An Introduction to Database Systems

Chapter 9. Integrity

spark@dblab.sogang.ac.kr

9.1 Introduction (1/3)

- **Integrity**
  - **the accuracy or correctness of data in database**
  - **ex)**
    - **supplier number : Snnnn(nnnn=up to four decimal digits) and unique**
    - **status value : 1 – 100**
    - **If the supplier's city is London, then the supplier's status must be 20**
    - **shipment quantity : multiple of 100**
    - **all red parts : stored in London**

spark@dblab.sogang.ac.kr

9.1 Introduction (2/3)

- **DBMS needs to be informed of such constraints, and needs to enforce them**

  ex) constraint SC3
  IS_EMPTY (S WHERE STATUS < 1 OR STATUS > 100);

  - **Registered in the system catalog under that name**
  - **That name (SC3) will appear in an diagnostic message**
  - **The constraint itself is specified a boolean expression that is required not to evaluate to false**
  - **Calculus analog :**

  ex) constraint SC3
  FORALL SX (SX.STATUS ≥ 1 AND SX.STATUS ≤ 100);

  DROP CONSTRAINT <constraint name> ;
  ex) DROP CONSTRAINT SC3 ;

spark@dblab.sogang.ac.kr

9.1 Introduction (3/3)

- **Declarative integrity support is important**
  - **vs procedural support using stored or triggered procedures**
- **A Constraint Classification Scheme**
  - **type constraint**
  - **attribute constraint**
  - **relvar constraint**
  - **database constraint**

spark@dblab.sogang.ac.kr

9.2 Type Constraints

- **A type constraint is just an enumeration of the legal values of the type**

  ex) TYPE WEIGHT POSSREP (RATIONAL)
  CONSTRAINT THE_WEIGHT (WEIGHT) > 0.0 ;

- **Type constraints can always be thought of, at least conceptually, as being checked during the execution of some selector invocation**

  ex) TYPE POINT POSSREP CARTESIAN (X RATIONAL, Y RATIONAL)
  CONSTRAINT ABS (THE_X (POINT)) ≤ 100.0 AND
  ABS (THE_Y (POINT)) ≤ 100.0 ;

- **The type checking is done during execution of invocations of the CARTESIAN selector**

spark@dblab.sogang.ac.kr

9.3 Attribute Constraints

- **An attribute constraint is basically just a declaration to the effect that a specified attribute is of a specified type**

```
ex) VAR  S  BASE  RELATION
    {       S#          S#,
            SNAME       NAME,
            STATUS      INTEGER,
            CITY        CHAR } … ;
```


spark@dblab.sogang.ac.kr

9.4 RELVAR Constraints

- **A relvar constraint is a constraint on an individual relvar**

```
ex) CONSTRAINT  SC5
            IS_EMPTY ( S WHERE CITY = 'London' AND STATUS ≠ 20 ) ;
    (" Suppliers in London must have status 20 ")

    CONSTRAINT  PC4
            IS_EMPTY ( P WHERE COLOR = COLOR('Red') )
                        AND CITY ≠ 'London' ) ;
    (" red parts must be stored in London ")

   CONSTRAINT  SCK
            COUNT ( S ) = COUNT ( S {S#} ) ;
    (" Supplier numbers are unique ")

   CONSTRAINT  PC7
            IF NOT ( IS_EMPTY ( P ) ) THEN
              COUNT ( P WHERE  COLOR = COLOR ( 'Red' ) ) > 0
          END IF
    (" If there are an parts at all, at least one of them must be red ")
```

- **Relvar constraints are always checked immediately**

spark@dblab.sogang.ac.kr

9.5 Database Constraints

- **A database constraint is a constraint that interrelates two or more distinct relvars**

  ex) CONSTRAINT DBC1
  IS_EMPTY (S JOIN SP) WHERE STATUS < 20 AND QTY > QTY(500)) ;
  (“ No supplier with status less than 20 can supply any part in quantity greater than 500 “)

  CONSTRAINT DBC2 SP {S#} ≤ S {S#} ;
  (“ every supplier number in the shipment relvar also exists in the supplier relvar “)
  - referential constraint

  CONSTRAINT DBC3 SP {P#} = P {P#} ;
  (“ every part must have at least one shipment “)

