Count Constraints for Inverse OLAP and Aggregate Data Exchange

D. Sacca[1] E. Serra[1] A. Guzzo[1]

DEIS[1], University of Calabria, 87036, Rende, Italy
{sacca,eserra,guzzo}@deis.unical.it

June 25, 2012

Reference Scenario: what is missing?

- Classical problem in DB theory: *does there exist a relation (or db) instance satisfying given dependency constraints?*

Reference Scenario: what is missing?

- Classical problem in DB theory: *does there exist a relation (or db) instance satisfying given dependency constraints?*
- Recent renewed deal of interest within the context of data exchange

Reference Scenario: what is missing?

- Classical problem in DB theory: *does there exist a relation (or db) instance satisfying given dependency constraints?*
- Recent renewed deal of interest within the context of data exchange
- **But no much investigations and results on checking constraints over aggregate data**

Our contribution

- We introduce **count constraints**: *the results of given count operations on a relation must be within a certain range!*

Our contribution

- We introduce **count constraints**: *the results of given count operations on a relation must be within a certain range!*
- Count constraints are defined by a suitable extension of first order predicate calculus, based on set terms

Our contribution

- We introduce **count constraints**: *the results of given count operations on a relation must be within a certain range!*
- Count constraints are defined by a suitable extension of first order predicate calculus, based on set terms
- New decisional problem, **Inverse OLAP**: *given a star schema, does there exist a relation instance satisfying a set of given count constraints?*

Our contribution

- We introduce **count constraints**: *the results of given count operations on a relation must be within a certain range!*
- Count constraints are defined by a suitable extension of first order predicate calculus, based on set terms
- New decisional problem, **Inverse OLAP**: *given a star schema, does there exist a relation instance satisfying a set of given count constraints?*
- Inverse OLAP can be used to generate "realistic" instances (eg, for benchmark, privacy)

Our contribution

- We introduce **count constraints**: *the results of given count operations on a relation must be within a certain range!*
- Count constraints are defined by a suitable extension of first order predicate calculus, based on set terms
- New decisional problem, **Inverse OLAP**: *given a star schema, does there exist a relation instance satisfying a set of given count constraints?*
- Inverse OLAP can be used to generate "realistic" instances (eg, for benchmark, privacy)
- Count constraints can be also used for aggregate data exchange and data exchange with count constraints.

An Example of OLAP Star Schema

We refer to a classical example of point-of-sales transaction star schema. The attributes of U are:

- T (Transaction)
- I (Item)
- B (Brand)
- S (Store)
- A (Area)

— their (finite) domains are: $\mathcal{D}_T$, $\mathcal{D}_I$ and so on.
We are also given the following functional dependencies (FDs):

- $T \rightarrow S$
- $S \rightarrow A$

$\{T, I, B\}$ is the *relation key*.

A Fact Table

T	I	B	S	A
t_1	a_1	b_1	s_1	r_1
t_1	a_2	b_1	s_1	r_1
t_1	a_3	b_2	s_1	r_1
t_2	a_1	b_3	s_1	r_1
t_3	a_1	b_1	s_2	r_1
t_3	a_3	b_4	s_2	r_1
t_4	a_1	b_1	s_3	r_2
t_4	a_3	b_4	s_3	r_2

Counts

- Items of t_1: $\{a_1, a_2, a_3\}$
- Items of t_2: $\{a_1\}$
- Items of t_3: $\{a_1, a_3\}$
- Items of t_4: $\{a_1, a_3\}$

T	I	B	S	A
t_1	a_1	b_1	s_1	r_1
t_1	a_2	b_1	s_1	r_1
t_1	a_3	b_2	s_1	r_1
t_2	a_1	b_3	s_1	r_1
t_3	a_1	b_1	s_2	r_1
t_3	a_3	b_4	s_2	r_1
t_4	a_1	b_1	s_3	r_2
t_4	a_3	b_4	s_3	r_2

Count Constraints

Definition of Count Constraint

Given a relation scheme $\mathcal{R}(A_1, \ldots, A_n)$ together with its domain relations $\mathcal{D}_1, \ldots, \mathcal{D}_n$, a count constraint C can be of two type:

