Understanding Database Reconstruction Attacks on Public Data

Simson Garfinkel, John Abowd, and Christian Martindale

U.S. Census Bureau

Abstract. Statistical agencies are mandated to publish summary statistics and micro-data while not providing data users with the ability to derive specific attributes for particular persons or establishments. Traditionally, these privacy guarantees are assured through the use of *Statistical Disclosure Limitation* (SDL) techniques. These techniques are not sufficient to prevent a database reconstruction attack of the sort anticipated by Dinur and Nissim (2003). To illustrate the problem, this paper presents a database reconstruction attack on a hypothetical block for which statistics have been published by an official statistics agency. We then show how the use of formally private noise reduces the privacy risk of the database reconstruction. The paper concludes with a discussion of the implications for the 2020 Census of Population and Housing.

I'm not sure about the use of this term. Maybe "noise introduced to satisfy the requirements of a formal privacy model"

Keywords: database reconstruction attack, SAT solver, privacy, disclosure avoidance

1 Introduction

In 2020 the Census Bureau will conduct the constitutionally mandated decennial Census of Population and Housing, with the goal of counting every person once, and only once, and in the correct place, and to fulfill the Constitutional requirement to apportion the seats in the U.S. House of Representatives among the states according to their respective numbers.

Beyond the Constitutionally mandated role of the decennial census, the US Congress has mandated many other purposes for the data. For example, the U.S. Department of Justice uses block-by-block counts by race for enforcing the Voting Rights Act. More generally, the results of the decennial census, combined with other data, are used to help distribute more than $675 billion in federal funds to states and local organizations.

In addition to collecting and distributing data on the American people, the Census Bureau is charged with protecting the Privacy confidentiality of survey and
responses. Specifically, all Census publications must uphold the confidentiality standard specified by Title 13 of the U.S. Code, which states, in part, that Bureau publications are prohibited from identifying “the data furnished by any particular establishment or individual.”[Title 13, Section 9] This section prohibits the Bureau from publishing respondent names, addresses, or any other information that might identify a specific person or establishment.

Upholding this confidentiality requirement frequently poses a challenge, because many statistical can inadvertently provide information in a way that can be attributed to a particular entity. For example, a survey of salaries in a region might report the average salary earned by people of different occupations. However, if there is only one bricklayer, than reporting the average salary of a bricklayer will allow anyone who sees to statistical product to infer that person's salary. If there are two bricklayers in the region, reporting the average salary will allow each bricklayer to infer the other's salary. Both of these cases would be clear violations of Title 13. The Bureau has traditionally used cell suppression to protect privacy in situations such as this: the cells of statistical tables that result from *small counts* are suppressed and not reported. If totals are reported, cell suppression requires that additional *complementary* cells be suppressed so that the values of the suppressed cells cannot be worked out with simple arithmetic.

In 2003, Dinur and Nissim showed that simple cell suppression is not sufficient to protect the underlying confidential data collected and used by a statistical agency to produce official statistics [2]. To the contrary, they showed that once an agency publishes more than a critical number of statistics, the underlying confidential data can be *reconstructed* by simply finding a consistent set of microdata that, when tabulated, produce the official statistics. One of the big surprises of the 2003 paper is that the number of tabulations required to enable a database reconstruction attack (DRA) is far fewer than might be intuitively expected: this is because of the internal constraints and consistency requirements in the published data.

you may want to also cite follow up work that generalized the results of dinur-nissim to hold in more settings.

While it is mathematically impossible prevent reconstruction of the underlying data, a data publisher can add noise to the published results so that the reconstructed data will not reveal the actual confidential responses that was used to create the published tables. Clearly, as more noise that is added, the respondents will enjoy greater privacy protection, but the data publications will have less resulting accuracy.

This sentence seems to be self inconsistent. Probably, because you have not formally defined reconstruction.

This statement (beginning with "clearly" is not generally true. respondent will enjoy greater privacy only if the noise is crafted carefully so as to provably disable attacks.

So how much noise needs to be added to protect privacy? Three years later, Dwork, McSherry, Nissim and Smith answered that question. In their paper "Calibrating Noise to Sensitivity in Private Data Analysis," [3] the researchers introduced the concept of *differential privacy*. The paper provides a mathematical definition of the privacy loss that persons suffer as a result of a data publication, and proposes a mechanism for determining how much noise needs to be added for any given level of privacy protection.

actually, the level of noise that is required and sufficient in the context of count queries was already analyzed in Dini03, Dwork-Nissim 04, and BDMN05. The 2006 paper introduced differential privacy hence helped put these (and other) understandings within a formal framework.

The 2020 census is expected to count roughly 320 million people living on roughly 8.5 million inhabited blocks, with some blocks having as few as a single person and other blocks having thousands. With this level of scale and diversity, it is difficult to visualize how such a data release might be susceptible to a database reconstruction attack. Nevertheless, with recent improvements in both computing power and big data applications, such reconstructions now pose a significant risk to the confidentiality of microdata that underlies unprotected statistical tables.

Is it really that "computing power and big data applications" make such an attack possible?