- **Database constraint checking cannot be done immediately, but must be deferred to end-of-transaction, i.e., to COMMIT time**
- **If a database constraint is violated at COMMIT time, the transaction is rolled back**

spark@dblab.sogang.ac.kr

9.6 The Golden Rule (1/2)

- **Any given relation – an associated predicate**
- **Tuples – true proposition**
- **Closed World Assumption (CWA)**

  ex) supplier relvar S - predicate
  “ The supplier with the specified supplier number (S#) has the specified name (SNAME) and the specified status value (STATUS), and is located in the specified city (CITY); moreover, no two suppliers have the same supplier number at the same time. “

- **The predicate for a given relvars serves as the *criterion for acceptability of updates* on the relvar**
- **DBMS would know and understand the predicate for every relvar**
  → unachievable

{ S# : S# (‘S1’) , SNAME : NAME (‘Smith’) , STATUS : 20 , CITY : ‘London’ }	{ S# : S# (‘S5’) , SNAME : NAME (‘Smith’) , STATUS : 50 , CITY : ‘Rome’ }
(O)	(X)

spark@dblab.sogang.ac.kr

9.6 The Golden Rule (2/2)

- **DBMS does know a good approximation to that predicate.**
  → define the relvar predicate for the suppliers relvar to be the logical AND of all relvar constraints that apply to that relvar

  ex) (IS_EMPTY (S WHERE STATUS < 1 OR STATUS > 100)) AND
  (IS_EMPTY (S WHERE CITY = 'London' AND STATUS ≠ 20)) AND
  (COUNT (S) = COUNT (S {S#}))

- **There are effectively two predicates associated with any given relvar**
  - **The informal or external predicate, which is understood by both users and the system → "relvar predicate" (the system will check whenever updates are attempted on the relvar)**
- **The golden rule**
  - **No update operation must ever be allowed to leave any relvar in a state that violates its own predicate. Likewise, no update transaction must ever be allowed to leave the database in a stat that violates its own predicate**

spark@dblab.sogang.ac.kr

9.7 State vs. Transition Constraints (1/2)

- All of the constraints discussed so far has been state constraints. They were concerned with correct state of database
- Transition constraints – constraints on legal transitions from one correct state to another

ex) valid	ex) invalid
Never married to married	Never married to widowed
Married to widowed	Never married to divorced
Married to divorced	Widowed to divorced
Widowed to married	Divorced to widowed

ex) " No supplier's status must ever decrease "

```
CONSTRAINT  TRC1  IS_EMPTY
( ( ( S' { S#, STATUS } RENAME AS STATUS' )
      JOIN S { S#, STATUS } )
      WHERE STATUS' > STATUS ) ;
```

- relvar' : prior to the update under consideration
- TRC1 : a relvar transition constraints. The checking is immdiate

spark@dblab.sogang.ac.kr

9.7 State vs. Transaction Constraints (2/2)

ex) " The total quantity of any given part, taken over all suppliers, can never decrease "

```
CONSTRAINT  TRC2  IS_EMPTY
( ( ( SUMMARIZE  SP' PER S' {S#}  ADD SUM ( QTY ) AS SQ' )
      JOIN
       ( SUMMARIZE SP PER S {S#}  ADD SUM ( QTY ) AS SQ ) )
      WHERE  SQ' > SQ ) ;
```

- **TRC2 : a database constraints. The checking is deferred to COMMIT time**
- **The concept of state vs transition constraints has no meaning for type or attribute constraints**

spark@dblab.sogang.ac.kr

9.8 Keys (1/22)

- **Candidate Keys**
  - **Let R be a relvar. By definition, the set of all attributes of R has the uniqueness property, meaning that, at any given time, no two tuples in R at that time are duplicates of one another**
- **Definition of candidate key**
  - **Let K be a set of attributes of relvar R. Then K is a candidate key for R iff it possesses both of the following properties**
    a. Uniqueness : No legal value of R ever contain two distinct tuples with the same value of K
    b. Irreducibility (Minimality) : No proper subset of K has the uniqueness property
- **syntax**

