Information Systems

Relational Databases

Temur Kutsia

Research Institute for Symbolic Computation
Johannes Kepler University of Linz, Austria
kutsia@risc.uni-linz.ac.at

Outline

Outline

Integrity Constraints

- Integrity Constraint: A boolean expression that is associated with some database and is required to evaluate at all times to TRUE.

Integrity Constraints. Examples

Suppliers-and-parts database satisfy the constraints:

- Every supplier status value is in the range 1 to 100 inclusive.
- Every supplier in London has status 20.
- If there are any parts at all, at least one of them is blue.
- No two distinct suppliers have the same supplier number.
- etc.

All examples today from suppliers-and-parts database.

Integrity Constraints

- Constraints must be formally declared to the DBMS and DBMS must enforce them.
- Declaring constraints is a matter of using relevant features of the database language.
- Enforcing them is a matter of the DBMS monitoring updates that might violate the constraints and rejecting those that do.

Example

To enforce the constraint *Every supplier status value is in the range 1 to 100 inclusive*, the DBMS will have to monitor all operations that attempt

- to insert a new supplier, or
- change an existing supplier's status.

Integrity Constraints

- Overall “shape” of integrity constraints:

  IF
  a certain tuple appears in certain relvars,
  THEN
  that tuple satisfies a certain condition.

Classification of Constraints

- Type constraint: A definition of the set of values that constitute a given type.
- Attribute constraint: Constrains values a given attribute is permitted to assume.
- Relvar constraint: Constrains values a given relvar is permitted to assume.
- Database constraint: Constrains values a given database is permitted to assume.

Type Constraints

- Definition of the set of values that constitute a given type.
- Example:

  ```
  TYPE WEIGHT POSSREP { D DECIMAL (5,1) }
          CONSTRAINT D > 0.0 AND D < 5000.0
  ```
- Meaning:
  Legal values of type WEIGHT are precisely those
  - that can possibly be represented by decimal numbers of five digits precision with one digit after the decimal point,
  - where the decimal number in question is greater than zero and less than 5000.
- Type constraints are thought of being checked during the execution of some selector invocation.
- WEIGHT (7500.0) will raise an exception at run time (value out of range).

Attribute Constraints

- Declaration to the effect that a specified attribute of a specified relvar is of a specified type.
- Example:

```
VAR S BASE RELATION
 {   S#         S#
     SNAME      NAME
     STATUS     INTEGER
     CITY       CHAR } . . . ;
```

- Part of the attribute definition itself, can be identified by the corresponding attribute name.

Relvar Constraints

- Constrain possible values of a given relvar.
- Example:
  - Every supplier status value is in the range 1 to 100 inclusive.
- For all supplier numbers *s#*, all names *sn*, all integers *st* and all character strings *sc*:
  - IF a tuple with S# *s#*, SNAME *sn*, STATUS *st*, and CITY *sc* appears in the suppliers relvar S,
  - THEN *st* is greater than or equal to 1 and less than or equal to 100.
- Constraint for S.

Relvar Constraints

- Any given relvar can be subject to many constraints.
- Example:
  - Every supplier status value is in the range 1 to 100 inclusive.
  - No two distict suppliers have the same supplier number.
- **The** relvar constraint: Conjunction of all constraints for the relvar.
- Golden Rule:
  - No update operation must ever assign to any relvar R a value that causes the constraint for R to evaluate to FALSE.

Database Constraints

- Database constraint: Conjunction of all the relvar constraints for all relvars contained in the database.
- Golden Rule:
  - No update operation must ever assign to any database a value that causes even the database constraint to evaluate to FALSE.

Integrity and Views

- Constrained relvars can be both base relvars and views.
- If a view R_V is derived from a base relvar R_B, then a constraint for R_V can be derived from the corresponding constraint for R_B just at R_V is derived from R_B.

Integrity and Views

Example

Let SST be a view obtained by projecting S over S#, SNAME, and STATUS:

S

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

SST

S#	SNAME	ST
S1	Smith	20
S2	Jones	10
S3	Blake	30
S4	Clark	20
S5	Adams	30

- Constraint: Every supplier status value is in the range 1 to 100 inclusive.

