Implementing security rules using assignment constraints

Quis custodet ipse custodes ?

Security – what ?

- Identifying all business risks

Security – what ?

- Identifying all business risks
- Identifying measures for eliminating/reducing those risks

Security – what ?

- Identifying all business risks
- Identifying measures for eliminating/reducing those risks
- Implement those measures

Which risks ?

- Not data-related
  - ...
- Data-related
  - Loss of data
  - Unauthorized use/access

Preventing unauthorized use/access

- Three Pillars :
  - Subjects
    - Identification
    - Authentication

Preventing unauthorized use/access

- Three Pillars :
  - Subjects
  - Objects
    - Identification
    - Authentication

Preventing unauthorized use/access

- Three Pillars :
  - Subjects
  - Objects
  - Usage Modes
    - Defining Usage Rules
    - Enforcing Usage Rules

Unauthorized access in a database context

- Subjects : Database Users
  - End users vs. programs vs. both ?

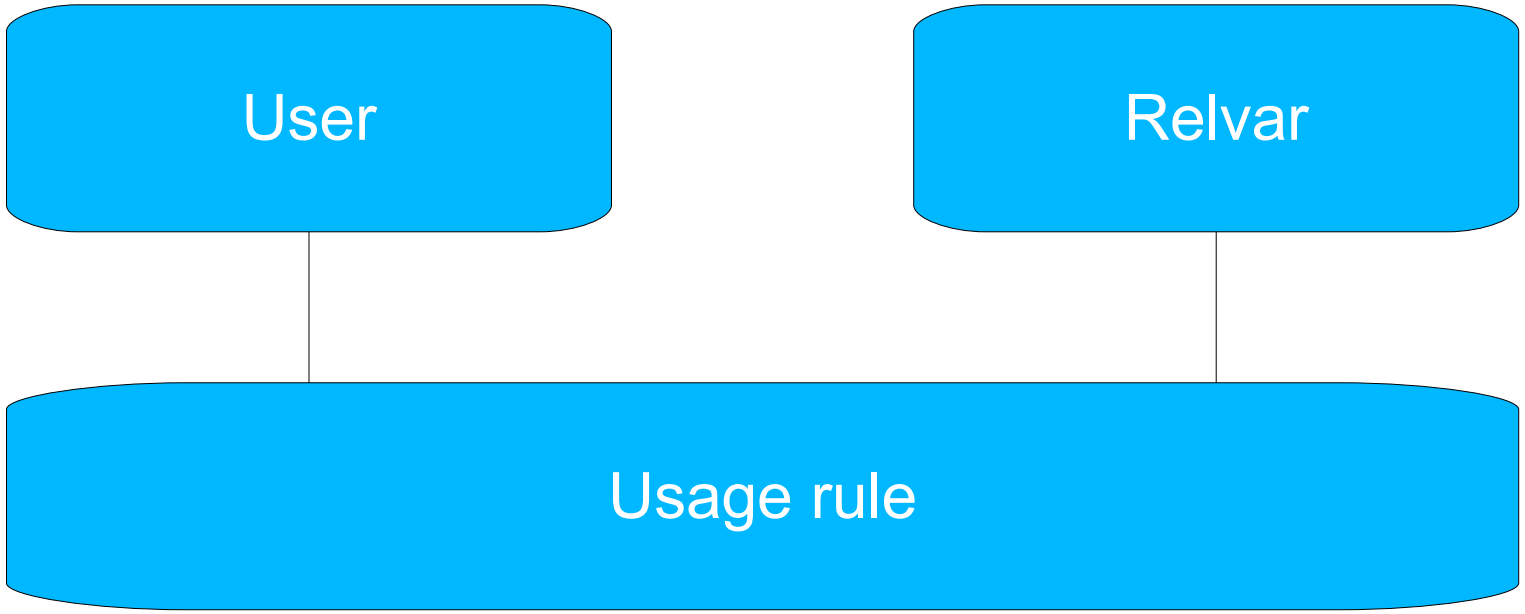
Unauthorized access in a database context

- Subjects : Database Users
- Objects : Relvars

Unauthorized access in a database context

- Subjects : Database Users
- Objects : Relvars
- Usage Modes :
  - Read
  - Write
    - Insert
    - Delete

An information model ©



Database Write Rules

- Rule that defines
  - For any given database update
  - Whether or not it is allowed
  - Possibly depending on user identity

