Differential Privacy for Relational Algebra: Improving the Sensitivity Bounds via Constraint Systems[*]

Catuscia Palamidessi
INRIA and LIX, Ecole Polytechnique, France
catuscia@lix.polytechnique.fr

Marco Stronati
Università di Pisa, Italy
marco.stronati@gmail.com

Differential privacy is a modern approach in privacy-preserving data analysis to control the amount of information that can be inferred about an individual by querying a database. The most common techniques are based on the introduction of probabilistic noise, often defined as a Laplacian parametric on the sensitivity of the query. In order to maximize the utility of the query, it is crucial to estimate the sensitivity as precisely as possible.
In this paper we consider relational algebra, the classical language for queries in relational databases, and we propose a method for computing a bound on the sensitivity of queries in an intuitive and compositional way. We use constraint-based techniques to accumulate the information on the possible values for attributes provided by the various components of the query, thus making it possible to compute tight bounds on the sensitivity.

1 Introduction

Differential privacy [6, 7, 8, 9] is a recent approach addressing the privacy of individuals in data analysis on statistical databases. In general, statistical databases are designed to collect global information in some domain of interest, while the information about the particular entries is supposed to be kept confidential. Unfortunately, querying a database might leak information about an individual, because the presence of her record may induce the query to return a different result.
To illustrate the problem, consider for instance a database of people affected by a certain disease, containing data such as age, height, etc. Usually the identity of the people present in the database is supposed to be secret, but if we are allowed to query the database for the number of records which are contained in it, and for – say – the average value of the data (height, age, etc.), then one can infer the precise data of the last person entry in the database, which poses a serious threat to the disclosure of her identity as well.
To avoid this problem, one of the most commonly used methods consists in introducing some *noise* on the answer. In other words, instead of giving the *exact* answer the curator gives an *approximated* answer, chosen randomly according to some probability distribution.
Differential privacy measures the level of privacy provided by such a randomized mechanism by a parameter ε: a mechanism $\mathcal{K}$ is ε-differentially private if for every pair of *adjacent* databases R and R' (i.e. databases which which differ for only one entry), and for every property $\mathcal{P}$, the probabilities that $\mathcal{K}(R)$ and $\mathcal{K}(R')$ satisfy $\mathcal{P}$ differ at most by the multiplicative constant e^{ε}.
The amount of noise that the mechanism must introduce in order to achieve ε differential privacy depends on the so-called *sensitivity* of the query, namely the maximum distance between the answers on two adjacent databases. For instance, one of the most commonly used mechanisms, the *Laplacian*, adds noise

[*]This work has been partially supported by the project ANR-09-BLAN-0169-01 PANDA

to the correct answer y by reporting an approximated answer z according to the following probability density function:

$$P_y(z) = c\, e^{-\frac{|y-z|}{\Delta f}\varepsilon}$$

where Δf is the sensitivity of the query f, and c is a normalization factor. Clearly, the higher is the sensitivity, the greater the noise, in the sense that the above function is more "flat", i.e. we get a higher probability of reporting an answer very different from the exact one.
Of course, there is a trade off between the privacy and the *utility* of a mechanism: the more noise a mechanism adds, the less precise the reported answer, which usually means that the result of querying the database becomes less useful – whatever the purpose.
For this reason, it is important to avoid adding excessive noise: one should add only the noise strictly necessary to achieve the desired level of differential privacy. This means that the sensitivity of the query should be computed as precisely as possible. At the same time, for the sake of efficiency it is desirable that the computation of the sensitivity is done *statically*. Usually this implies that we cannot compute the precise sensitivity, but only approximate it from above. The goal of this paper is to explore a *constraint-based methodology* in order to compute strict upper bounds on the sensitivity.
The language we chose to conduct our analysis is *relational algebra* [4, 5], a formal and well defined model for relational databases, that is the basis for the popular Structured Query Language (SQL, [2]). It consists in a collection of few operators that take relations as input and return relations as output, manipulating rows or columns and computing aggregation of values.
Sensitivity on aggregations often depends on attribute ranges, and these restrictions can be exploited to provide better bounds. To this purpose, we extend mechanisms already in place in modern database systems: In RDBMS (Relational Data Bases Management Systems) implementations, during the creation of a relation, it is possible to define a set of constraints over the attributes of the relation, to further restrict the type information. For instance:

```
Persons{(Name, String)(Age, Integer)} {Age > 0 ∧ Age < 120)}
```

refines the type integer used to express the age of a person in the database, by establishing that it must be a positive value smaller than 120.
Constraints in RDBMS can be defined on single attributes (*column constraints*), or on several attributes (*table constraints*), and help define the structure of the relation, for example by stating whether an attribute is a primary key or a reference to an external key. In addition, so called *check constraints* can be defined, to verify the insertion of correct values. In the example above, for instance, the constraint would avoid inserting an age of, say, 200. Check constraints are particularly useful for our purposes because they restrict the possible values of the attributes, thus allowing a finer analysis of the sensitivity.

Contribution Our contribution is twofold:

1. we propose a method to compute a bound on the sensitivity of a query in relational algebra in a compositional way, and
2. we propose the use of constraints and constraint solvers to refine the method and obtain strict bounds on queries which have aggregation functions at the top level.

