An Introduction to Database Systems

Chapter 6. Relations

spark@dblab.sogang.ac.kr

6.2 Tuples

- Given a collection of types Ti (i=1,2,…,n), not necessary all distinct, a tuple value on those types – t – is a set of ordered triples of the form <Ai, Ti, vi>, where Ai is an attribute name, Ti is a type name, and vi is a value of type ti, and :
  - The value n is the degree or arity of t.
  - The ordered triple <ai, Ti, vi> is a component of t.
  - The ordered pair <ai, Ti> is an attribute of t, and it is uniquely identified by the attribute name Ai. The value of vi is the attribute value for attribute Ai of t.
  - The complete set of attributes is the heading of t.
  - The tuple type of t is determined by the heading of t, and the heading and that tuple type both have the same attributes and the same degree as t does.

spark@dblab.sogang.ac.kr

6.2 Tuples

- The tuple type name is precisely :
  - TUPLE { A1 T1, A2 T2, … , An Tn }
  - heading

MAJOR_P#: P#	MINOR_P#: P#	QTY: QTY
P2	P4	7

  - Type
    - Tuple { MAJOR_P# P#, MINOR_P# P#, QTY QTY }

spark@dblab.sogang.ac.kr

6.2 Tuples

- **Properties of Tuples**
  - **Every tuple contains exactly one value(of the appropriate type) for each of its attributes**
  - **There is no left-to-right ordering to the components of a tuple. (a set of components)**
  - **Every subset of a tuple is a tuple (and every subset of a heading is a heading).**
- **The TUPLE Type Generator**
  - TUPLE { <ATTRIBUTE COMMALIST> }
  - VAR ADDR TUPLE

```
VAR ADDR TUPLE { STREET    CHAR,
                 CITY      CHAR,
                 STATE     CHAR,
                 ZIP       CHAR } ;
```


spark@dblab.sogang.ac.kr

6.2 Tuples

- **EX) tuple selector;**
  - TUPLE { STREET '1600 Pennsylvania Ave.' , CITY 'Washington' , STATE 'DC' , ZIP '20500' }

Operators on Tuples

- **Tuple equality: Tuples t1 and t2 are equal if and only if ey have the same attributes A1, A2, …, An and , for all I (i=1,2,, …, n), the value v1 of Ai in t1 is equal to the value v2 of Ai in t2.**
- **Tuple projection**
  - **ADDR { CITY, ZIP } => TUPLE { CITY 'Washington', ZIP '20500' }**
- **Extraxt**
  - **ZIP FRM ADDR => '20500'**
- **Tuple type inference – determine the type of the result**
  - **ADDR { CITY, ZIP } => TUPLE { CITY CHAR, ZIP CHAR }**

spark@dblab.sogang.ac.kr

6.2 Tuples

- WRAP and UNWRAP
  - TUPLE { NAME NAME, ADDR TUPLE { STREET CHAR, CITY CHAR, STATE CHAR, ZIP CHAR } }   ----- TYPE TT1
  - TUPLE { NAME NAME, STREET CHAR, CITY CHAR, STATE CHAR, ZIP CHAR } }   ----------- TYPE TT2
  - Let NADDR1 and NADDR2 be tuple var of types TT1 and TT2
  - NADDR1 := NADDR2 WRAP { STREET, CITY, STATE, ZIP } AS ADDR ; (The result of the expression is of type TT1)
  - NADDR1 UNWRAP ADDR (--- TYPE TT2)
- Tuple Types vs. Possible Representations