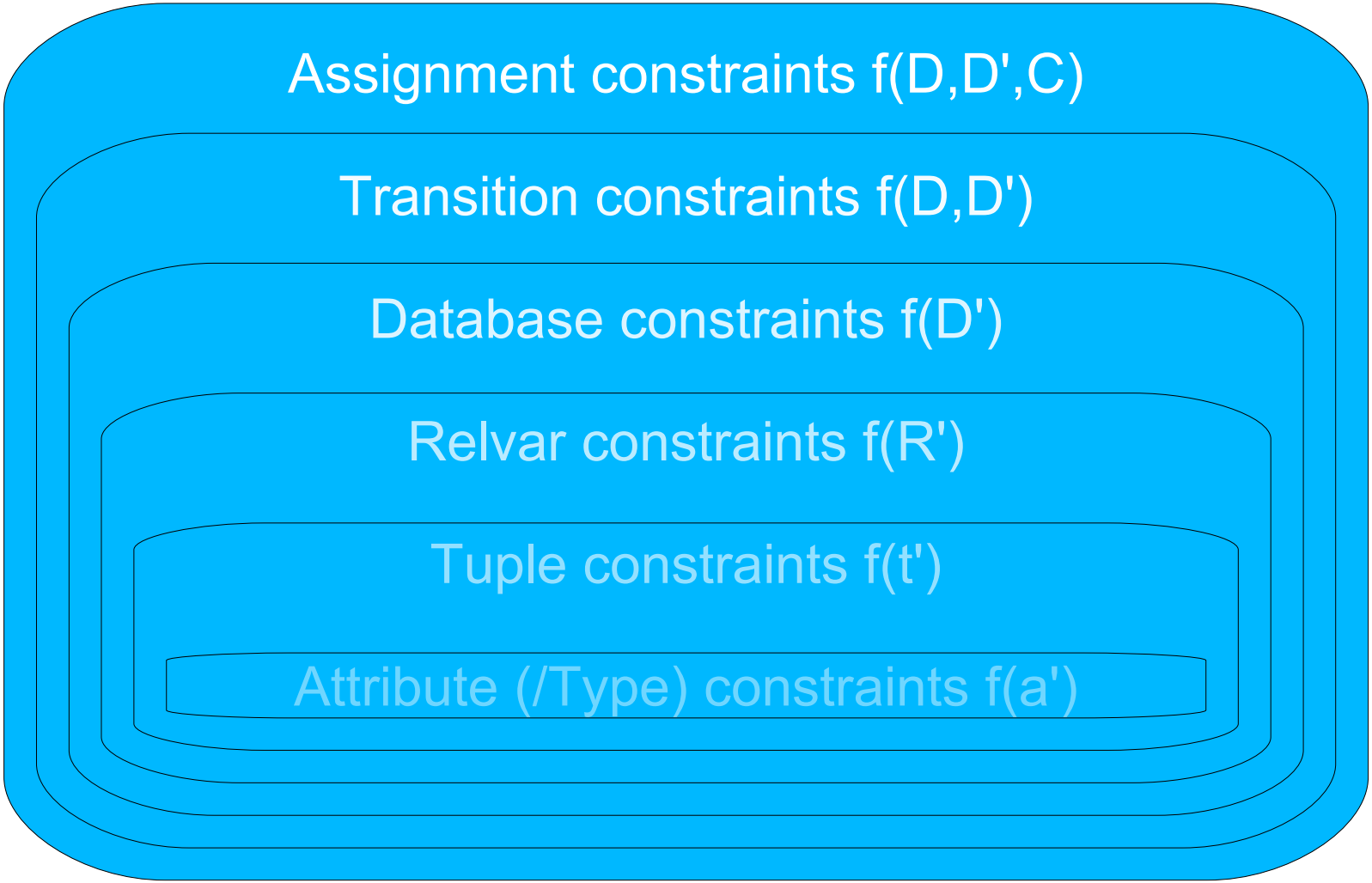
Database Write Rules

- Rule that defines
  - For any given database update
  - Whether or not it is allowed
  - Possibly depending on user identity
- First two lines ring a bell …
  - Database update = database transition
  - Whether or not it is allowed = constraint

Database Write Rules

- Rule that defines
  - For any given database update
  - Whether or not it is allowed
  - Possibly depending on user identity
- Can transition constr. refer to user identity ?
  - Agreement (in the literature) ?
  - Contextual information in general ?
  - “Assignment” constraint