Plan of the paper Next section recalls some preliminary notions about relational databases and differential privacy. Section 3 introduces a constraint system and the idea of carrying along the information provided by the constraints as we analyze the query. Section 4 proposes a generalization of differential privacy and sensitivity to generic metric spaces. This generalization will be useful in order to compute the sensitivity of a query in a compositional way. Sections 6, 7 and 8 analyze the sensitivity and the propagation of constraints for the various operators of relational algebra. Finally Section 9 proposes a method to compute a sensitivity bound on the global query, and shows its correctness and the improvement provided by the use of constraints. Section 10 discusses some related work, and Section 11 concludes. Due to space limitations, in this version we have omitted several proof. The interested reader can find them in the full online version of the paper [3].

2 Preliminaries

We recall here some basic notions about relational databases and relational algebra, differential privacy, and sensitivity.

2.1 Relational Databases and Relational Algebra

Relational algebra [4, 5] can be considered as the theoretic foundation of database query languages and in particular of SQL [2]. It is based on the concept of relation, which is the mathematical essence of a (relational) database, and of certain operators on relations like union, intersection, projections, filters, etc.. Here we recall the basic terminology used for relational databases, while the operators will be illustrated in detail in the technical body of the paper.
A relation (or database) based on a certain schema is a collection of tuples (or records) of values. The schema defines the types (domain) and the names (attributes) of these values.

Definition 1 (Relation Schema). *A relation schema $r(a_1 : D_1, a_2 : D_2, \ldots, a_n : D_n)$ is composed of the relation name r and a set of attributes $a_1, a_2, \ldots, a_n$ associated with the domains $D_1, D_2, \ldots, D_n$, respectively. We use the notation $dom(a_i)$ to refer to D_i.*

Definition 2 (Relation). *A relation R on a relation schema $r(a_1 : D_1, a_2 : D_2, \ldots, a_n : D_n)$ is a subset of the Cartesian product $D_1 \times D_2 \times \ldots \times D_n$.*

A relation is thus composed by a set of n-tuples, where each n-tuple τ has the form $(d_1, d_2, \ldots, d_n)$ with $d_i \in D_i$. Note that τ can also be seen as a partial function from attributes to atomic values, i.e. $\tau(a_i) = d_i$. Given a schema, we will denote the universe of possible tuples by $\mathcal{T}$, and the set of all possible relations by $\mathcal{R} = 2^{\mathcal{T}}$.
Relational algebra is a language that operates from relations to relations. Differentially private queries, however, can only return a value, and for this reason they must end with an aggregation (operator γ). Nevertheless it is possible to show that the full power of relational algebra aggregation can be retrieved.

2.2 Differential Privacy

Differential privacy is a property meant to guarantee that the participation in a database does not constitute a threat for the privacy of an individual. More precisely, the idea is that a (randomized) query satisfies differential privacy if two relations that differ only for the addition of one record are almost indistinguishable with respect to the results of the query.
Two relations $R, R' \in \mathcal{R}$ that differ only for the addition of one record are called *adjacent*, denoted by $R \sim R'$. Formally, $R \sim R'$ iff $R \setminus R' = \{\tau\}$ or viceversa $R' \setminus R = \{\tau\}$, where τ is a tuple.

Definition 3 (Differential privacy [6]). *A randomized function $\mathcal{K} : \mathcal{R} \rightarrow Z$ satisfies ε-differential privacy if for all pairs $R, R' \in \mathcal{R}$, with $R \sim R'$, and all $Y \subseteq Z$, we have that:*

$$Pr[\mathcal{K}(R) \in Y] \leq Pr[\mathcal{K}(R') \in Y] \cdot e^{\varepsilon}$$

where $Pr[E]$ represents the probability of the event E.

Differentially private mechanisms are usually obtained by adding some random noise to the result of the query. The best results are obtained by calibrating the noise distribution according to the so-called sensitivity of the query. When the answers to the query are real numbers ($\mathbb{R}$), its sensitivity is defined as follows. (We represent a query as a function from databases to the domain of answers.)

Definition 4 (Sensitivity [6]). *Given a query $Q : \mathcal{R} \rightarrow \mathbb{R}$, the sensitivity of Q, denoted by Δ_Q, is defined as:*

$$\Delta_Q = \sup_{R \sim R'} |\, Q(R) - Q(R')\,|.$$

The above definition can be extended to queries with answers on generic domains, provided that they are equipped with a notion of distance.

3 Databases with constraints

As explained in the introduction, one of the contributions of our paper is to provide strict bounds on the sensitivity of queries by using constraints. For an introduction to the notions of constraint, constraint solver, and constraint system we refer to [1].
In this section we define the constraint system that we will use, and we extend the notion of database schema so to accommodate the additional information provided by the constrains during the analysis of a query.

Definition 5 (Constraint system). *Our constraint system is defined as follows:*

- *Terms are constructed from:*
  - *variables, ranging over the attribute names of the schemas,*
  - *constants, ranging over the domains of the schemas,*
  - *applications of n-ary functions (e.g. $+, \times$) to n terms.*
- *Atoms are applications of n-ary predicates to n terms. Possible predicates are $\geq, \leq, =, \in$.*
- *Constraints are constructed from:*
  - *atoms, and*
  - *applications of logical operators ($\neg, \wedge, \vee, \equiv$) to constraints.*

We denote the composition of constraint by $\otimes$. The solutions of a set of constraints C is the set of tuples that satisfy C, denoted $Sol(C)$. The relations that can be build from $sol(C)$ are denoted by $\mathcal{R}(C) = \mathcal{P}(sol(C))$. The solutions with respect to an attribute a is denoted $sol(C,a)$. Namely, $sol(C,a)$ is the projection on a of $sol(C)$. When the domain is equipped with an ordering relation, we also use $inf(C,a)$ and $sup(C,a)$ to denote the infimum and the supremum values, respectively, of $sol(C,a)$. Typically the solutions and the *inf* and *sup* values can be computed automatically using constraint solvers. Finally we define the diameter of a constraint C as the maximum distance between the solutions of C.

