ Domain
  =\{S1, S2, ..., S100\}
  $S\#$ Range
  =\{S1, S2, S3, S4\}

<table>
<thead>
<tr>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>S#</td>
</tr>
<tr>
<td>P#</td>
</tr>
<tr>
<td>QTY</td>
</tr>
<tr>
<td>-----</td>
</tr>
</tbody>
</table>
Domain Calculus: attributes WHERE membership_condition

- Domain Calculus expressions:
  - e.g.1 SX
    (i.e. all possible values of supplier number)
  - e.g.2 SX WHERE S(S#:SX)
    (i.e. all S# in relation S)
  - 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'
  - 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:**
    
    ```sql
    SX.S#
    WHERE SX.CITY = 'Paris'
    AND SX.STATUS > 20
    ```
  - **Domain calculus:**
    
    ```sql
    SX
    WHERE EXISTS STATUSX
    (STATUSX > 20) AND
    S(S#:SX, STATUS:STATUSX, CITY:'Paris')
    ```
  - **QBE:**
    
    | S | S# | SNAME | STATUS | CITY  |
    |---|----|-------|--------|-------|
    | P. |    |       | >20    | “Paris” |
Query-by-Example (cont.)

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

- **SQL**:  
  ```sql
  SELECT FIRST.S#, SECOND.S#
  FROM S FIRST, S SECOND
  WHERE FIRST.CITY = SECOND.CITY AND FIRST.S# < SECOND.S#;
  ```

- **Tuple calculus**:  
  ```plaintext
  FIRSTS# = SX.S#, SECONDS# = SY.S#
  WHERE SX.CITY = SY.CITY AND SX.S# < SY.S#
  ```

- **Domain calculus**:  
  ```plaintext
  \{S_1, S_4\}  \quad \text{FIRSTS# = SX, SECONDS# = SY}
  \{S_2, S_3\}  \quad \text{WHERE EXISTS CITYZ}
  \quad (S(S#:SX,CITY:CITYZ) \text{ AND } S(S#.SY,CITY:CITYZ) \text{ AND } SX<SY)
  ```

- **QBE**:  
  ```plaintext
  \_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
  \[ Optimization \]
  More efficient algebraic expression
  \[ \text{Evaluated by the already implemented algebraic operations} \]
  \[ \text{Evaluated by the already implemented algebraic operations} \]
  Result

(p. 3-47)
more in Unit 11

Unit 11
e.g. Join
Concluding Remarks (cont.)

- A spectrum of data management system:

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

<table>
<thead>
<tr>
<th>S</th>
<th>M</th>
<th>Relational Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>I</td>
<td>Tabular</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>(Minimally) Relational</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>Relationally complete</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>Fully Relational</td>
</tr>
</tbody>
</table>
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));
SQL vs. Relational Operators

- A SQL SELECT contains several relational operators.

\(<\text{e.g.}>\>

SQL:

\[
\text{SELECT } S\#, \text{ SNAME} \\
\text{FROM } S, SP \\
\text{WHERE } S.S\# = SP.S\# \\
\text{AND } \text{CITY} = 'London' \\
\text{AND } \text{QTY} > 200
\]

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

- BNF \((p. 3-44)\)