Definition of Count Constraint

Given a relation scheme $\mathcal{R}(A_1, \ldots, A_n)$ together with its domain relations $\mathcal{D}_1, \ldots, \mathcal{D}_n$, a count constraint C can be of two type:

1. tuple count constraint
   $\forall \mathbf{X} \; (\; \phi(\mathbf{X}) \; \rightarrow \; \beta_{min} \leq \#(\{ \; \mathbf{Y} : \exists \mathbf{Z} \; \psi(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) \; \}) \leq \beta_{max} \;)$

Definition of Count Constraint

Given a relation scheme $\mathcal{R}(A_1, \ldots, A_n)$ together with its domain relations $\mathcal{D}_1, \ldots, \mathcal{D}_n$, a count constraint C can be of two type:

1. tuple count constraint
   $\forall \mathbf{X} \; (\; \phi(\mathbf{X}) \; \rightarrow \; \beta_{min} \leq \#(\{ \; \mathbf{Y} : \exists \mathbf{Z} \; \psi(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) \; \}) \leq \beta_{max} \;)$
2. group count constraint
   $\forall \mathbf{X} \; (\; \phi(\mathbf{X}) \; \rightarrow \; \beta_{min} \leq \#(\; \{ \; \mathbf{W} : t * \{ \; \mathbf{Y} : \exists \mathbf{Z} \; \psi(\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{W}) \; \} \; \}) \leq \beta_{max} \;)$

Definition of Count Constraint

Given a relation scheme $\mathcal{R}(A_1, \ldots, A_n)$ together with its domain relations $\mathcal{D}_1, \ldots, \mathcal{D}_n$, a count constraint C can be of two type:

1. tuple count constraint
   $\forall \mathbf{X} \; (\; \phi(\mathbf{X}) \; \rightarrow \; \beta_{min} \leq \#(\{ \; \mathbf{Y} : \exists \mathbf{Z} \; \psi(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) \; \}) \leq \beta_{max} \;)$
2. group count constraint
   $\forall \mathbf{X} \; (\; \phi(\mathbf{X}) \; \rightarrow \; \beta_{min} \leq \#(\; \{ \; \mathbf{W} : t * \{ \; \mathbf{Y} : \exists \mathbf{Z} \; \psi(\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{W}) \; \} \; \}) \leq \beta_{max} \;)$

where

- where **X**, **Y**, **Z** and **W** are disjunct lists of variables (**X** and **Z** can be empty);
- ϕ is a (possibly empty) conjunction of domain and comparison predicates;
- ψ is a conjunction of relation and comparison predicates;

Definition of Count Constraint

Given a relation scheme $\mathcal{R}(A_1, \ldots, A_n)$ together with its domain relations $\mathcal{D}_1, \ldots, \mathcal{D}_n$, a count constraint C can be of two type:

1. tuple count constraint
   $\forall \mathbf{X} \; (\; \phi(\mathbf{X}) \; \rightarrow \; \beta_{min} \leq \#(\{ \; \mathbf{Y} : \exists \mathbf{Z} \; \psi(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) \; \}) \leq \beta_{max} \;)$
2. group count constraint
   $\forall \mathbf{X} \; (\; \phi(\mathbf{X}) \; \rightarrow \; \beta_{min} \leq \#(\; \{ \; \mathbf{W} : t * \{ \; \mathbf{Y} : \exists \mathbf{Z} \; \psi(\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{W}) \; \} \; \}) \leq \beta_{max} \;)$

where

- where **X**, **Y**, **Z** and **W** are disjunct lists of variables (**X** and **Z** can be empty);
- ϕ is a (possibly empty) conjunction of domain and comparison predicates;
- ψ is a conjunction of relation and comparison predicates;
- β_{min} and β_{max} are integers;
- operator $*$ can be either $\subseteq$, $\subset$ or $=$;