To help understand the importance of adopting formal privacy methods, in this article we presents a database reconstruction attack on a much smaller statistical publication: a hypothetical block containing seven people distributed over two households. We show that even a relatively small number of constraints results in an exact solution the blocks' inhabitants. Finally, we show how differential privacy can protect the published data by creating uncertainty. Finally, we discuss implications for the decennial census.

readers may think that recovering info of 7 people is insignificant. You can mention that the attack can be repeated on the block level and is expected to be successful on many, so the total number individuals for which reconstruction poses a threat is very large.

2 An Example Database Reconstruction Attack

To present the attack, we consider the census of a fictional geographic frame (for example, a suburban block), conducted by a fictional statistical agency. For every block, the agency collects each resident's age, sex and race, and publishes a variety of statistics. To simplify the example, the fictional world has only two races, black and white, and two sexes, female and male. The statistical agency is prohibited from publishing the raw microdata, and instead publishes a tabular report (Table 1).

Notice that a substantial amount of information in Table 1 has been suppressed (censored). In this case, the statistical agency's disclosure avoidance rules prohibit it from publishing statistics based on one or two people. This suppression rule is sometimes called "the rule of three," because cells in the report sourced from fewer than three people are suppressed.

Table 1. Fictional statistical data for a fictional block published by a fictional statistics agency. Item numbers are for identification purpose only.

			Age	
Statistic	Group	Count	Median	Mean
1A	total population	7	30	38
2A	female	4	30	33.5
2B	male	3	30	44
2C	Black or African American	4	51	48.5
2D	White	3	24	24
3A	single adults	[redacted]	[redacted]	[redacted]
3B	married adults	4	51	54
4A	Black or African American female	3	36	36.7
4B	Black or African American male	[redacted]	[redacted]	[redacted]
4C	White male	[redacted]	[redacted]	[redacted]
4D	White female	[redacted]	[redacted]	[redacted]
5A	persons under 5 years	[redacted]	[redacted]	[redacted]
5B	persons under 18 years	[redacted]	[redacted]	[redacted]
5C	persons 64 years or over	[redacted]	[redacted]	[redacted]

Note: Married persons must be 15 or over

2.1 Encoding the Constraints

To perform the database reconstruction attack, we view the attributes of the persons living on the block as a collection of free variables. We then extract from the published table a set of constraints. The database reconstruct attack merely finds a set of attributes that are consistent with the constraints. If statistics are highly constrained, the only a single reconstruction will be possible, and that reconstruction should be the same as the underlying microdata used to create original table.

reconstruction?

inkorekt english sentence

For example, statistic 2B states that there are 3 males living in the geography. Because age is reported to the nearest year, and age must necessarily be [0..115], there are only a finite number of possible age combinations, specifically:

$$\binom{116}{3} = \frac{116 \times 115 \times 114}{3 \times 2 \times 1} = 253,460$$

I dont understand. why must the three ages be distinct?

Table 2. The 30 possible ages for which the median is 30 and the mean is 44

a	b	c	a	b	c	a	b	c
1	30	101	11	30	91	21	30	81
2	30	100	12	30	90	22	30	80
3	30	99	13	30	89	23	30	79
4	30	98	14	30	88	24	30	78
5	30	97	15	30	87	25	30	77
6	30	96	16	30	86	26	30	76
7	30	95	17	30	85	27	30	75
8	30	94	18	30	84	28	30	74
9	30	93	19	30	83	29	30	73
10	30	92	20	30	82	30	30	72

However, within the 253,460 possible age combinations, there are 30 combinations that satisfy the constraint of having a median of 30 and a mean of 44 (see Table 1). So by applying the constraints imposed by the published statistical tables, we are able to reduce the possible combinations of ages for the three males from 253,460 to 30.

To mount a full reconstruction attack, an attacker extracts all of these constraints and then creates a single mathematical model that reflects them all. An automated solver can then find an assignment of the variables that satisfies these constraints.

To continue with our example, statistic 1A establishes the universe of the constraint system. Because the block contains 7 people, and there are four attributes for each (age, sex, race and marital status), we create 28 free variables representing those four attributes for each person. These variables are A1..A7 (age), S1..S7 (sex), R1..R7 (race), and M1..M7 (marital status), as shown in Table 3.

Table 3. The variables associated with the database reconstruction attack. The coding for the categorical attributes is presented in the key.