Constraints overview



Database Write Rules

- Rule that defines
  - For any given database update
  - Whether or not it is allowed
  - Possibly depending on user identity
- “Assignment” constraints are all you need to enforce update authorization rules

Database updating

- The “assignment” model
  - D' = new DB value

Database updating

- The “assignment” model
  - D' = new DB value
- The “deltas” model
  - D' = do_inserts_and_deletes (D, S)

Database updating

- The “assignment” model
  - D' = new value
- The “deltas” model
  - D' = do_inserts_and_deletes (D, S)
- Both are equivalent :
  - D = undo_i's_and_d's (D', S)
  - S = dbdelta(D', D)

Database Write Rules

- Assignment constraint = f(D, D', C)
- = f(D, do_inserts_and_deletes (D, S), C)
- = g(D, S, C)

Database Write Rules

- Assignment constraint = g(D, S, C)
- Canonical form
  - IS_EMPTY(<rel exp>)
  - <rel exp> in terms of D, S, C
  - <rel exp> defines all the possible violations of the rules
  - The IS_EMPTY then says that “there cannot be any such violations”

The STATEMENT tuple

- Is a tuple

The STATEMENT tuple

- Is a tuple
- Of degree equal to that of the DBVAR, with the same attribute names as the DBVAR,

The STATEMENT tuple

- Of degree equal to that of the DBVAR, with the same attribute names as the DBVAR,
- Where the values of those attributes are TUPLES of degree two, with the attribute names being (e.g.) 'I' and 'D',

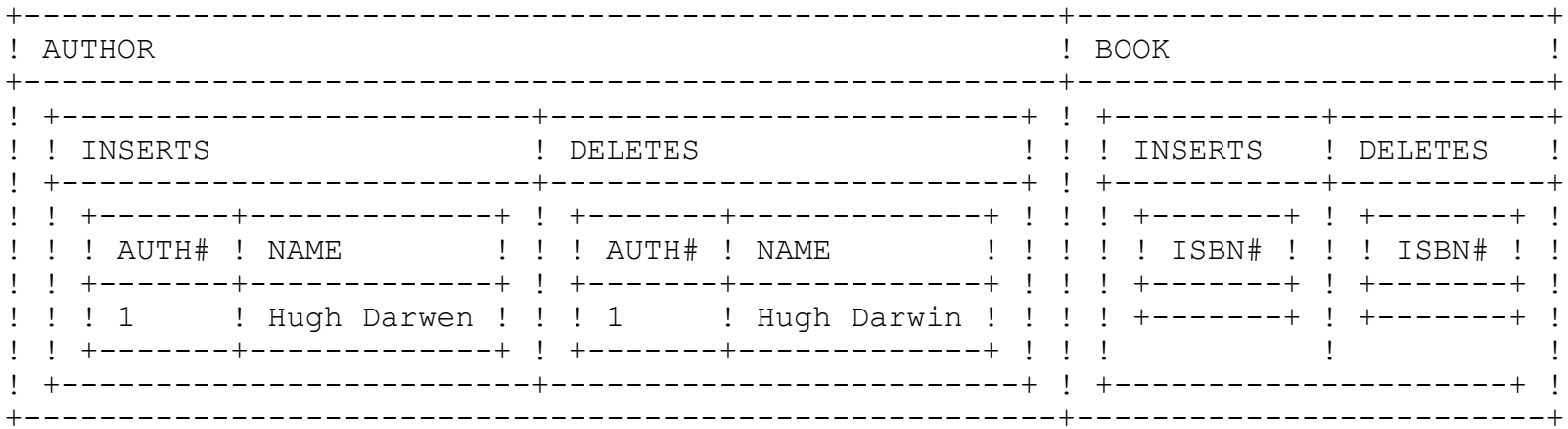
The STATEMENT tuple

- Where the values of those attributes are TUPLES of degree two, with the attribute names being (e.g.) 'I' and 'D',
- These I and D attributes are relation-valued, holding attempted inserts and attempted deletes respectively,

The STATEMENT tuple

- Where the values of those attributes are TUPLES of degree two, with the attribute names being (e.g.) 'I' and 'D',
- These I and D attributes are relation-valued, holding attempted inserts and attempted deletes respectively,
- And each I/D pair is disjoint.

The STATEMENT tuple



- D expressions :
  - Inserts from author from statement
  - ...

Attempted vs. Actual I/D

- Actual = subset of attempted
  - (Deletes from author from statement) intersect author
  - (Inserts from author from statement) minus author
- But data access authorization is best based on attempted updates !