Integrity and Views

Example

Let SST be a view obtained by projecting S over S#, SNAME, and STATUS:

S

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

SST

S#	SNAME	ST
S1	Smith	20
S2	Jones	10
S3	Blake	30
S4	Clark	20
S5	Adams	30

- Constraint: Every supplier status value is in the range 1 to 100 inclusive.
- For S: For all supplier numbers *s#*, all names *sn*, all integers *st* and all character strings *sc*:
    - IF a tuple with S# *s#*, SNAME *sn*, STATUS *st*, and CITY *sc* appears in the relvar S, THEN $1 \leq st \leq 100$.

Integrity and Views

Example

Let SST be a view obtained by projecting S over S#, SNAME, and STATUS:

S

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

SST

S#	SNAME	ST
S1	Smith	20
S2	Jones	10
S3	Blake	30
S4	Clark	20
S5	Adams	30

- Constraint: Every supplier status value is in the range 1 to 100 inclusive.
- For SST: For all supplier numbers *s#*, all names *sn*, and all integers *st*:
  - IF a tuple with S# *s#*, SNAME *sn*, STATUS *st*, appears in the relvar SST, THEN *st* $1 \leq st \leq 100$.

Keys

Candidate Key

- Let K be a set of attributes of relvar R. Then K is a candidate key for R iff it has both of the following properties:
  - Uniqueness: No legal value of R ever contains two distinct tuples with the same value for K.
  - Irreducibility: No proper subset of K has the uniqueness property.

- Every relvar has at least one candidate key.
- candidate keys do not include any attributes that are irrelevant for unique identification purposes.

Keys

Example

Examples of Candidate Keys.

- VAR S BASE RELATION
  ```
  {   S#        S#
      SNAME     NAME
      STATUS    INTEGER
      CITY      CHAR }
  KEY { S# }
  ```
  Simple candidate kay.
- VAR SP BASE RELATION
  ```
  {   S#     S#
      P#     P#
      QTY    QTY }
  KEY { S#, P# }
  ```
  Composite candidate key.

Keys

Example

Examples of Candidate Keys.

- Several candidate keys are possible

```
VAR MARRIAGE BASE RELATION
 {   HUSBAND    NAME
     WIFE       NAME
     DATE       DATE }
 KEY { HUSBAND, DATE }
 KEY { DATE, WIFE }
 KEY { WIFE, HUSBAND }
```

Keys

- A candidate key definition is a shorthand for a certain relvar constraint.

Example

- { S# } is a candidate key.
- Corresponding constraint: No two distinct suppliers have the same supply number.
- A bit more formally:
  For all supplier numbers *x#* and *y#*, all names *xn* and *yn*, all integers *xt* and *yt*, and all character strings *xc* and *yc*:
  - IF tuples with

    S# *x#*, SNAME *xn*, STATUS *xt*, CITY *xc* and

    S# *y#*, SNAME *yn*, STATUS *yt*, CITY *yc*

    appear in the suppliers relvar S,
  - THEN IF *x#*$=$ *y#*

    THEN $xn = yn$, $xt = yt$, and $xc = yc$.

Keys

- A given relvar can have two or more candidate keys.
- Exactly one of those keys (at least for base relvars) are chosen as the primary key.
- The others are called alternate keys.

Keys

- A foreign key in a relvar R_2 is a set of attributes of R_2, say FK, such that:
  - There exists a relvar R_1 (R_1 and R_2 not necessarily distinct) with a candidate key CK.
  - Each value of FK (or a renamed copy of FK) in the current value of R_2 is identical to the value of CK in some tuple in the current value of R_1.

Keys

- A foreign key in a relvar R_2 is a set of attributes of R_2, say FK, such that:
  - There exists a relvar R_1 (R_1 and R_2 not necessarily distinct) with a candidate key CK.
  - Each value of FK (or a renamed copy of FK) in the current value of R_2 is identical to the value of CK in some tuple in the current value of R_1.
- Points:
  - Every value of FK must appear as a value of CK, the converse is not necessary.