Key { < attribute name commalist > }

ex) VAR S BASE RELATION
{ S# S#,
SNAEM NAME,
STATUS INTEGER,
CITY CHAR }
KEY { S# } ;

spark@dblab.sogang.ac.kr

9.8 Keys (2/22)

- **syntax**

```
ex)  VAR   SP  BASE  RELATION
           { S#          S#,
             P#          P#,
             QTY         QTY  }
           KEY { S#, P# } … ;

     VAR  ELEMENT  BASE  RELATION  { NAME          NAME,
                                      SYMBOL      CHAR,
                                      ATOMIC#     INTEGER }
                                KEY { NAME }
                                KEY { SYMBOL }
                                KEY { ATOMIC# } ;

     VAR  MARRIAGE  BASE  RELATION  { HUSBAND                  NAME,
                                      WIFE                     NAME,
                                      DATE  /* of marriage */  DATE  }
     /* assume no polyandry, no polygyny, and no husband and  */
     /* wife marry each other more than once …                */
                                KEY { HUSBAND, DATE }
                                KEY { DATE, WIFE }
                                KEY { WIFE, HUSBAND } ;
```


spark@dblab.sogang.ac.kr

9.8 Keys (3/22)

- **irreducibility**
  - **reducible candidate key**
    - not be aware of the true state of affairs
    - not be able to enforce the associated integrity constraint properly
  - **foreign key that referenced a "reducible" candidate key would be *"reducible"* too.**
- **no implication that there has to be an index on a candidate key**

spark@dblab.sogang.ac.kr

9.8 Keys (4/22)

- **What are candidate keys for ?**
  - **the basic** *tuple-level addressing mechanism* **in a relational system**

    S WHERE S# = S#('S3‘)

    is guaranteed to yield at most one tuple

    S WHERE CITY = 'Paris'

    will yield an unpredictable number of tuples

spark@dblab.sogang.ac.kr

8.8 Keys (5/22)

- ***The candidate keys are just as fundamental to the successful operation of a relational system as main memory addresses are to the successful operation of the underlying machine***
- ***As a consequence,***
  - **"relation" that do not have a candidate key**
    - display strange and anomalous behavior
  - **a system that has no knowledge of candidate key**
    - display behavior on occasion that is not "truly relational"

spark@dblab.sogang.ac.kr

9.8 Keys (6/22)

- **Points :**
  - **Superkey : uniqueness property but not necessarily irreducibility property**
  - **If SK is a superkey for relvar R and A is an attribute of R, then the Functional dependency SK→A holds true in R**
  - **The logical notion of a candidate key should not confused with the physical notion of a unique index**

spark@dblab.sogang.ac.kr

9.8 Keys (7/22)

- **Primary keys and Alternate Keys**
  - **possible to have more than one candidate key**
  - **primary key**
    - **exactly one of candidate keys be chosen as the primary key**
  - **alternate key**
    - **the remainder, if any**
  - **the choice of which is to be primary**
    - **arbitrary**

spark@dblab.sogang.ac.kr

9.8 Keys (8/22)

- **Foreign Keys**
  - **In suppliers-and-parts database,**
    - **A given value for S# of relation SP should be permitted to appear in the database only if that same value also appears as a value of the primary key S# of relation S**
  - **definition of term foreign key**
    - **Let R2 be a relvar. Then foreign key in R2 is a subset of the set of attributes of R2, say FK, such that**
      - There exists a base relation R1 with a candidate key CK, (R1 and R2 not necessarily distinct)
      - For all time, each value of FK in the current value of R2 is identical to the value of CK in some tuple in the current value of R1

spark@dblab.sogang.ac.kr

9.8 Keys (9/22)

- **Foreign Keys**
  - **points**
    1. **The definition requires every value of a given foreign key to appear as a value of the matching candidate key. The converse of foreign key definition is not a requirements**

       the supplier S5 appears in relvar S but not in relvar SP
    2. **composite foreign key iff the candidate key it matches is composite too**
    3. **the same name and type as the corresponding component of the matching candidate key**

spark@dblab.sogang.ac.kr

9.8 Keys (10/22)

- **points**
  - **4. terminology**
    - referenced tuple(target tuple) :candidate key value
    - referential integrity : valid foreign key value
    - referential constraint
    - referencing relation and referenced relation
  - **5. referential diagram**
    - representing referential constraint

      S <- SP -> P

      - labeling each arrow, sometimes

spark@dblab.sogang.ac.kr

9.8 Keys (11/22)

- **points**
  - **6. referential path**

    A relation can be both a referenced relation and referencing relation as is the case with relation R2

    R3 -> R2 -> R1

    There is a referential constraint form Rn to R(n-1), a referential constraint from R(n-1) to R(n-2).... from R2 to R1.