Definition 6 (Diameter). *The diameter of a constraint C, denoted $diam(C)$, is the graph diameter of the adjacency graph $(\mathcal{R}(C), \sim)$ of all possible relations composed by tuples that satisfy C.*

We now extend the classical definition of schema to contain also the set of constraints.

Definition 7 (Constrained schema). *A constrained schema $r(A,C)$ is composed of the relation name r, a set of attributes A, and a set of constraints C. A relation on a constrained schema is a subset of $sol(C)$. We will use $schema(R)$ to represent the constrained relation schema of a relation R.*

The above definition extends the notion of relation schema (Definition 1): In fact here each a_i can be seen as associated with $sol(C,a_i)$. Definition 1 can then be retrieved by imposing as only constraints those of the form $a_i \in D_i$.

Example 1. *Consider the constrained schema* $\texttt{Items}(A,C)$, *where* $A = \{\texttt{Item}, \texttt{Price}, \texttt{Cost}\}$, *and* $C = \{(\texttt{Cost} \leq \texttt{Price} \leq 1000, 0 < \texttt{Cost} \leq 1000)\}$. *The following R is a possible relation over this schema. R:*

Items	{Item, Price, Cost}
	{(Cost ≤ Price ≤ 1000, 0<Cost≤1000)}

$\texttt{Items}(A,C)$

Item	Price	Cost
Oil	100	10
Salt	50	11

R

4 Differential privacy on arbitrary metrics

The classic notions of differential privacy and sensitivity are meant for queries defined on $\mathcal{R}$, the set of all relations on a given schema. The adjacency relation induces a graph structure (where the arcs correspond to the adjacency relation), and a metric structure (where the distance is defined as the distance on the graph).
In order to compute the sensitivity bounds in a compositional way, we need to cope with different structures at the intermediate steps, and with different notions of distance. Consequently, we need to extend the notions of differential privacy and sensitivity to general metric domains.
We start by defining the notions of distance that we will need.

Definition 8 (Hamming distance d_H). *The distance between two relations $R, R' \in \mathcal{R}$ is the Hamming distance $d_H(R,R') = |R \ominus R'|$, the cardinality of the symmetric difference between R and R'. The symmetric difference is defined as $R \ominus R' = (R \setminus R') \cup (R' \setminus R)$.*

Note that d_H coincides with the graph-theoretic distance on the graph induced by the adjacency relation $\sim$, and that $d_H(R,R') = 1 \Leftrightarrow R \sim R'$. We now extend the Hamming distance to tuples of relations, to deal with n-ary operators.

Definition 9 (Distance d_{nH}). *The distance d_{nH} between two tuples of n relations $(R_1,\dots,R_n)$, $(R'_1,\dots,R'_n) \in \mathcal{R}^n$ is defined as: $d_{nH}((R_1,\dots,R_n),(R'_1,\dots,R'_n)) = max(d_H(R_1,R'_1),\dots,d_H(R_n,R'_n))$*

Note that d_{nH} coincides with the Hamming distance for $n = 1$. We chose this maximum metric instead of other distances because it allows us to compute the sensitivity compositionally, while this is not the case for other notions of distance. We can show counterexamples, for instance, for both the Euclidian and the Manhattan distances.

Definition 10 (Distance d_E). *The distance between two real numbers $x, x' \in \mathbb{R}$ is the usual euclidean distance $d_E(x,x') = |x - x'|$.*

In summary, we have two metric spaces over which the relational algebra operators work, namely $(\mathcal{R}^n, d_{nH})$, and $(\mathbb{R}, d_E)$.

Example 2. *Consider a relation R and two tuples τ, π such that $\tau \notin R$ and $\pi \in R$. We define its neighbors R^+ and $R^\pm$, obtained by adding one record, and by changing one record, respectively:*

$$R^+ = R \cup \{\tau\} \qquad\qquad R^\pm = R \cup \{\tau\} \setminus \{\pi\}$$

Their distance from R is : $d_H(R, R^+) = |R \ominus R^+| = 1$, and $d_H(R, R^\pm) = |R \ominus R^\pm| = 2$. Note also that $R \sim R^+$.

Notation 1. *In the following, we will use the notation R^+ to denote $R \cup \{\tau\}$ for a generic tuple τ, with the assumption (unless otherwise specified) that $\tau \notin R$.*

We now adapt the definition of differential privacy to arbitrary metric spaces (X, d) (where X is the support set and d the distance function).

Definition 11 (Differential privacy extended). *A randomized mechanism $\mathcal{K} : X \to \mathcal{Z}$ on a generic metric space (X, d) provides ε-differential privacy if for any $x, x' \in X$, and any set of possible outputs $Y \subseteq Z$,*

$$Pr[\mathcal{K}(x) \in Y] \;\leq\; Pr[\mathcal{K}(x') \in Y] \cdot e^{\varepsilon \cdot d(x,x')}$$

It can easily be shown that Definitions 11 and 3 are equivalent if $d = d_H$.
We now define the sensitivity of a function on a generic metric space.