Update steps

- Compute attempted deletes/inserts

Update steps

- Compute attempted deletes/inserts
- [Compute actual deletes/inserts]

Update steps

- Compute attempted deletes/inserts
- [Compute actual deletes/inserts]
- For each nonempty I() and/or D(), determine the set of constraints to be checked.

Update steps

- Compute attempted deletes/inserts
- [Compute actual deletes/inserts]
- For each nonempty I() and/or D(), determine the set of constraints to be checked.
- For all constraints in that set, compute the value of its expression g(D,S,C).

Update steps

- Compute attempted deletes/inserts
- [Compute actual deletes/inserts]
- For each nonempty I() and/or D(), determine the set of constraints to be checked.
- For all constraints in that set, compute the value of its expression g(D,S,C).
- Iff all constraints satisfied, apply the update

Update steps

- Compute attempted deletes/inserts
  - Following the rules of RM PRE 21

Update steps

- Compute attempted deletes/inserts
  - Following the rules of RM PRE 21
  - Applying the 'transformations' of appendix E if assignment targets are virtuals.

Update steps

- Compute attempted deletes/inserts
  - Following the rules of RM PRE 21
  - Applying the 'transformations' of appendix E if assignment targets are virtuals.
  - Including the processing of triggers, compensatory actions, and suchlike

Update steps

- Compute attempted deletes/inserts
  - Following the rules of RM PRE 21
    - R := v
      - D(R) = R minus v, I(R) = v minus R ?
      - D(R) = R, I(R) = v ?
      - D(R) = U, I(R) = v ?
      - Option 1 guarantees I(R) disjoint with D(R)

Update steps

- Compute attempted deletes/inserts
  - Following the rules of RM PRE 21
    - R := v
    - Update “orthogonal” - treated as D+I
      - D(R) MATCHING I(R){identifying key}
      - I(R) MATCHING D(R){identifying key}

Update steps

- Compute attempted deletes/inserts
  - Following the rules of RM PRE 21
    - R := v
    - Update “orthogonal” - treated as D+I
    - Insert v into R still eqv to assignment ?
      - Assume v = R
      - I(R) = R, AI(R) = {} is intended
      - R := R union R,
      - I(R) = (R union R) minus R = {}
      - Problem ???

Update steps

- Compute attempted deletes/inserts
  - Applying the 'transformations' of appendix E if assignment targets are virtuals.
    - Approach gets independent of base/virtual

Update steps

- Compute attempted deletes/inserts
  - Applying the 'transformations' of appendix E if assignment targets are virtuals.
    - Approach gets independent of base/virtual
    - But no direct support for using I(V) in constraint expressions …

Update steps

- Compute attempted deletes/inserts
  - Applying the 'transformations' of appendix E if assignment targets are virtuals.
    - Approach gets independent of base/virtual
    - But no direct support for using I(V) in constraint expressions …
    - AI(V)/AD(V) are computable
    - [Also good means for implementing checks of the Assignment Principle ?]

Update steps

- Compute attempted deletes/inserts
  - Including the processing of triggers, compensatory actions, and suchlike
    - <=> CJD (“cascades mustn't be done until after the updates themselves are done”)

Update steps

- Compute attempted deletes/inserts
  - Including the processing of triggers, compensatory actions, and suchlike
    - <=> CJD (“cascades mustn't be done until after the updates themselves are done”)
    - There will be nonempty AI()/AD() for relvars not mentioned in the update statement.
    - Potential conflict with update authorizations on these relvars !

Update steps

- Compute attempted deletes/inserts
- [Compute actual deletes/inserts]
- For each nonempty I() and/or D(), determine the set of constraints to be checked.
- For all constraints in that set, compute the value of its expression g(D,S,C).
- Iff all constraints satisfied, apply the update

Update steps

- Compute actual deletes/inserts
  - As per I(R) minus R and D(R) intersect R
  - [Important as the basis for checking the database constraints]
  - [Important to 'cache' them as the input for the final apply-the-updates step]
  - [performance : access to R inevitable]
    - D_UNION ===> exception or not ?
    - UNION ===> actual insert or not ?

Update steps

- ✔ Compute attempted deletes/inserts
- ✔ [Compute actual deletes/inserts]
- For each nonempty I() and/or D(), determine the set of constraints to be checked.
- For all constraints in that set, compute the value of its expression g(D,S,C).
- Iff all constraints satisfied, apply the update

Update steps

- Determine constraints to be checked.
  - User is required to declare this info ?