Examples of Count Constraint

- The FD $T \rightarrow S$ can be expressed as:

$$\mathcal{D}_T(T) \rightarrow 0 \leq \#(\{S : \exists I, B, A \ \ \mathcal{R}(T, I, B, S, A)\}) \leq 1$$

Examples of Count Constraint

- The FD $T \rightarrow S$ can be expressed as:
$$\mathcal{D}_T(T) \rightarrow 0 \leq \#(\{S : \exists I, B, A \ \ \mathcal{R}(T, I, B, S, A)\}) \leq 1$$

- The relation key $\{T, I, B\}$ can be enforced as:
$$\mathcal{D}_T(T) \wedge \mathcal{D}_I(I) \wedge \mathcal{D}_B(B) \rightarrow$$
$$0 \leq \#(\{S, A : \ \ \mathcal{R}(T, I, B, S, A))\}) \leq 1$$

Examples of Count Constraint

- The FD $T \rightarrow S$ can be expressed as:
$$\mathcal{D}_T(T) \rightarrow 0 \leq \#(\{S : \exists I, B, A \ \ \mathcal{R}(T, I, B, S, A)\}) \leq 1$$

- The relation key $\{T, I, B\}$ can be enforced as:
$$\mathcal{D}_T(T) \wedge \mathcal{D}_I(I) \wedge \mathcal{D}_B(B) \rightarrow$$
$$0 \leq \#(\{S, A : \ \mathcal{R}(T, I, B, S, A))\}) \leq 1$$

- There must be between 5000 and 10000 tuples in any instance of R:
$$\rightarrow 5000 \leq \#(\{T, I, B, S, A : \ \mathcal{R}(T, I, B, S, A)\}) \leq 10000$$

Examples of Count Constraint: *continued*

- There must be between 1000 and 2000 transactions in every region, except in "Cal" for which the upper bound is increased to 9000:

$$\rightarrow \ 1000 \leq \#(\{T : \exists I, B, S \ \ \mathcal{R}(T, I, B, S, "Cal")\}) \leq 9000$$

$$\mathcal{D}_A(A) \wedge A \neq "Cal" \ \rightarrow$$
$$1000 \leq \#(\{T : \exists I, B, S \ \ \mathcal{R}(T, I, B, S, A)\}) \leq 2000$$

Examples of Count Constraint: *continued*

- There must be between 1000 and 2000 transactions in every region, except in "Cal" for which the upper bound is increased to 9000:

$$\rightarrow \ 1000 \leq \#(\{T : \exists I, B, S \ \ \mathcal{R}(T, I, B, S, "Cal")\}) \leq 9000$$

$$\mathcal{D}_A(A) \wedge A \neq "Cal" \ \rightarrow$$
$$1000 \leq \#(\{T : \exists I, B, S \ \ \mathcal{R}(T, I, B, S, A)\}) \leq 2000$$

- Enforce the above constraints in every store of an area:

$$\mathcal{D}_{S,A}(S, "Cal") \rightarrow$$
$$1000 \leq \#(\{T : \exists I, B \ \ \mathcal{R}(T, I, B, S, "Cal")\}) \leq 9000$$

$$\mathcal{D}_{S,A}(S, A) \wedge A \neq "Cal" \ \rightarrow$$
$$1000 \leq \#(\{T : \exists I, B \ \ \mathcal{R}(T, I, B, S, A)\}) \leq 2000$$

Group count constraints

Both the sets of items ($i = \{a_1, a_2\}$ and $j = \{a_1, a_3\}$) must be present in at least 1 and in at most 3 transactions:

$$s = i \vee s = j \rightarrow$$
$$1 \leq \#(\{T : s \subseteq \{I : \exists B, S, A \ \mathcal{R}(T, I, B, S, A)\}\}) \leq 3;$$

Group count constraints

Both the sets of items ($i = \{a_1, a_2\}$ and $j = \{a_1, a_3\}$) must be present in at least 1 and in at most 3 transactions:

$$s = i \vee s = j \rightarrow$$
$$1 \leq \#(\{T : s \subseteq \{I : \exists B, S, A\ \mathcal{R}(T, I, B, S, A)\}\}) \leq 3;$$