Definition 12 (Sensitivity extended). *Let (X, d_X) and (Y, d_Y) be metric spaces. The sensitivity Δ_f of a function $f : (X, d_X) \to (Y, d_Y)$ is defined as*

$$\Delta_f \;=\; \sup_{\substack{x,x' \in X \\ x \neq x'}} \frac{d_Y(f(x), f(x'))}{d_X(x, x')}$$

Again, we can show that Definitions 12 and 4 are equivalent if $d_X = d_{nH}$ (proof in full version [3]). This more general definition makes clear that the sensitivity of a function is a measure of how much it increases distances from its inputs to its outputs.
As a refinement of the definition of sensitivity, we may notice that this attribute does not depend on the function alone, but also on the domain, where the choice of x, x' ranges to compute the supremum. In our framework this is particularly useful because we have a very precise description of the restrictions on the domain of an operator, thanks to its input constrained schema (Def 7).

Definition 13 (Sensitivity constrained). *Given a function $f : (X, d_X) \to (Y, d_Y)$, and a set of constraints C on X, the sensitivity of f with respect to C is defined as*

$$\Delta_f(C) \;=\; \sup_{\substack{x,x' \in sol(C) \\ x \neq x'}} \frac{d_Y(f(x), f(x'))}{d_X(x, x')}$$

The introduction of constraints, in addition to an improved precision, allows us to define conveniently function composition. It should be noted that when combining two functions $f \circ g$, where $g : (Y, d_Y) \to (Z, d_Z)$, the domain of g actually depends on the restrictions introduced by f and we can take this into account maximizing over $y, y' \in sol(C \otimes C_f)$, that is the domain obtained combining the initial constraint C and the constraint introduced by f.

5 Operators

We now proceed to compute a bound on the sensitivity of each relational algebra operator through a static analysis that depends only on the relation schema the operator is applied to, and not on its particular instances.
From a static point of view each operator will be considered as a transformation from schema to schema (instead of a transformation from relations to relations): they may add or remove attributes, and modify constraints.
The following analysis is split in operators $\mathtt{op} : (\mathcal{R}^n, d_{nH}) \to (\mathcal{R}, d_H)$, with n equals 1 or 2, and aggregation $\gamma_f : (\mathcal{R}, d_H) \to (\mathbb{R}, d_E)$. In the sensitivity analysis of the formers, given they work only on Hamming metrics, we are only interested in their effect on the number of rows. In our particular case, these relational algebra operators treats all rows equally, without considering their content. This simplification grants us the following property:

Proposition 1. *If* $\mathtt{op} : (\mathcal{R}, d_H) \to (\mathcal{R}, d_H)$ *and C is an arbitrary set of constraints*

$$\Delta_{\mathtt{op}}(C) = \sup_{\substack{R,R' \in \mathcal{R}(C) \\ R \neq R'}} \frac{d_H(\mathtt{op}(R), \mathtt{op}(R'))}{d_H(R,R')} = \min\left(\Delta_{\mathtt{op}}(\emptyset), diam(C \otimes C_{\mathtt{op}})\right)$$

(The proposition holds analogously for the binary case). This property, that does not hold for general functions, allows us in the case of relational algebra to decouple the computation of sensitivity from the constraint system, and solve them separately. $\Delta_{\mathtt{op}}(\emptyset)$ (from now on just $\Delta_{\mathtt{op}}$) can be seen as the sensitivity intrinsic to each operator, the maximum value of sensitivity the operator can cause, when the constraints are loose enough [1] to be omitted. While $diam(C \otimes C_{\mathtt{op}})$, the diameter of the co-domain of the operator, limits the maximum distance the operator can produce, that is the numerator in the distances ratio.

6 Row operators

In this section we consider a first group of operators of type $(\mathcal{R}^n, d_{nH}) \to (\mathcal{R}, d_H)$ with $n = 1, 2$, which are characterized by the fact that they can only add or remove tuples, not modify their attributes. Indeed the header of the resulting relation maintains the same set of attributes and only the relative constraints may be modified.

6.1 Union $\cup$

The union of two relation is the set theoretic union of two set of tuples with the same attributes. The example below illustrates this operation:

Name	Age	Height
John	30	180
Tim	10	100

$\cup$

Name	Age	Height
Alice	45	160
Tim	10	100

$=$

Name	Age	Height
John	30	180
Tim	10	100
Alice	45	160

The union of two relations may reduce their distance, leave it unchanged or in the worst case it could double it, so the sensitivity of union is 2.

Proposition 2. *The union has sensitivity* 2*:* $\Delta_{\cup} = 2$.

[1]for all possible domains the sensitivity can't be greater.