Keys

- A foreign key in a relvar R_2 is a set of attributes of R_2, say FK, such that:
  - There exists a relvar R_1 (R_1 and R_2 not necessarily distinct) with a candidate key CK.
  - Each value of FK (or a renamed copy of FK) in the current value of R_2 is identical to the value of CK in some tuple in the current value of R_1.
- Points:
  - Every value of FK must appear as a value of CK, the converse is not necessary.
  - FK is simple or composite according as CK is simple or composite.

Keys

- A foreign key in a relvar R_2 is a set of attributes of R_2, say FK, such that:
    - There exists a relvar R_1 (R_1 and R_2 not necessarily distinct) with a candidate key CK.
    - Each value of FK (or a renamed copy of FK) in the current value of R_2 is identical to the value of CK in some tuple in the current value of R_1.
- Points:
    - Every value of FK must appear as a value of CK, the converse is not necessary.
    - FK is simple or composite according as CK is simple or composite.
    - An FK value represents a reference to the tuple containing the matching CK value (the referenced tuple).

Keys

- A foreign key in a relvar R_2 is a set of attributes of R_2, say FK, such that:
  - There exists a relvar R_1 (R_1 and R_2 not necessarily distinct) with a candidate key CK.
  - Each value of FK (or a renamed copy of FK) in the current value of R_2 is identical to the value of CK in some tuple in the current value of R_1.
- Points:
  - Every value of FK must appear as a value of CK, the converse is not necessary.
  - FK is simple or composite according as CK is simple or composite.
  - An FK value represents a reference to the tuple containing the matching CK value (the referenced tuple).
  - The constraint that values of FK must match the values of CK is known as referential constraint.

Keys

- A foreign key in a relvar R_2 is a set of attributes of R_2, say FK, such that:
  - There exists a relvar R_1 (R_1 and R_2 not necessarily distinct) with a candidate key CK.
  - Each value of FK (or a renamed copy of FK) in the current value of R_2 is identical to the value of CK in some tuple in the current value of R_1.
- Points:
  - Every value of FK must appear as a value of CK, the converse is not necessary.
  - FK is simple or composite according as CK is simple or composite.
  - An FK value represents a reference to the tuple containing the matching CK value (the referenced tuple).
  - The constraint that values of FK must match the values of CK is known as referential constraint.
  - R_2 is the referencing relvar and R_1 is a referenced relvar.

Keys

- Referential constraints in the suppliers-and-parts database can be represented by means of the referential diagram: S $\leftarrow_{\text{S\#}}$ SP $\rightarrow_{\text{P\#}}$ P.
- A given relvar can be both referenced and referencing: $R_n \rightarrow R_{n-1} \rightarrow \cdots \rightarrow R_1$.
  Referential chain from R_n to R_1.
- A relvar might include a foreign key whose values are required to match the values of some candidate key in the same relvar (self-referencing):

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

- Referencing cycles are allowed.

Keys

- The relational model includes the following rule:

  Referential Integrity: The database must not contain any unmatched foreign key values.

Keys

- A foreign key definition is a shorthand for a certain relvar constraint:

Example

- { S# } is a foreign key for shipments.
- Constraint: Every shipment involves an existing supplier (supplier-and-parts db).
- A bit more formally:
  For all supplier number *s#*, all part number *p#*, and all integers *q*:
  - IF a tuple with S# *s#*, P# *p#*, QTY *q* appears in the shipments relvar SP,
  - THEN there exists a name *sn*, an integer *st*, and a character string *sc* such that the tuple with S# *s#*, SNAME *sn*, STATUS *st*, CITY *sc* appears in the suppliers relvar S.

Referential Actions

Example

- DELETE S WHERE S# = S# ('S1') ;
- Deletes supplier tuple for S1.
- Assume:
  - The database includes some shipments for S1
  - The application does not delete those shipments.
- Then the system raises an exception when it checks the referential constraint from shipments to suppliers.

Alternative approach possible.