T	I	B	S	A
t_1	a_1	b_1	s_1	r_1
t_1	a_2	b_1	s_1	r_1
t_1	a_3	b_2	s_1	r_1
t_2	a_1	b_3	s_1	r_1
t_3	a_1	b_1	s_2	r_1
t_3	a_3	b_4	s_2	r_1
t_4	a_1	b_1	s_3	r_2
t_4	a_3	b_4	s_3	r_2

Group count constraints

Both the sets of items ($i = \{a_1, a_2\}$ and $j = \{a_1, a_3\}$) must be present in at least 1 and in at most 3 transactions:

$$s = i \vee s = j \rightarrow$$
$$1 \leq \#(\{T : s \subseteq \{I : \exists B, S, A\ \mathcal{R}(T, I, B, S, A)\}\}) \leq 3;$$

T	I	B	S	A
t_1	a_1	b_1	s_1	r_1
t_1	a_2	b_1	s_1	r_1
t_1	a_3	b_2	s_1	r_1
t_2	a_1	b_3	s_1	r_1
t_3	a_1	b_1	s_2	r_1
t_3	a_3	b_4	s_2	r_1
t_4	a_1	b_1	s_3	r_2
t_4	a_3	b_4	s_3	r_2

Grouping Operation:

- $t_1 = \{a_1, a_2, a_3\}$
- $t_2 = \{a_1\}$
- $t_3 = \{a_1, a_3\}$
- $t_4 = \{a_1, a_3\}$

Group count constraints

Both the sets of items ($i = \{a_1, a_2\}$ and $j = \{a_1, a_3\}$) must be present in at least 1 and in at most 3 transactions:

$$s = i \vee s = j \rightarrow$$
$$1 \leq \#(\{T : s \subseteq \{I : \exists B, S, A\ \mathcal{R}(T, I, B, S, A)\}\}) \leq 3;$$

T	I	B	S	A
t_1	a_1	b_1	s_1	r_1
t_1	a_2	b_1	s_1	r_1
t_1	a_3	b_2	s_1	r_1
t_2	a_1	b_3	s_1	r_1
t_3	a_1	b_1	s_2	r_1
t_3	a_3	b_4	s_2	r_1
t_4	a_1	b_1	s_3	r_2
t_4	a_3	b_4	s_3	r_2

Grouping Operation:

- $t_1 = \{a_1, a_2, a_3\}$
- $t_2 = \{a_1\}$
- $t_3 = \{a_1, a_3\}$
- $t_4 = \{a_1, a_3\}$

Count Operation ($\subseteq$):

- $s = \{a_1, a_2\}$
  $\#(\{t_1\}) = 1$
- $s = \{a_1, a_3\}$
  $\#(\{t_1, t_3, t_4\}) = 3$

Group count constraints

Both the sets of items ($i = \{a_1, a_2\}$ and $j = \{a_1, a_3\}$) must be present in at least 1 and in at most 3 transactions:

$$s = i \vee s = j \rightarrow$$
$$1 \leq \#(\{T : s \subset \{I : \exists B, S, A\ \mathcal{R}(T, I, B, S, A)\}\}) \leq 3;$$

T	I	B	S	A
t_1	a_1	b_1	s_1	r_1
t_1	a_2	b_1	s_1	r_1
t_1	a_3	b_2	s_1	r_1
t_2	a_1	b_3	s_1	r_1
t_3	a_1	b_1	s_2	r_1
t_3	a_3	b_4	s_2	r_1
t_4	a_1	b_1	s_3	r_2
t_4	a_3	b_4	s_3	r_2

Grouping Operation:

- $t_1 = \{a_1, a_2, a_3\}$
- $t_2 = \{a_1\}$
- $t_3 = \{a_1, a_3\}$
- $t_4 = \{a_1, a_3\}$

Count Operation ($\subset$):

- $s = \{a_1, a_2\}$
  $\#(\{t_1\}) = 1$
- $s = \{a_1, a_3\}$
  $\#(\{t_1\}) = 1$