Proof. If $d_{2H}((R_1,R_2),(R_3,R_4)) = 1$ then we have two cases

a) $R_3 = R_1^+, R_4 = R_2$ or $R_3 = R_1, R_4 = R_2^+$. For the symmetry of distance only one case needs to be considered:

$$|(R_1 \cup R_2) \ominus (R_1^+ \cup R_2)| = \begin{cases} 0 & \tau \in R_2 \\ 1 & o.w. \end{cases}$$

The only difference is the tuple τ. If $\tau \in R_2$ then τ would be in both results, leading to identical relations, thus reducing the distance to zero. If $\tau \notin R_2$ then τ will again be the only difference between the results, thus resulting into distance 1.

b) $R_3 = R_1^+, R_4 = R_2^+$

$$|(R_1 \cup R_2) \ominus (R_1^+ \cup R_2^+)| = \begin{cases} 0 & \tau_1 \in R_2 \wedge \tau_2 \in R_1 \\ 1 & \tau_1 \in R_2 \vee \tau_2 \in R_1 \\ 2 & \tau_1 \notin R_2 \wedge \tau_2 \notin R_1 \end{cases}$$

In this case we have two records differing, τ_1 and τ_2, and in the worst case they may remain different in the results, giving a final sensitivity of 2 for the operator. □

Definition 14 (Constraints for union). *Let schema*$(R_1) = (A, C_1)$ *and schema*$(R_2) = (A, C_2)$*. Then schema*$(R_1 \cup R_2) = (A, C_1 \vee C_2)$.

6.2 Intersection $\cap$

The intersection of two relation is the set theoretic intersection of two set of tuples with the same attributes.
As for the union, the intersection applied to arguments at distance 1 may result in a distance 0, 1 or 2.

Proposition 3. *The intersection has sensitivity* 2*:* $\Delta_\cap = 2$.

Proof. Similar to the case of Proposition 2. □

Definition 15 (Constraints for intersection). *Let schema*$(R_1) = (A, C_1)$ *and schema*$(R_2) = (A, C_2)$*. Then schema*$(R_1 \cap R_2) = (a, C_1 \wedge C_2)$.

6.3 Difference $\setminus$

The difference of two relation is the set theoretic difference of two set of tuples with the same attributes.
As in the case of the union, the difference applied to arguments at distance 1 may result in a distance 0, 1 or 2.

Proposition 4. *The set difference has sensitivity* 2*:* $\Delta_\setminus = 2$.

Proof. Similar to the case of Proposition 2. □

Definition 16 (Constraints for set difference). *Let schema*$(R_1) = (A, C_1)$ *and schema*$(R_2) = (A, C_2)$*. Then schema*$(R_1 \setminus R_2) = (A, C_1 \wedge (\neg C_2))$.

6.4 Restriction σ

The restriction operator $\sigma_\varphi(R)$ removes all rows not satisfying the condition φ (typically constructed using the predicates $=, \neq, <, >$ and the logical connectives $\vee, \wedge, \neg$), over a subset of R attributes.
As an example, consider the following SQL program that removes all people whose age is smaller than 20 or whose height is greater than 180. The table illustrates an example of application of the corresponding restriction $\sigma_{Age \geq 20 \wedge Height < 180}$.

```
SELECT *
FROM   R
WHERE  Age>=20 AND Height<=180
```

$$\sigma_{Age \geq 20 \wedge Height < 180}\left(\begin{array}{lll} \text{Name} & \text{Age} & \text{Height} \\ \hline \text{John} & 30 & 180 \\ \text{Tim} & 10 & 100 \\ \text{Alice} & 45 & 160 \\ \text{Natalie} & 20 & 175 \end{array}\right) = \begin{array}{lll} \text{Name} & \text{Age} & \text{Height} \\ \hline \text{Alice} & 45 & 160 \\ \text{Natalie} & 20 & 175 \end{array}$$

The restriction can be expressed in terms of set difference: $\sigma_\varphi(R) = R \setminus \{\tau \mid \neg\varphi(\tau)\}$. However the sensitivity is different because the operator is unary, the second argument is fixed by the condition φ

Proposition 5. *The restriction has sensitivity* 2*:* $\Delta_{\sigma_\varphi} = 1$.

Definition 17 (Constraints for restriction). *Let* $schema(R) = (A, C)$ *and* $A' \subseteq A$. *Then define* $schema(\sigma_{\varphi(A')}(R)) = (A, C \wedge \varphi(A'))$.

7 Attribute operators

The following set of operators, unlike those analyzed so far, can affect the number of tuples of a relation, as well as its attributes.

7.1 Projection π

The projection operator $\pi_{a_1,\ldots,a_n}(R)$ eliminates the columns of R with attributes other than $a_1, \ldots, a_n$, and then deletes possible duplicates, thus reducing distances or leaving them unchanged. It is the opposite of the restricted Cartesian product $\times_1$ which will be presented later.
The following example illustrates the use of the projection. Here, the attribute to preserve are `Name` and `Age`.

```
SELECT Name,Age
FROM   R
```

$$\pi_{Name,Age}\left(\begin{array}{lll} \text{Name} & \text{Age} & \text{Car} \\ \hline \text{John} & 30 & \text{Ford} \\ \text{John} & 30 & \text{Renault} \\ \text{Alice} & 45 & \text{Fiat} \end{array}\right) = \begin{array}{ll} \text{Name} & \text{Age} \\ \hline \text{John} & 30 \\ \text{Alice} & 45 \end{array}$$

Proposition 6. *The projection has sensitivity* 1*:* $\Delta_\pi = 1$.