Referential Actions

Example

Alternative Approach

- DELETE S WHERE S# = S# ('S1') ;
- Deletes supplier tuple for S1.
- Alternative approach: If the database includes some shipments for S1, delete those shipments as well.
- The effect achieved by extending the foreign key definition:

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

- ON DELETE CASCADE defined a delete rule for the foreign key.
- CASCADE: referential action. The DELETE operation on S will cascade to delete matching tuples in the shipments relvar as well.

Referential Actions

- RESTRICT: Another common referential action.
- In case of DELETE (suppliers-and-parts db), RESTRICT would mean that the operation is restricted to the case when there are no matching shipments.
- Otherwise, the operations are rejected.
- Omitting a referential action for a particular foreign key is equivalent to specifying to NO ACTION.
- Referential actions make sense also for UPDATE.

Triggers

- A trigger is a statement (in the query language) the DBMS executes automatically whenever a set of conditions becomes true.

Example

- Let LONDON_SUPPLIER be a view:

```
CREATE VIEW LONDON_SUPPLIER
        AS SELECT S#, SNAME, STATUS
                FROM S
                WHERE CITY = 'London' ;
```

- When trying to insert a row in this view, a row will be inserted in the underlying base table S with the default value for CITY.
- If the default value is not 'London', the row will not appear in the view.

Triggers

Example (Cont.)

- Create a triggered procedure:

```
CREATE TRIGGER LONDON_SUPPLIER_INSERT
    INSTEAD OF INSERT ON LONDON_SUPPLIER
    REFERENCING NEW ROW AS R
    FOR EACH ROW
    INSERT INTO S ( S#, SNAME, STATUS, CITY )
        VALUES ( R.S#, R.SNAME, R.STATUS, 'London' );
```

Triggers

Example (Cont.)

- Create a triggered procedure:
```
CREATE TRIGGER LONDON_SUPPLIER_INSERT
    INSTEAD OF INSERT ON LONDON_SUPPLIER
    REFERENCING NEW ROW AS R
    FOR EACH ROW
    INSERT INTO S ( S#, SNAME, STATUS, CITY )
        VALUES ( R.S#, R.SNAME, R.STATUS, ’London’ );
```
- Inserting a row in the view will cause a row to be inserted into the underlying base table with CITY value equal to London instead of the default value, and the new row will appear in the view.

Triggers

Example (Cont.)

- Create a triggered procedure:

```
CREATE TRIGGER LONDON_SUPPLIER_INSERT
    INSTEAD OF INSERT ON LONDON_SUPPLIER
    REFERENCING NEW ROW AS R
    FOR EACH ROW
    INSERT INTO S ( S#, SNAME, STATUS, CITY )
        VALUES ( R.S#, R.SNAME, R.STATUS, 'London' );
```

- CREATE TRIGGER specifies
  - *The event:* an operation on the database (INSERT ON LONDON_SUPPLIER).
  - *The condition:* a boolean expression that has to evaluate to true in order the action to be executed (No condition specified explicitly in the example above);
  - *The action:* the triggered procedure proper (INSERT INTO S).

Outline

Functional Dependencies

- Functional Dependence (FD): Many-to-one relationship from one set of attributes to another within a given relvar, satisfying certain conditions.
- Should distinguish clearly between
  (a) the value of a given relvar at a given point of time, and
  (b) the set of possible values that the given relvar might assume at different times.

Functional Dependencies

Functional Dependence, Case (a)

- Let r be a relation, and X and Y be arbitrary subsets of the set of attributes of r.
- Y is functionally dependent on X, written $X \rightarrow Y$ iff whenever two tuples of r agree on their X value, they also agree on their Y value.