Group count constraints

Both the sets of items ($i = \{a_1, a_2\}$ and $j = \{a_1, a_3\}$) must be present in at least 1 and in at most 3 transactions:

$$s = i \vee s = j \rightarrow$$
$$1 \leq \#(\{T : s = \{I : \exists B, S, A\ \mathcal{R}(T, I, B, S, A)\}\}) \leq 3$$

T	I	B	S	A
t_1	a_1	b_1	s_1	r_1
t_1	a_2	b_1	s_1	r_1
t_1	a_3	b_2	s_1	r_1
t_2	a_1	b_3	s_1	r_1
t_3	a_1	b_1	s_2	r_1
t_3	a_3	b_4	s_2	r_1
t_4	a_1	b_1	s_3	r_2
t_4	a_3	b_4	s_3	r_2

Grouping Operation:

- $t_1 = \{a_1, a_2, a_3\}$
- $t_2 = \{a_1\}$
- $t_3 = \{a_1, a_3\}$
- $t_4 = \{a_1, a_3\}$

Count Operation ($=$):

- $s = \{a_1, a_2\}$
  $\#(\{\}) = 0$
- $s = \{a_1, a_3\}$
  $\#(\{t_3, t_4\}) = 2$

Count Constraints for Inverse OLAP

Inverse OLAP: Definition and Complexity

Problem

Given a relation scheme $\mathcal{R}(A_1, \ldots, A_n)$ with $n > 0$, the domains $D_1, \ldots, D_n$ with possibly a fixed additional number of hierarchy domains, and a set of general count constraints C on R, the Inverse OLAP problem consists of deciding whether there exists a relation r on $\mathcal{R}$ such that $r \models C$.

Inverse OLAP: Definition and Complexity

Problem

Given a relation scheme $\mathcal{R}(A_1, \ldots, A_n)$ with $n > 0$, the domains $D_1, \ldots, D_n$ with possibly a fixed additional number of hierarchy domains, and a set of general count constraints C on R, the Inverse OLAP problem consists of deciding whether there exists a relation r on $\mathcal{R}$ such that $r \models C$.

Theorem

The Inverse OLAP problem is in NEXP.

Inverse OLAP: Definition and Complexity

Problem

Given a relation scheme $\mathcal{R}(A_1, \ldots, A_n)$ with $n > 0$, the domains $D_1, \ldots, D_n$ with possibly a fixed additional number of hierarchy domains, and a set of general count constraints C on R, the Inverse OLAP problem consists of deciding whether there exists a relation r on $\mathcal{R}$ such that $r \models C$.

Theorem

The Inverse OLAP problem is in NEXP.

Complexity Types

- program complexity: domain sizes are constant
- data complexity: the sizes of $\mathcal{R}$ and of C are constant
- combined complexity: all sizes are not constant

3. Count Constraints for Aggregate Data Exchange

Aggregate data exchange for preserving privacy

We first recall the classical data exchange setting:

Aggregate data exchange for preserving privacy

We first recall the classical data exchange setting:

$$(S, T, \Sigma_{st}, \Sigma_t)$$

Aggregate data exchange for preserving privacy

We first recall the classical data exchange setting:

$$(S, T, \Sigma_{st}, \Sigma_t)$$

where

- S and T are source and target relational scheme, respectively.

Aggregate data exchange for preserving privacy

We first recall the classical data exchange setting:

$$(S, T, \Sigma_{st}, \Sigma_t)$$

where

- S and T are source and target relational scheme, respectively.
- Σ_{st} and Σ_t are source-to-target or target dependencies of the form:

Aggregate data exchange for preserving privacy

We first recall the classical data exchange setting:

$$(S, T, \Sigma_{st}, \Sigma_t)$$

where

- S and T are source and target relational scheme, respectively.
- Σ_{st} and Σ_t are source-to-target or target dependencies of the form:
  - $\forall \mathbf{X}(\phi_S(\mathbf{X}) \rightarrow \chi_T(\mathbf{X}))$
  - $\forall \mathbf{X}(\phi_T(\mathbf{X}) \rightarrow \chi_T(\mathbf{X}))$