Proof. $$|\pi_{a_1,\ldots,a_n}(R) \ominus \pi_{a_1,\ldots,a_n}(R^+)| = \left\{ \begin{array}{ll} 0 & \exists \rho \in R.\, \forall i \in \{1,\ldots,n\}\ \rho(a_i) = \tau(a_i) \\ 1 & o.w. \end{array} \right.$$ □

Definition 18 (Constraints for projection). *Let* $schema(R) = (A, C)$ *and* $A' \subseteq A$. *Then* $schema(\pi_{A'}(R)) = (A', C)$.

7.2 Cartesian product

The Cartesian product of two relation is the set theoretic Cartesian product of two set of tuples with different attributes, with the exception that in relations the order of attributes does not count, thus making the operation commutative. The following example illustrate this operation.

Name	Age	Height
John	30	180
Alice	45	160

$\times$

Car	Owner
Fiat	Alice
Ford	Alice

$=$

Name	Age	Height	Car	Owner
John	30	180	Fiat	Alice
John	30	180	Ford	Alice
Alice	45	160	Fiat	Alice
Alice	45	160	Ford	Alice

This operator may seem odd in the context of a query language, but it is in fact the base of the *join*, the operator to merge the information of two relations.

$$R \underset{R.a_i=T.a_i}{\bowtie} T = \sigma_{R.a_i=T.a_i}(R \times T)$$

We analyze now the sensitivity of the Cartesian product.

One record $\times_1$ We first consider a restricted version $\times_1$, where on one side we have a single tuple.

Proposition 7. *The operator* $\times_1$ *has sensitivity* 1*:* $\Delta_{\times_1} = 1$.

N records $\times$ We consider now the full Cartesian product operator. It is immediate to see that a difference of a single row can be expanded to an arbitrary number of records, thus causing and unbounded sensitivity.

Proposition 8. *The (unrestricted) Cartesian product has unbounded sensitivity.*

We now define how constraints propagate through Cartesian product:

Definition 19 (Constraints for product). *Let* $schema(R_1) = (A_1, C_1)$ *and* $schema(R_2) = (A_2, C_2)$*. Then* $schema(R_1 \times R_2) = (A_1 \cup A_2, C_1 \wedge C_2)$*.*

7.3 Restricted $\times$

The effect of Cartesian product is to expand each record with a block of records, a behavior clearly against our objective of distance-preserving computations. However we propose some restricted versions of the operator in order to maintain its functionality to a certain extent:

- $\times_n$: product with blocks of a fixed n size, to obtain n sensitivity. In this case n representative elements can be chosen from the relation, the definition of policies to pick these elements is left to future developments.
- $\times_\gamma$: a new single record is built as an aggregation of the relation, through the operator $\emptyset\gamma f$ (presented later), thus falling in the case of $\times_1$ sensitivity.
- a mix the two approaches could be considered, building n aggregations, possibly using the operator ${}_{\{a_i\}}\gamma_f$ (presented later).

In both approaches the rest of the query can help to select the right records from the block, for example an external restriction could be anticipated.

8 Aggregation γ

The classical relational algebra operator for aggregation ${}_{\{a_1,\ldots,a_m\}}\,\gamma\,_{\{f_1,\ldots,f_k\}}(R)$ performs the following steps:

- it partitions R, so that each group has all the tuples with the same values for each a_i,
- it computes all f_i for each group,
- it returns a single tuple for each group, with the values of a_i and of f_i.

The most common function founds on RDBMS are `count,max,min,avg,sum` and we will restrict our analysis to these ones. The following example illustrates how we can use an aggregation operator to know, for each type of `Car`, how many people own it and what is their average height.

```
SELECT  Car, Count(*), Avg(Height)
FROM    R
GROUPBY Car
```

$$
{}_{\{Car\}}\gamma_{\{Count, Avg(Height)\}}\left(\begin{array}{llll} \text{Name} & \text{Age} & \text{Height} & \text{Car} \\ \hline \text{Alice} & 45 & 160 & \text{Ford} \\ \text{John} & 30 & 180 & \text{Fiat} \\ \text{Frank} & 45 & 165 & \text{Renault} \\ \text{Natalie} & 20 & 170 & \text{Ford} \end{array}\right) = \begin{array}{lll} \text{Car} & \text{Count} & \text{Avg(Height)} \\ \hline \text{Ford} & 2 & 165 \\ \text{Fiat} & 1 & 180 \\ \text{Renault} & 1 & 165 \end{array}
$$

In the domain of differential privacy special care must be taken when dealing with this operator as it is in fact the point of the query in which our analysis of sensitivity ends and the noise must be added to the result of the function application.
A differentially private query should return a single value, in our case in $\mathbb{R}$, and the only queries that statically guarantee this property are those ending with the operator ${}_{\emptyset}\gamma_f : (\mathcal{R}, d_H) \to (\mathbb{R}, d_E)$ (from here on abbreviated γ_f), that apply only one function f to the whole relation without grouping. For this reason we will ignore grouping for now, and focus on queries of the form ${}_{\emptyset}\gamma_f(Q)$ where Q is a sub-query without aggregations. It is however possible to recover the original ${}_{A}\gamma_F$ behavior and use it in sub-queries.