6.3 Relation Types(1/8)

- **relation variable**
  - **variable in the usual programming language sense**
- **relation value**
  - **value of such a variable at any given time**
- **definition of relation (a relation value)**
  - **Given a collection of n types *Ti* (i=1, 2, … , n), not necessarily all distinct, r is a <u>relation</u> on those types if it consists of two parts, a *heading* and a *body*, where :**
    a. **The <u>heading</u> is a set of n <u>attributes</u> of the form *Ai:Ti*, where *Ai* (which must all be distinct) are the *attribute names* of r and the *Ti* are the corresponding *type names* (i=1, 2, …, n) ;**
    b. **The <u>body</u> is a set of m <u>tuples</u> t, where t in turn is a set of components of the form *Ai:vi* in which *vi* is a value of type *Ti* – the *attribute value* for attribute *Ai* of tuple t (i=1, 2, …, n) ;**
  - **The values m and n are called the cardinality and the degree, respectively, of relation r.**

spark@dblab.sogang.ac.kr

6.3 Relation Types(2/8)

- in fig 5.1,
  - four underlying types : S#, NAME, STATUS, CITY
  - table have two parts :
    - row of column headings , set of data rows

- Set of ordered pairs

(S#, SNAME, STATUS, CITY)

{ S# : S#,
SNAME : NAME,
STATUS : STATUS,
CITY : CITY }

(S1, Smith, 20, London)

{ S# : S# (‘S1’),
SNAME : NAME (‘Smith’),
STATUS : 20,
CITY : ‘London’ }

spark@dblab.sogang.ac.kr

6.3 Relation Types(3/8)

- **Relation type name**
  - **RELATION { A1 T1, A2 T2, … , An Tn }**
- "Nondistinct types"

PART-STRUCTURE (MAJOR_P#, MINOR_P#, QTY)

MAJOR_P#: P#	MINOR_P#: P#	QTY: QTY
P1	P2	5
P1	P3	3
P2	P3	2
P2	P4	7
P3	P5	4
P4	P6	8

spark@dblab.sogang.ac.kr

6.4 Relation Values

- **The table in Fig. 5.1 can indeed be regarded as a picture of a relation – provided we can agree on how to read such a picture (rules of interpretation)**
- **Rules of interpretation**
  - **There are some underlying types**
  - **Each column corresponds to exactly one of those types**
  - **Each row represents a tuple**
  - **Each attribute value is a value of the relevant type**

  If we can agree on all of these "rules of interpretation" then we can agree that a "table" is a reasonable picture of a relation
- **A Relation is what the definition says it is, namely a abstract kind of object, and a table is a concrete picture of such an abstract object. They are not quite the same. Tabular representation suggests some things that are not true.**

Properties of relations

6.4 Relation Values

- **Properties of Relations**
  1. **There are no duplicate tuples**
     - **the body of relations is mathematical set of tuples**
     - **SQL permit tables to contain duplicate rows**

     (relation and table are not the same thing)
     - **corollary**
       - ***There is always a primary key***

       (Since tuples are unique, at least, the combination of all attributes of the relations has the uniqueness property)
  2. **Tuples are unordered(top to bottom)**
     - **mathematical set property**
     - **reverse sequence of relation : the same relation**
     - **no such things "the fifth tuple", "the 97th tuple", "the next tuple"**
     - **no concept of *positional addressing* and "*nextness*"**
     - **The rows of a table do have a top-to-bottom ordering, whereas the tuples of a relation do not. (*a relation and a table are not the same thing*)**

spark@dblab.sogang.ac.kr

6.4 Relation Values

- Properties of relations (cont.)
  3. Attributes are unordered(left to right)
     - heading of a relation is also defined as a set of attributes
     - no such things "the first attribute",...
     - no concept of "*nextness*"
       - attributes are always referenced by name never by position
     - The columns of a table obviously do have a left-to-right ordering, but the attributes of a relation do not
  4. Each value contains exactly one value for each attribute
     - A tuple is a set of n components or ordered pairs of the form *Ai:vi* (i=1, 2, …, n) (by definition)
     - A relation that satisfies this property is said to be normalized (first normal form (1NF))

spark@dblab.sogang.ac.kr

6.4 Relation Values

- Relation-Valued Attributes
  - The values that make up a type can be of any kind at all. We can have a type whose values are relations