Aggregate data exchange for preserving privacy

We first recall the classical data exchange setting:

$$(S, T, \Sigma_{st}, \Sigma_t)$$

where

- S and T are source and target relational scheme, respectively.
- Σ_{st} and Σ_t are source-to-target or target dependencies of the form:
  - $\forall \mathbf{X}(\phi_S(\mathbf{X}) \rightarrow \chi_T(\mathbf{X}))$
  - $\forall \mathbf{X}(\phi_T(\mathbf{X}) \rightarrow \chi_T(\mathbf{X}))$
- Our target relational database scheme consists of a unique relation scheme: $\mathcal{R}(T, I, B, S, A)$

Aggregate data exchange for preserving privacy

We first recall the classical data exchange setting:

- S and T are source and target relational scheme, respectively.
- Σ_{st} and Σ_t are source-to-target or target dependencies of the form:
  - $\forall \mathbf{X}(\phi_S(\mathbf{X}) \rightarrow \chi_T(\mathbf{X}))$
  - $\forall \mathbf{X}(\phi_T(\mathbf{X}) \rightarrow \chi_T(\mathbf{X}))$
- Our target relational database scheme consists of a unique relation scheme: $\mathcal{R}(T, I, B, S, A)$
- The source relational database scheme consists of three relation schemes: $\mathcal{TR}(T, I, B)$, $\mathcal{ST}(T, S)$ and $\mathcal{AR}(S, A)$

Aggregate data exchange for preserving privacy

Aggregate data exchange for preserving privacy

We use the classical setting to implement two natural joins of the source relations instead of only one so that we can later perform a permutation of transactions inside every store:

$$\mathcal{TR}(T, I, B) \wedge \mathcal{ST}(T, S) \wedge \mathcal{AR}(S, A) \rightarrow \exists \hat{T} \ \ \mathcal{R}(\hat{T}, I, B, S, A)$$

$$\mathcal{ST}(T, S) \rightarrow \exists I, B, A \ \ \mathcal{R}(T, I, B, S, A)$$

Aggregate data exchange for preserving privacy

We use the classical setting to implement two natural joins of the source relations instead of only one so that we can later perform a permutation of transactions inside every store:

$$\mathcal{TR}(T, I, B) \wedge \mathcal{ST}(T, S) \wedge \mathcal{AR}(S, A) \rightarrow \exists \hat{T} \ \ \mathcal{R}(\hat{T}, I, B, S, A)$$

$$\mathcal{ST}(T, S) \rightarrow \exists I, B, A \ \ \mathcal{R}(T, I, B, S, A)$$

Count constraint enforcing that the total number of tuples in $\mathcal{TR}$ is equal to the total number of tuples in $\mathcal{R}$:

$$s = \#(\{T, I, B : \mathcal{TR}(T, I, B)\}) \ \rightarrow \\ s = \#(\{T, I, B, S, A : \ \ \mathcal{R}(T, I, B, S, A)\})$$

Aggregate data exchange for preserving privacy

We use the classical setting to implement two natural joins of the source relations instead of only one so that we can later perform a permutation of transactions inside every store:

$$\mathcal{TR}(T, I, B) \wedge \mathcal{ST}(T, S) \wedge \mathcal{AR}(S, A) \rightarrow \exists \hat{T} \ \ \mathcal{R}(\hat{T}, I, B, S, A)$$

$$\mathcal{ST}(T, S) \rightarrow \exists I, B, A \ \ \mathcal{R}(T, I, B, S, A)$$

Count constraint enforcing that the total number of tuples in $\mathcal{TR}$ is equal to the total number of tuples in $\mathcal{R}$:

$$s = \#(\{T, I, B : \mathcal{TR}(T, I, B)\}) \ \rightarrow \\ s = \#(\{T, I, B, S, A : \ \ \mathcal{R}(T, I, B, S, A)\})$$