8.1 Functions

In this section we analyze the sensitivity of the common mathematical functions `count,sum, max,min` and `avg`. The application of functions coincide with the change of domain, in fact they take as input a relation in $(\mathcal{R}, d_H)$ and return a single number in $(\mathbb{R}, d_E)$, (not to be confused with a relation with a single tuple, which also contains a single value).
Extending standard results [6], we can prove that, when $f =$ `count,sum,max,min,avg` then $\Delta_f(C)$ can be computed as follows:

Proposition 9.

$$
\begin{aligned}
\Delta_{\texttt{count}}(C) &= 1 \\
\Delta_{\texttt{sum}_{a_i}}(C) &= \max\{|\sup(C,a_i)|, |\inf(C,a_i)|\}
\end{aligned}
\qquad
\begin{aligned}
\Delta_{\texttt{max}_{a_i}}(C) &= |\sup(C,a_i) - \inf(C,a_i)| \\
\Delta_{\texttt{min}_{a_i}}(C) &= |\sup(C,a_i) - \inf(C,a_i)| \\
\Delta_{\texttt{avg}_{a_i}}(C) &= \frac{|\sup(C,a_i) - \inf(C,a_i)|}{2}
\end{aligned}
$$

8.2 Exploiting the constraint system

The sensitivity of aggregation functions, as shown above, depends on the range of the values of an attribute, so clearly it is important to compute the range as accurately as possible.

The usual approach is to consider the bounds given by the domain of each attribute. In terms of constraint system, this corresponds to consider the solutions of the constraint $C_I = a_1 \in D_1 \wedge a_2 \in D_2 \wedge \ldots \wedge a_n \in D_n$. I.e. the standard approach computes the sensitivity of aggregation functions for an attribute a on the basis of $\sup(C_I, a)$ and $\inf(C_I, a)$.
In our proposal we also use C_I: for us it is the initial constraint, at the beginning of the analysis of the query. The difference is that our approach updates this constraints with information provided by the various components of the query, and then exploits this information to compute more accurate ranges for each attribute. The following example illustrates the idea.

Example 3. *Assume that* $schema(R) = (\{Weight, Height\}, C_I)$*, and that the domain for* $Weight$ *is* $[0,150]$ *and for* $Height$ *is* $[0,200]$*. The following query asks the average weight of all the individuals whose weight is below the height minus* 100*.*

$$\gamma_{avg(Weight)}(\sigma_{Weight \leq Height-100}(R))$$

Below we show the initial constraint C_I *and the constraints* C_Q *computed by taking into account the condition of* σ*. Compare the sensitivity computed using* C_I *with the one computed using* C_Q*: They differ because in* C_Q *the max value of* $Weight$ *is* 100*, while in* C_I *is* 150*.*

$$C_I = \{W \in [0,150] \wedge H \in [0,200]\} \qquad \Delta(C_I, \gamma_{avg(W)}) = \frac{|max(C_I,W) - min(C_I,W)|}{2} = 75$$

$$C_Q = \{W \in [0,150] \wedge H \in [0,200] \wedge W \leq H - 100\} \qquad \Delta(C_Q, \gamma_{avg(W)}) = \frac{|max(C_Q,W) - min(C_Q,W)|}{2} = 50$$

Hence exploiting the constraints generated by the query can lead to a significant reduction of the sensitivity.

8.3 Constraints generated by the functions

We now define how to add new constraints for the newly created attributes computed by the functions.

Definition 20 (Constraints for functions). *Let* $schema(R) = (A, C)$*,* $A' \subseteq A$ *and* $F = \{f_1(a_1), \ldots, f_n(a_n)\}$*, where* $a_1, \ldots, a_n \in A$*. Then* $schema(_{A'}\gamma_F(R)) = (A' \cup \{a_{f_1} \ldots a_{f_n}\},\ C \wedge c_{f_1} \wedge \ldots \wedge c_{f_n})$*, where:*

$$c_{f_i} = \begin{cases} min(C, a_i) \leq & a_{f_i} \quad \leq sup(C, a_i) \quad if\ f_i = max/min/avg \\ 0 \leq & a_{f_i} \quad\quad\quad\quad\quad\quad\quad\ \ if\ f_i = sum/count \end{cases}$$

9 Global sensitivity

We have concluded the analysis for all operators of relational algebra, and we now define the sensitivity of the whole query in a compositional way.
For the computation of the sensitivity, we need to take into account the constraint generated by it. We start by showing how to compute this constraint, in the obvious (compositional) way. Remember that we have already defined the constraints generated by each relational algebra operator in Sections 6, 7 and 8.

Definition 21 (Constraint generated by an intermediate query). *The global constraint generated by an intermediate query* Q *on relations with relational schema* $r(a_1 : D_1, a_2, D_2, \ldots, a_n : D_n)$ *is defined statically as:*

$$C_Q = schema(Q(R))$$

where R *is any relation such that* $schema(R) = (\{a_1, a_2, \ldots, a_n\}, C_I)$*, with* $C_I = a_1 \in D_1 \wedge a_2 \in D_2 \wedge \ldots \wedge a_n \in D_n$*.*

We assume, the top-level operator in a query is an aggregation γ_f, followed by a query composed freely using the other operators. We now show how to compute the sensitivity of the latter. Since it is a recursive definition, for the sake of elegance we will assume an identity query *Id*.