Functional Dependencies

Example

Functional Dependence, Case (a)

SPC

S#	CITY	P#	QTY
S1	London	P1	100
S1	London	P2	100
S2	Paris	P1	200
S2	Paris	P2	200
S3	Paris	P2	300
S4	London	P2	400
S4	London	P4	400
S4	London	P5	400

FD's:

$$\{S\#\} \rightarrow \{CITY\}$$
$$\{S\#, P\#\} \rightarrow \{QTY\}$$
$$\{S\#, P\#\} \rightarrow \{CITY\}$$
$$\{S\#, P\#\} \rightarrow \{S\#\}$$
$$\{S\#, P\#\} \rightarrow \{S\#, P\#, CITY, QTY\}$$
$$\{S\#\} \rightarrow \{QTY\}$$
$$\{QTY\} \rightarrow \{S\#\}$$

Functional Dependencies

Functional Dependence, Case (b)

- Let R be a relation variable, and X and Y be arbitrary subsets of the set of attributes of R.
- Y is functionally dependent on X, written $X \rightarrow Y$ iff in every possible value legal of R, whenever two tuples agree on their X value, they also agree on their Y value.

Functional Dependencies

Example

Functional Dependence, Case (b)

SPC

S#	CITY	P#	QTY
S1	London	P1	100
S1	London	P2	100
S2	Paris	P1	200
S2	Paris	P2	200
S3	Paris	P2	300
S4	London	P2	400
S4	London	P4	400
S4	London	P5	400

Some of the FD's for SCP:

$\{S\#, P\#\} \rightarrow \{QTY\}$

$\{S\#, P\#\} \rightarrow \{CITY\}$

$\{S\#, P\#\} \rightarrow \{S\#\}$

$\{S\#, P\#\} \rightarrow \{S\#, P\#, CITY, QTY\}$

$\{S\#\} \not\rightarrow \{QTY\}$

$\{QTY\} \not\rightarrow \{S\#\}$

Functional Dependencies

- From now on FD refers to FD, case (b).
- If X is a candidate key for relvar R, then all attributes Y of R must be functionally dependent on X.
- FD for a given relvar can be large.
- Problem: Find a smaller (ideally, minimal) subset of FD's for a given relvar that implies all the FD's for that relvar.
- Why is this problem interesting?
    - FD's are certain integrity constraints.
    - DBMS should enforce them.
    - If we find a smaller subset required in the problem statement, it sufficient the DBMS to enforce just FD's from that subset. All the other FD's will be enforced automatically.

Functional Dependencies

- Trivial FD: Right side is a subset of the left side (e.g., {S#, P# } $\rightarrow$ { P#}).
- Trivial dependencies are not interesting in practice.

Functional Dependencies

Armstrong's axioms:

- Reflexivity: If $B \subseteq A$, then $A \rightarrow B$.
- Augmentation: If $A \rightarrow B$, then $A \cup C \rightarrow B \cup C$.
- Transitivity: If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$.

Armstrong's axioms are sound and complete.

Functional Dependencies

Additional rules (follow from Armstrong's axioms):

- Self-determination: If $A \to A$.
- Decomposition: If $A \to B \cup C$, then $A \to B$ and $A \to C$.
- Union: If $A \to B$ and $A \to C$, then $A \to B \cup C$.
- Composition: If $A \to B$ and $C \to D$, then $A \cup C \to B \cup D$.

Functional Dependencies

Example

- Given: relvar R with attributes A, B, C, D, E, F and FD's: $\{A\} \rightarrow \{B, C\}, \{B\} \rightarrow \{E\}, \{C, D\} \rightarrow \{E, F\}$.
- Show: $\{A, D\} \rightarrow \{F\}$ holds for R.
- 1. $\{A\} \rightarrow \{B, C\}$ (given)
  2. $\{A\} \rightarrow \{C\}$ (1, decomposition)
  3. $\{A, D\} \rightarrow \{C, D\}$ (2, augmentation)
  4. $\{C, D\} \rightarrow \{E, F\}$ (given)
  5. $\{A, D\} \rightarrow \{E, F\}$ (3 and 4, transitivity)
  6. $\{A, D\} \rightarrow \{F\}$ (5, decomposition)

Functional Dependencies. Computing the Closure

- Given: a relvar R, a set of attributes Z for R, and a set S of FD's that hold for R.
- Compute: The set of all attributes of R that are functionally dependent on Z—the closure Z^+ of Z under S.
- Algorithm:

```
$CLOSURE[Z, S] := Z$ ;
do “Forever” ;
    for each FD $X \rightarrow Y$ in $S$
        do ;
            if $X \subseteq CLOSURE[Z, S]$
            then $CLOSURE[Z, S] := CLOSURE[Z, S] \cup Y$;
        end
    if $CLOSURE[Z, S]$ did not change on this iteration
    then leave the loop ;
end
```