Count constraint imposing that the original structure of transactions is preserved, except for transactions IDs permutation:

$$\mathcal{ST}(T, S) \wedge s = \{I, B : \mathcal{TR}(T, I, B)\}) \ \rightarrow \\ \exists \hat{T} \ (\ s = \{I, B : \exists A \ \ \mathcal{R}(\hat{T}, I, B, S, A)\}\ )$$

Data exchange to an OLAP scheme

Let now $\mathcal{R}(T, I, B, S, A)$ be the source scheme.
The target scheme is an OLAP scheme $\mathcal{SN}(S, I, B, N)$ that, for every store, represents in N the total number of item-brand pairs that are in all transactions of that store.
Count constraint aggregating data in the target relation:

$$\mathtt{n} = \#(\{\mathtt{T} : \mathcal{R}(\mathtt{T}, \mathtt{i}, \mathtt{b}, \mathtt{s}, \mathtt{a})\}) \rightarrow \mathcal{SN}(\mathtt{s}, \mathtt{i}, \mathtt{b}, \mathtt{n}).$$

Count constraint imposing that the target relation cannot store additional tuples:

$$\mathtt{x} = \#(\{\mathtt{S}, \mathtt{I}, \mathtt{B} : \exists \mathtt{T}, \mathtt{A} \ \mathcal{R}(\mathtt{T}, \mathtt{I}, \mathtt{B}, \mathtt{S}, \mathtt{A})\}) \rightarrow \mathtt{x} = \#(\{\mathtt{S}, \mathtt{I}, \mathtt{B} : \exists \mathtt{N} \ \mathcal{SN}(\mathtt{S}, \mathtt{I}, \mathtt{B}, \mathtt{N})\}$$

Conclusion

Conclusion

- We have introduced a new type a constraints, called count constraints, that extend cardinality constraints

Conclusion

- We have introduced a new type a constraints, called count constraints, that extend cardinality constraints
- We have also introduced a new inverse mining problem, called Inverse OLAP, that is a powerful extension of Inverse Frequent itemsets Mining

Conclusion

- We have introduced a new type a constraints, called count constraints, that extend cardinality constraints
- We have also introduced a new inverse mining problem, called Inverse OLAP, that is a powerful extension of Inverse Frequent itemsets Mining
- The new problem turns out to be NEXP complete under various conditions: combined complexity, program complexity and data complexity

Conclusion

- We have introduced a new type a constraints, called count constraints, that extend cardinality constraints
- We have also introduced a new inverse mining problem, called Inverse OLAP, that is a powerful extension of Inverse Frequent itemsets Mining
- The new problem turns out to be NEXP complete under various conditions: combined complexity, program complexity and data complexity
- We have also shown that our setting for expressing count constraints can be used for performing aggregate data exchange.

Conclusion

- We have introduced a new type a constraints, called count constraints, that extend cardinality constraints
- We have also introduced a new inverse mining problem, called Inverse OLAP, that is a powerful extension of Inverse Frequent itemsets Mining
- The new problem turns out to be NEXP complete under various conditions: combined complexity, program complexity and data complexity
- We have also shown that our setting for expressing count constraints can be used for performing aggregate data exchange.
- Despite the high complexity of the Inverse OLAP problem, an approximate solution can be found in a limited amount of time in some practical situations even for large instances, e.g., by adopting and extending classical techniques used for solving large-scale linear programming.

Conclusion

- We have introduced a new type a constraints, called count constraints, that extend cardinality constraints
- We have also introduced a new inverse mining problem, called Inverse OLAP, that is a powerful extension of Inverse Frequent itemsets Mining
- The new problem turns out to be NEXP complete under various conditions: combined complexity, program complexity and data complexity
- We have also shown that our setting for expressing count constraints can be used for performing aggregate data exchange.
- Despite the high complexity of the Inverse OLAP problem, an approximate solution can be found in a limited amount of time in some practical situations even for large instances, e.g., by adopting and extending classical techniques used for solving large-scale linear programming.
- We are presently working at extending the notion of chase to count constraints (with some limitations)

Thank you!
Any question?