<table>
<tr><th>S#</th><th>SNAME</th><th>STATUS</th><th>CITY</th><th>PQ</th></tr>
<tr><td>S1</td><td>Smith</td><td>20</td><td>London</td><td><table><tr><th>P#</th><th>QTY</th></tr><tr><td>P1</td><td>300</td></tr><tr><td>P2</td><td>200</td></tr><tr><td>…</td><td>…</td></tr><tr><td>P6</td><td>100</td></tr></table></td></tr>
<tr><td>S2</td><td>Jones</td><td>10</td><td>Paris</td><td><table><tr><th>P#</th><th>QTY</th></tr><tr><td>P1</td><td>300</td></tr><tr><td>P2</td><td>400</td></tr></table></td></tr>
<tr><td>…</td><td>…</td><td>…</td><td>…….</td><td>……………….</td></tr>
<tr><td>S5</td><td>Adams</td><td>30</td><td>Athens</td><td><table><tr><th>P#</th><th>QTY</th></tr><tr><td></td><td></td></tr></table></td></tr>
</table>

spark@dblab.sogang.ac.kr

6.4 Relation Values

- **Relation and Their Interpretation**
  - **(a) The heading of any given relation can be regarded as a predicate, and (b) the tuples of that relation can be regarded as true propositions, obtained from the predicate by substituting values of the appropriate type for the parameters of that predicate.**
  - **Closed World Assumption :**
    - **If a valid tuple does not appear in the body of the relation, then we can assume the corresponding proposition is false.**

spark@dblab.sogang.ac.kr

6.5 Relation Variables(1/6)

- **Base Relvar Definition**
  - **syntax :**

VAR <relvar name> BASE <relation type>
<candidate key definition list>
[<foreign key definition list>] ;

<relation type> :
RELATION { <attribute commalist> }

<attribute> :
<attribute name> <type name>

spark@dblab.sogang.ac.kr

6.5 Relation Variables(2/6)

- Example for the suppliers and parts database

```
VAR      S   BASE    RELATION
         {  S#       S#,
            SNAME    NAME,
            STATUS   INTEGER,
            CITY     CHAR  }
         PRIMARY KEY { S# } ;
```

```
VAR      P   BASE    RELATION
         {  P#       P#,
            PNAME    NAME,
            COLOR    COLOR,
            WEIGHT   WEIGHT,
            CITY     CHAR  }
         PRIMARY KEY { P# } ;
```

```
VAR      SP   BASE    RELATION
         { S#         S#,
           P#         P#,
           QTY        QTY  }
         PRIMARY KEY { S#,  P# }
         FOREIGN KEY { S# } REFERENCES S
         FOREIGN KEY { P# } REFERENCES P;
```


spark@dblab.sogang.ac.kr

6.5 Relation Variables(3/6)

- Explanation :
  1. types of these 3 base relvars
     RELATION is a type generator
  2. All possible relation values are of the same relation type
  3. All interpretation
  4. Cataloging
  5. Candidate key definition
  6. Foreign key definition

- DROP VAR < relvar name >

spark@dblab.sogang.ac.kr

5.4 Relation Variables(4/6)

- **Updating Relvar**

  <relvar name> := <relational expression> ;

  Tutorial D syntax

VAR R BASE RELATION
{ S# S#, SNAME NAME, STATUS INTEER, CITY CHAR } … ;

- **R := S ;**
- **R := S WHERE CITY = 'London' ;**
- **R := S MINUS (S WHERE CITY = 'Paris') ;**
- **S := S WHERE CITY = 'London' ;**
- **S := S MINUS (S WHERE CITY = 'Paris') ;**