Update steps

- Determine constraints to be checked.
  - User is required to declare this info ?
  - Hypothetical D syntax :

    ASSIGNMENT CONSTRAINT <name> <bool exp>
    CHECK ON <update type list>;

    <update type> ::= INSERT TO <relvarname> | DELETE FROM <relvarname>

Update steps

- Determine constraints to be checked.
  - User is required to declare this info ?
  - Hypothetical D syntax :

    ASSIGNMENT CONSTRAINT <name> <bool exp>
    CHECK ON <update type list>;

    <update type> ::= INSERT TO <relvarname> | DELETE FROM <relvarname>
  - Explicitly recorded in the catalogue

Update steps

- Determine constraints to be checked.
  - Info derivable from g(D, S, C) ?

Update steps

- Determine the constraints to be checked.
  - Info derivable from g(D, S, C) ?
  - That's why we wanted the canonical form IS_EMPTY(...) !

Update steps

- Determine the constraints to be checked.
  - Info derivable from g(D, S, C) ?
  - That's why we wanted the canonical form IS_EMPTY(...) !
  - Full set of possible 'causes' of non-emptiness of any <rel exp> is computable
    - I(R1) INTERSECT I(R2)
    - I(R1) MINUS I(R2)
    - (R MINUS D(R)) UNION I(R)

Update steps

- Determine the constraints to be checked.
  - Full set of possible 'causes' of non-emptiness of any <rel exp> is computable
    - (R MINUS D(R)) UNION I(R)
  - R nonempty at the time of introduction of this assignment constraint
    - Constraint always violated !
    - Which updates to reject ?
    - Conclusion : ???

Update steps

- Determine the constraints to be checked.
  - Full set of possible 'causes' of non-emptiness of any <rel exp> is computable
  - “Department managers not allowed to update the database”
    - ISEMPTY(C JOIN (RENAME DEPT ...))
    - Which updates to reject ?
    - Conclusion : ???

Update steps

- Determine the constraints to be checked.
  - 'Actual' non-emptiness only known at run-time, 'potential' non-emptiness to be used at compile time.

Update steps

- Determine the constraints to be checked.
  - 'Actual' non-emptiness only known at run-time, 'potential' non-emptiness to be used at compile time.
  - 'Conflict' with view updating ?
    - V = R1 UNION R2
    - User U not authorized to update R1
    - User U issues INSERT t into V

Update steps

- Compute attempted deletes/inserts
- [Compute actual deletes/inserts]
- For each nonempty I() and/or D(), determine the set of constraints to be checked.
- For all constraints in that set, compute the value of its expression g(D,S,C).
- Iff all constraints satisfied, apply the update

Update steps

- For all constraints in that set, compute the value of its expression g(D,S,C).
  - Efficiency of the security mechanism is a function of how good the optimiser is

Update steps

- For all constraints in that set, compute the value of its expression g(D,S,C).
  - Efficiency of the security mechanism is a function of how good the optimiser is
  - Limitations to g ? Essentially none.

Update steps

- Compute attempted deletes/inserts
- [Compute actual deletes/inserts]
- For each nonempty I() and/or D(), determine the set of constraints to be checked.
- For all constraints in that set, compute the value of its expression g(D,S,C).
- Iff all constraints satisfied, apply the update

Database Read Rules

- Rule that defines
  - For any given database read
  - Whether or not it is allowed
  - Possibly depending on user identity

Database Read Rules

- Rule that defines
  - For any given database read
  - Whether or not it is allowed
  - Possibly depending on user identity
- Rings a bell of anything ?
  - Unfortunately no ...

Database Read Rules

- Typical nature of read rules :
  - “U not allowed to see relvar R's contents”
    - Not really a problem

Database Read Rules

- Typical nature of read rules :
  - “U not allowed to see relvar R's contents”
  - “U allowed to see R's contents partially”
    - “row-wise” : based on some RESTRICT
    - “column-wise” : based on some PROJECT
    - Typical answer for either case : define a view
    - But then what about updating the view ?