    Rn -> R(n-1) -> ..... R2 -> R1

spark@dblab.sogang.ac.kr

9.8 Keys (12/22)

- **points**
  - **7. *Self-Referencing***

    Relations R1 and R2 in the foreign key definition are *not necessarily distinct*

```
VAR EMP BASE RELATION
        { EMP# EMP#, ..., MGR_EMP# EMP#,... )
    PRIMARY KEY {EMP#}
    FOREIGN KEY { RENAME MRG_EMP# AS EMP# }
                REFERENCES EMP ;
```


spark@dblab.sogang.ac.kr

9.8 Keys (13/22)

- **points**

  8. referential cycle

  if Rn includes a foreign key referring to R(n-1),

  R(n-1) includes a foreign key referring to R(n-2),

  ,.....,

  R1 includes a foreign key referring to R(n-1).

  Rn -> R(n-1) -> ,...., R1 -> Rn

  9. special way of matching

  relationship "colocation" between S and P, represented by CITY attributes of S and P

  - CITY attributes are not foreign keys

spark@dblab.sogang.ac.kr

9.8 Keys (14/22)

- **referential integrity rule**
  - **The database must not contain any unmatched foreign key values**
    - **If B references A, then A must exist**
    - **"foreign key" and "referential integrity" are defined in terms of each other**

spark@dblab.sogang.ac.kr

9.8 Keys (15/22)

- **Syntax**

FOREIGN KEY { < item commalist > }
REFERENCES <relvar name >
<item> : either <attribute name> of the referencing relvar or an expression
RENAME <attribute name> AS <attribute name>

ex) Fig. 3.9

- **A foreign key definition is really just a shorthand for a certain database constraint unless the foreign key definition is extended to include referential actions**

spark@dblab.sogang.ac.kr

9.8 Keys (16/22)

- **referential Actions**
  - **referential integrity rule**
    - **in terms of database state**
  - **How exactly are such incorrect states to be avoided ?**
    - **The rule itself does not say.**
  - **one possibility**
    - **rejection any operation that would result in illegal state**
    - **certain additional compensating action**

spark@dblab.sogang.ac.kr

9.8 Keys (17/22)

- **referential Actions**
  - **database designer**
    - **specify which operations should be rejected**
    - **specify what compensating operations (if any) should be performed**
  - **What should happen on an attempt to delete the target of a foreign key reference ?**
    - **RESTRICTED : "restricted" to the case where there are no matching.**
    - **CASCADES : " cascades" to delete those matching also**

spark@dblab.sogang.ac.kr

9.8 Keys (18/22)

- **referential Actions**
  - **What should happen on an attempt to update a candidate key that is the target of a foreign key reference ?**
    - **RESTRICTED : " restricted" to the case where there are no matching.**
    - **CASCADES : "cascades" to update the foreign key in those matching also**

spark@dblab.sogang.ac.kr

9.8 Keys (19/22)

- **referential Actions**
  - **for each foreign key in the design,**
    - **not only attribute or attribute combination constituting that foreign key**
    - **but also particular *referential actions***

```
VAR  SP  BASE  RELATION { … } …
        FOREIGN   KEY  { S# }  REFERENCES  S
                                ON DELETE CASCADE ;
```


spark@dblab.sogang.ac.kr

9.8 Keys (20/22)

- **referential actions**
  - **points**

    1.option

    – invoking an installation-defined "*database procedure*" (stored procedure, triggered procedure)

    2. R3 -> R2 -> R1

    referential constriant : CASCADES

    – deletion tuple of R1 imply deletion certain tuple of R2

    – deletion tuple of R2 imply deletion certain tuple of R3

    – If one of DELETEs fails, entire operation fails and database remains unchanged. (atomic operation)

spark@dblab.sogang.ac.kr

9.8 Keys (21/22)

- **referential actions**
  - **points**
    - **3. some constraint checking**
      - cannot be done at the time of individual update
      - but must instead be deferred to some later time (COMMIT time)
      - otherwise, there will be no way to insert the first tuple into the database

spark@dblab.sogang.ac.kr

9.8 Keys (22/22)

- **Triggered Procedures**
  - **A procedure that is invoked automatically on the occurrence of some specified event or trigger condition**
  - **Triggered condition : the occurrence of a specified exception (the violation of a specified integrity constraint), or the passage of a specified interval of time**

    ex) CASCADE referential action
  - **Triggered procedures are applicable to a much wide variety of problems**

spark@dblab.sogang.ac.kr

9.9 SQL Facilities (1/9)

- **SQL's integrity constraints**
  - **Domain constraints**
  - **Base Table constraints**
  - **General constraints (" assertions ")**
- **SQL does not support type constraints at all**
- **SQL's domain constraints are an undesirably generalized form of our attribute constraints**
- **SQL's base table constraints and assertions are loosely equivalent to our relvar and database constraints taken together**