Definition 22 (Intermediate query sensitivity). *Assume* $\texttt{op} : (\mathcal{R}^n, d_{nH}) \to (\mathcal{R}, d_H)$ *and* $C_{\texttt{op}}$ *the constraint obtained* after *the application of* $\texttt{op}$*:*

$$\begin{array}{lcll} S(Id) & = & \min(1, diam(C_{Id})) & \textit{base case} \\ S(\texttt{op} \circ Q) & = & \min\left(\Delta_{\texttt{op}} \cdot S(Q), diam(C_{\texttt{op}\circ Q})\right) & \textit{if } n = 1 \\ S(\texttt{op} \circ (Q_1, Q_2)) & = & \min\left(\Delta_{\texttt{op}} \cdot \max(S(Q_1), S(Q_2)), diam(C_{\texttt{op}\circ(Q_1,Q_2)})\right) & \textit{if } n = 2 \end{array}$$

where $\texttt{op}$ *can be any of* $\cup, \cap, \setminus, \sigma, \pi, \times, \times_1$ *and the (classic)* ${}_A\gamma_F$*.*

We are now ready to define the global sensitivity of the query:

Definition 23 (global sensitivity). *The global sensitivity GS of a query* $\gamma_f(Q)$ *is defined as:*

$$GS(\gamma_f(Q)) \quad = \quad \begin{cases} \Delta_f(C_Q) \cdot S(Q) & \textit{if } f = \texttt{count}, \texttt{sum}, \texttt{avg} \\ \Delta_f(C_Q) & \textit{if } f = \texttt{max}, \texttt{min} \end{cases}$$

The following theorem, (proof in full version [3]), expresses the soundness and the strictness of the bound computed with our method.

Theorem 1 (Soundness and strictness). *The sensitivity bound computed by* $GS(\cdot)$ *is sound and strict. Namely:*

$$GS(\gamma_f(Q)) \; = \; \Delta_{\gamma_f(Q)}$$

10 Related Work

The field of privacy in statistical databases has often been characterized by ad-hoc solutions or algorithms to solve specific cases [7]. In recent years however there have been several efforts to develop a general framework to define differentially private mechanisms. In the work [12] the authors have proposed a functional query language equipped with a type system that guarantees differential privacy. Their approach is very elegant, and based on deep logical principles. However, it may be a bit far from the practices of the database community, addressing which is the aim of our paper.
The work that is closest to ours, is the PINQ framework [11], where McSherry extends the LINQ language, with differential privacy functionalities developed by himself, Dwork and others in [10].
Despite this existing implementation we felt the need for a more universal language to explore our ideas, and the mathematically-based framework of relational algebra seemed a natural choice. Furthermore the use of a constraint system to increase the precision of the sensitivity bound was, to out knowledge, never explored before.

11 Conclusions and future work

We showed how a classical language like relational algebra can be a suitable framework for differential privacy and how technology already in place, like check constraints, can be exploited to improve the precision of our sensitivity bounds.

Our analysis showed how the most common operation on databases, the join $\bowtie$, poses great privacy problems and in future we hope to develop solutions to this issue, possibly along the lines already presented in Section 7.3.
In this paper we have considered only the sensitivity, that is the effect on distances of operators, while another interesting aspect would be to compute the effect on the ε exponent as explored in [11], and possibly propose convenient strategies to query as much as possible over disjoint data sets.

References

[1] Krzysztof R. Apt (2003): *Principles of constraint programming*. Cambridge University Press.

[2] Alan Beaulieu (2009): *Learning SQL - Master SQL Fundamentals (2. ed.)*. O'Reilly.

[3] Marco Stronati Catuscia Palamidessi (2012): *Differential Privacy for Relational Algebra: improving the sensitivity bound via constraints systems*. Available at `http://www.lix.polytechnique.fr/~stronati/papers/qapl12.pdf`. Full Version.

[4] E. F. Codd (1970): *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM 13(6), pp. 377–387.

[5] E. F. Codd (1972): *Relational Completeness of Database Sublanguages*. In R. Rustin, editor: Data Base Systems, Prentice-Hall, New Jersey.

[6] Cynthia Dwork (2006): *Differential Privacy*. In: Proc. of ICALP, LNCS, 4052, Springer, pp. 1–12. Available at `http://dx.doi.org/10.1007/11787006_1`.

[7] Cynthia Dwork (2008): *Differential Privacy: A Survey of Results*. In: TAMC, pp. 1–19. Available at `http://dx.doi.org/10.1007/978-3-540-79228-4_1`.

[8] Cynthia Dwork (2011): *A Firm Foundation for Private Data Analysis*. Communications of the ACM 54(1), pp. 86–96. Available at `http://doi.acm.org/10.1145/1866739.1866758`.

[9] Cynthia Dwork & Jing Lei (2009): *Differential Privacy and Robust Statistics*. In: Proc. of STOC, ACM, pp. 371–380. Available at `http://doi.acm.org/10.1145/1536414.1536466`.

[10] Cynthia Dwork, Frank McSherry, Kobbi Nissim & Adam Smith (2006): *Calibrating Noise to Sensitivity in Private Data Analysis*. In: TCC, pp. 265–284. Available at `http://dx.doi.org/10.1007/11681878_14`.

[11] Frank McSherry (2009): *Privacy integrated queries: an extensible platform for privacy-preserving data analysis*. In: SIGMOD Conference, pp. 19–30. Available at `http://doi.acm.org/10.1145/1559845.1559850`.

[12] Jason Reed & Benjamin C. Pierce (2010): *Distance makes the types grow stronger: a calculus for differential privacy*. In: ICFP, pp. 157–168. Available at `http://doi.acm.org/10.1145/1863543.1863568`.