5.4 Relation Variables(5/6)

- INSERT INTO S

```
RELATION { TUPLE { S# S# ( 'S6' ),
                   SNAME NAME ( 'Smith' ),
                   STATUS 50,
                   CITY 'Rome ' } } ;
```

```
S := S UNION
     RELATION { TUPLE { S# S# ( 'S6' ),
                        SNAME NAME ( 'Smith' ),
                        STATUS 50,
                          CITY 'Rome ' } } ;
```


spark@dblab.sogang.ac.kr

5.4 Relation Variables(6/6)

- **DELETE S WHERE CITY = 'Paris' ;**

S := S MINUS (S WHERE CITY = 'Paris') ;

- **UPDATE S WHERE CITY = 'Paris'**

  STATUS := 2 * STATUS,

  CITY := 'Rome' ;

The assignment equivalent see reference [3.3].

- **syntax of INSERT, DELETE and UPDATE :**
  - **INSERT INTO <relvar name> <relational expression> ;**
  - **DELETE <relvar name> [WHERE <boolean expression>] ;**
  - **UPDATE <relvar name> [WHERE <boolean expression>]**

    <attribute update commalist> ;

spark@dblab.sogang.ac.kr

6.6 SQL Facilities

- **Rows**
- **SQL does not support tuples; it supports rows, which hve a left-to-right ordering to their components.**
  - **ROW (P# ('P2'), P# ('P4'), QTY (7))**
  - **ROW type constructor (its counterpart to TUPLE type generator)**
- **Table Types**
  - **SQL does not support relations; it supports tables.**
- **Table Values and variables**

spark@dblab.sogang.ac.kr

6.6 SQL Facilities

- Table Values and Variables
  1) SQL tables are allowed to include duplicate rows (not necessarily a primary key)
  2) SQL tables are considered to have a left-to-right column ordering

- Base Table :

  CREATE TABLE <base table name>

  (<base table element commalist>) ;

  - <base table element> is either a <column definition> or a <constraint>
  - <column definition>
    - <column name> <type name> [<default spec>]
  - <default spec> defines the default value or default
  - NULL : default default
  - Fig. 4.1

spark@dblab.sogang.ac.kr

6.6 SQL Facilities

□ DROP TABLE

DROP TABLE <base table name> <option> ;

- option : RESTRICT or CASCADE

1) If RESTRICT is specified and the base table is referenced in an view definition or integrity constraint, the DROP will fail.
2) If CASCADE is specified, the DROP will succeed (dropping the table and deleting all of its rows), and any referencing view definitions and integrity constraints will be dropped also.

spark@dblab.sogang.ac.kr

6.6 SQL Facilities

- **ALTER TABLE statement :**
  - A new column can be added;
  - A new default can be defined for an existing column;
  - An existing column default can be deleted;
  - An existing column can be deleted;
  - A new integrity constraint can be specified;
  - An existing integrity constraint can be deleted

ex) ALTER TABLE S ADD COLUMN DISTINCT INTEGER DEFALUT -1 ;

spark@dblab.sogang.ac.kr

6.6 SQL Facilities

- Structured Types
  - CREATE TYPE POINT AS (X FLOAT, Y FLOAT) NOT FINAL ;
  - CREATE TABLE NADDR

```
( NAME ... ,
  ADDR ... ,
  LOCATION POINT ...,
  ... )
SELECT   NT.LOCATION.X, NT.LOCATION.Y
FROM     NADDR AS NT
WHERE    NAME = ... ;

UPDATE   NADDR AS NT
SET      NT.LOCATION.X = 5.0
WHERE    NAME = ... ;
```


spark@dblab.sogang.ac.kr

6.6 SQL Facilities

```
CREATE  TYPE POINT1 AS ( X FLOAT, Y FLOAT ) NOT FINAL ;
CREATE  TYPE POINT2 AS ( X FLOAT, Y FLOAT ) NOT FINAL ;
DECLARE V1 POINT1 ;
DECLARE V2 POINT2 ;
```

variables V1 and V2 are of different types.

spark@dblab.sogang.ac.kr