spark@dblab.sogang.ac.kr

9.9 SQL Facilities (2/9)

- **Domain Constraints**
  - **SQL-style domain constraint**

```
CREATE  DOMAIN  COLOR  CHAR(6)  DEFAULT  '???'
        CONSTRAINT   VALID_COLORS
                CHECK ( VALUE  IN
                        ( 'Red', 'Yellow', 'Blue', 'Green', '???' ) ) ;

CREATE  TABLE  P ( … ,  COLOR  COLOR, … ) ;
```


spark@dblab.sogang.ac.kr

9.9 SQL Facilities (3/9)

- **Base Table Constraints**
  - **a candidate key definition**
  - **a foreign key definition**
  - **a check constraint definition**

CONSTRAINT < constraint name >

1. Candidate keys

UNIQUE (< column name commalist>)

or PRIMARY KEY (< column name commalist >)

- at most one PRIMARY KEY

- NOT NULL

spark@dblab.sogang.ac.kr

9.9 SQL Facilities (4/9)

CONSTRAINT < constraint name >

2. Foreign keys

FOREIGN KEY (< column name commalist >)

REFERENCES < base table name > [(< column name commalist >)]

- **Referential action : NO ACTION(default), CASCADE, SET DEFALUT, SET NULL**
- **Not column name but column position**

3. Check constraints

CHECK (< conditional expression >)

- **An attempt to create a row r with base table T is considered to violate a check constraint for T if the conditional expression specified within that constraint evaluate to false for r**

spark@dblab.sogang.ac.kr

9.9 SQL Facilities (5/9)

ex)

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


spark@dblab.sogang.ac.kr

9.9 SQL Facilities (6/9)

- **Assertions**

```
CREATE  ASSERTION  < constraint name >
            CHECK  ( < conditional expression > ) ;

DROP  ASSERTION  < constraint name >
```

1. " Every supplier has status at least five "

```
CREATE  ASSERTION  IC13 CHECK
            ( ( SELECT MIN (S.STATUS ) FROM S ) > 4 ) ;
```

2. " Every part has a positive weight "

```
CREATE  ASSERTION  IC18 CHECK
            ( NOT EXISTS ( SELECT * FROM P
                        WHERE NOT ( P.WEIGHT > 0.0 ) ) ) ;
```


spark@dblab.sogang.ac.kr

9.9 SQL Facilities (7/9)

3. “ All red parts must be stored in London “

```
CREATE  ASSERTION  IC99 CHECK
        ( NOT EXISTS ( SELECT * FROM P
                       WHERE P.COLOR = ‘Red’
                       AND  P.CITY<> ‘London’ ) ) ;
```

4. “ No shipment has a total weight greater that 20,000 “

```
CREATE  ASSERTION  IC46 CHECK
        ( NOT EXISTS ( SELECT * FROM P, SP
                       WHERE P.P# = SP.P#
                       AND ( P.WEIGHT * SP.QTY ) > 20000.0 ) );
```


spark@dblab.sogang.ac.kr

9.9 SQL Facilities (8/9)

5. “ No supplier with status less than 20% can supply any part in a quantity grater than 5000 “

```
CREATE  ASSERTION  IC95 CHECK
        ( NOT EXISTS ( SELECT * FROM S, P
                       WHERE P.P# = S.STAUTS < 20
                       AND    S.S# = SP.S#
                       AND    SQ.QTY < 500 ) ) ;
```


spark@dblab.sogang.ac.kr

9.9 SQL Facilities (9/9)

- **Deferred Checking**
  - **Our scheme : database constraints are checked at COMMIT time, others are checked immediately**
  - **SQL : DEFERABLE / NOT DEFERABLE**
    - **If a given constraint is DEFERABLE, it can be defined to be INITIALLY DEFERABLE, or INITIALLY IMMEDIATE, which defines its state at beginning of each transaction**
    - **DEFERALBE constraints can be dynamically switched on and off**

SET CONSTRAINTS < constraint name commalist > < option > ;

ex) SET CONSTRAINT IC46, IC95 DEFERRED ;

- **NOT DEFERABLE constraints are always checked immediately**
- **If any integrity check fails, the transaction is rolled back**

spark@dblab.sogang.ac.kr