Database Read Rules

- Partial, attribute-oriented :
  - Make “non-partial” by vertical decomposition
    - In the same go, allows for more fine-grained definition of update rules
    - Without view updating entering the picture

Database Read Rules

- Partial, attribute-oriented, alternative view
  - Is a constraint to the effect that

Database Read Rules

- Partial, attribute-oriented, alternative view
  - Is a constraint to the effect that
  - Some particular relvar attribute

Database Read Rules

- Partial, attribute-oriented, alternative view
  - Is a constraint to the effect that
  - Some particular relvar attribute
  - Can or cannot be included in the result sets

Database Read Rules

- Partial, attribute-oriented, alternative view
  - Is a constraint to the effect that
  - Some particular relvar attribute
  - Can or cannot be included in the result sets
  - Of queries issued by some user

Database Read Rules

- Partial, attribute-oriented, alternative view
  - But what does “include” mean ?
    - “SELECT” SALARY / 2 FROM ...
    - “SELECT” INT(LOG10(SALARY)) FROM ...
    - …
  - Conclusion : alternative view hard to pin down precisely for all cases
  - 2nd conclusion : vertical decomposition probably the right approach

Database Read Rules

- Partial, tuple-oriented :
  - Boils down to some predicate p :
    - Determined by user identity
    - Defines which tuples are “visible”

Database Read Rules

- Partial, tuple-oriented :
  - Boils down to some predicate p :
    - Determined by user identity
    - Defines which tuples are “visible”
  - Aside : this gives rise to predicates like “User <U> is allowed to see tuples from <R> if those tuples satisfy <P>”.
    - <P> is a variable, and <P> is a predicate
    - ===> 2nd order logic problem ?

Database Read Rules

- Partial, tuple-oriented :
  - Boils down to some predicate P(U,R)
  - Turned into “non-partial” by horizontal decomposition using that very predicate
    - But not needed for more fine-grained definition of update rules
    - Combinatorial explosion if multiple Users :
      - R WHERE P(U1,R) AND P(U2,R)
      - R WHERE P(U1,R) AND NOT P(U2,R)
      - R WHERE NOT P(U1,R) AND P(U2,R)
      - ...

Database Read Rules

- Partial, tuple-oriented :
  - Boils down to some predicate P(U,R)
  - Gives rise to de-facto virtual relvars :
    - V1 = R WHERE P(U1,R)
    - V2 = R WHERE P(U2,R)

Database Read Rules

- Partial, tuple-oriented :
  - Boils down to some predicate P(U,R)
  - Gives rise to de-facto virtual relvars :
    - V1 = R WHERE P(U1,R)
    - V2 = R WHERE P(U2,R)
  - Updates to R by Un automatically interpreted as updates to Vn ?
    - Can be done using Assignment principle ?
    - Eliminates need to repeat the rule as AC.

Database Read Rules

- Partial, tuple-oriented :
  - Boils down to some predicate P(U,R)
  - When to check P(U,R) ?
    - Whenever a reference to R is evaluated
    - “Automatically” propagates P(U,R) to any virtual relvar defined in terms of R
    - [Except perhaps in the case of “materialised” views !]

Database Read Rules

- Partial, tuple-oriented :
  - Boils down to some predicate P(U,R)
  - When to check P(U,R) ?
  - Can R itself be virtual ?
    - Inconsistencies become possible with any P(U,B) on the base relvars in terms of which R is defined
    - Avoiding those inconsistencies probably involves lots of repetition
    - ===> At the very least strongly discouraged

Database Read Rules

- Partial, tuple-oriented, alternative view
  - “Convert” the 'reading' problem into a 'writing' problem.
  - Each read transformed into a write-to-log.
  - Enforce read security by means of enforcing write-to-log security.

Database Read Rules

- Partial, tuple-oriented, alternative view
  - Enforce read security by means of enforcing write-to-log security.
    - But : STATEMENT tuple for this case not complete until end-of-processing ===> checks impossible at start-of-processing
    - Update security violation = exception. For read security violations too ?

Conclusions

- Enforcing data access rules possible

Conclusions

- Enforcing data access rules possible :
  - Irrespective of complexity of the rule

Conclusions

- Enforcing data access rules possible :
  - Irrespective of complexity of the rule
  - Without requiring “security-specific” information models (think catalogs that record GRANT/REVOKE data)

Conclusions

- Enforcing data access rules possible :
  - Irrespective of complexity of the rule
  - Without requiring “security-specific” information models (think catalogs that record GRANT/REVOKE data)
  - Both for writes and for reads

Conclusions

- Enforcing data access rules possible :
  - Irrespective of complexity of the rule
  - Without requiring “security-specific” information models (think catalogs that record GRANT/REVOKE data)
  - Both for writes and for reads
  - But write/read security is only similar at a very high level of abstraction (“they are both predicates”)