Functional Dependencies. Computing the Closure

Example

- Given: a relvar R with attributes A, B, C, D, E, F and FD's:

  $\{A\} \rightarrow \{B, C\}$

  $\{E\} \rightarrow \{C, F\}$

  $\{B\} \rightarrow \{E\}$

  $\{C, D\} \rightarrow \{E, F\}$
- Compute: The closure of $\{A, B\}$.
- Computation:
  1. Initialization: The closure is $\{A, B\}$
  2. After the first iteration of the algorithm: $\{A, B, C, E\}$
  3. After the second iteration: $\{A, B, C, E, F\}$
  4. After the third iteration: $\{A, B, C, E, F\}$
  5. Answer: $\{A, B, C, E, F\}$.

Functional Dependencies

Important corollary:

- Given: A set S of FD's.
- Decide: Does a specific FD $X \rightarrow Y$ follow from S?
- Decision procedure:
  1. Compute the closure X^+ of X under S.
  2. Check whether Y is a subset of X^+.
  3. If yes, $X \rightarrow Y$ follows from S, and does not, otherwise. (No need to compute S^+.

Functional Dependencies

- If S_1 and S_2 are two sets of FD's with $S_1^+ \subseteq S_2^+$, then S_2 is called a cover of S_1.
- If DBMS enforces the FD's in S_2, then it will automatically be enforcing those in S_1.
- If $S_1^+ = S_2^+$, then S_1 and S_2 are equivalent.
- A set S of FD's is irreducible iff
  1. The right side of every FD in S is a singleton set.
  2. The left side of each FD in S is irreducible—no attribute can be discarded from them without changing S^+.
  3. No FD can be discarded from S without changing S^+.
- Every set of FD's is equivalent to at least one irreducible set.

Summary

- The integrity problem is the problem to ensure that the data is consistent.
- Integrity constraints take the general form:

  IF
  a certain tuple appears in certain relvars,
  THEN
  that tuple satisfies a certain condition.
- The relvar constraint (the relvar predicate) is the conjunction of all constraints for the relvar.
- The database constraint (the database predicate) is the conjunction of all the constraints for all relvars in the database.
- The Golden Rule:
  - No update operation must ever assign to any relvar R a value that causes the constraint for R to evaluate to FALSE.

Summary

- The integrity constraints represent the meaning of the data (semantics).
- Integrity constraints into four categories:
  - Type constraints.
  - Attribute constraints.
  - Relvar constraints.
  - Database constraints.
- Keys: candidate, primary, alternate, foreign.
- Candidate keys satisfy uniqueness and irreducibility properties.

Summary

- Referential constraint: Values of a given foreign key must match the values of the corresponding candidate key.
- Referential actions and triggers.

Summary

- Functional dependency: A many-to-one relationship between two sets of attributes of given relvar, satisfying a certain functionality condition.
- Trivial FD: Right side is a subset of the left side.
- Given a set S of FD's, the closure S^+ of that set is the set of all FD's implied by the FD's in S.
- Armstrong's axioms provide a sound and complete rules to compute the closure of a given set of FD's.

Summary

- Given a subset Z of the set of attributes of relvar R and a set S of FD's that hold for R, the closure Z^+ of Z under S is the set of all attributes A of R such that the FD $Z \to A \in S^+$.
- We gave a simple algorithm for computing Z^+ from Z and S.
- Using the algorithm, one can determine whether a given FD $X \to Y$ belongs to S^+: Just check whether $Y \in X^+$.
- Two sets of FD's S_1 and S_2 are equivalent iff they are covers of each other: $S_1^+ = S_2^+$.
- Every set of FD's is equivalent to at least one irreducible set.
- If I is an irreducible set equivalent to S, enforcing the FD's in I will automatically enforce the FD's.