Person	Age	Sex	Race	Marital Status
1	A1	S1	R1	M1
2	A2	S2	R2	M2
3	A3	S3	R3	M3
4	A4	S4	R4	M4
5	A5	S5	R5	M5
6	A6	S6	R6	M6
7	A7	S7	R7	M7

Key:				
female		0		
male		1		
black			0	
white			1	
single				0
married				1

Because the mean age is 38, we know that:

$$A1 + A2 + A3 + A4 + A5 + A6 + A7 = 7 \times 38 \tag{1}$$

We use a language called Sugar [8] to encode the constraints into a form that can be processed by our solver. Sugar represents constraints as *s-expressions* [7]. For example, equation 1 is can be represented this way:

```
; First define the integer variables, with the range 0..115
(int A1 0 115)
(int A2 0 115)
(int A3 0 115)
(int A4 0 115)
(int A5 0 115)
(int A6 0 115)
(int A7 0 115)

; Statistic 1A: Mean age is 30  should be 38?
(= (+ A1 A2 A3 A4 A5 A6 A7)
   (* 7 38)
)
```

Once the constraints in the statistical table are turned into s-expressions, Sugar encodes the s-expressions into a set of Boolean constraints that can be fed into a SAT-solver. For example, each age variable is encoded using uniary notation as 116 Boolean variables. Using this notation, the decimal value 0 is encoded as 116 false Boolean variables, the decimal value 1 is encoded as 1 true and 115 false values, and so on. Although this conversion is not space efficient, it is fast, and the uniary notation makes it easy to encode integer inequalities as simple functions of Boolean variables.

Although Sugar makes it relatively simple to encode constraints that represent counts and means, encoding medians is more complicated. The approach we settled upon requires that we first order the ages of the person records. An unanticipated advantage of ordering the ages is that it eliminates many degenerate solutions that different only in permuted labels for the reconstructed individuals:

this is a bit confusing because it suggests that you are performing some sorting algorithm. what you do is introduce sorting constraints on the ages.

```
;; Assure that the output is sorted by age. This does a good job
;; eliminating dupliate answers that simply have swapped records.
;; This is called "breaking symmetry" in the formal methods literature.
(<= A1 A2)
(<= A2 A3)
(<= A3 A4)
(<= A4 A5)
(<= A6 A7)
```

Having sorted the ages, we know that the middle variable must be median:

same

```
(= A4 30)
```

Sugar has an `if` function that allows us to encode constraints for a subset of the population. Recall that statistic 2B contains three constraints: there are three males, their median age is 30, and their average age is 44. We can use the value `0` to represent a female and `1` to represent a male:

```
(int FEMALE 0)
(int MALE   1)
```

Using the variable S*n* to represent the sex of person n, we then have the constraint:

$$S1 + S2 + S3 + S4 + S5 + S6 + S7 = 3 \tag{2}$$

This can be represented as:

```
(= (+ S1 S2 S3 S4 S5 S6 S7) 3)
```

Now, using the `if` function, it is straightforward to create a constraint for mean age of male persons is 44:

```
(= (+ (if (= S1 MALE) A1 0)    ; average male age = 44
      (if (= S2 MALE) A2 0)
      (if (= S3 MALE) A3 0)
      (if (= S4 MALE) A4 0)
```

```
    (if (= S5 MALE) A5 0)
    (if (= S6 MALE) A6 0)
    (if (= S7 MALE) A7 0)
    )
(* 3 44))
```

We translated Table 1 into 160 individual S-expressions extending over 443 lines. Sugar then translated this into 6,740 Boolean variables consisting of 252,478 clauses in the conjunctive normal form (CNF).

did you include suppressed data as constraints (e.g., there should be 1 or 2 persons of age under 5) It would be nice to demonstrate that suppression does not mean that no information is leaked.

Translating the constraints into CNF allows them to be solved using either a SAT (satisfiability), SMT (satisfiability module theories), or MIPM (mixed integer programming model) solver. There are many such solvers, and many take input in the so-called DIMACS file format, which is a standardized form for representing CNF equations. The DIMACS format was popularized by a series of annual SAT solver competitions; one of the results of these competitions was a tremendous speed-up of SAT solvers over the past two decades. Many solvers can now solve CNF systems with millions of variables and clauses in just a few minutes.

In our case, the open source PicoSAT [1] SAT solver is able to find a solution to the CNF problem in less than 2 seconds on a 2013 MacBook Pro with a 2.8GHz Intel i7 processor and 16GiB of RAM (although the program is not limited by RAM), while the open source Glucose SAT solver can solve the problem in under 0.1 seconds on the same computer. The stark difference between the two solvers shows the speedup possible with an improved solving algorithm.

2.2 Exploring the Solution Universe

Both solvers create a satisfying assignment for the 6,740 Boolean variables. After the solver runs, we can use Sugar to translate these assignments back into integer values of the constructed variables. (SMT and MIPM solvers can represent the constraints at a higher level of abstraction, but for our purposes a SAT solver is sufficient.)

There exists a *solution universe* of all the possible solutions to this set of constraints. If the solution universe contains a single possible solution, then the published statistics completely reveal the underlying confidential data. If there are multiple satisfying solutions, then any element (person) in common between all of the solutions is revealed. If the equations have no solution, the set of published statistics are inconsistent.

Normally SAT, SMT and MIPM solvers will stop when they find a single satisfying solution. One of the advantages of PicoSAT is that it can produce the solution universe of all possible solutions to the CNF problem. However, in this case, there is a single satisfying assignment that produce the statistics in Table 1. That assignment is:

Age	Sex	Race	Marital Status		Solution #1
8	F	B	S		8FBS
18	M	W	S		18MWS
24	F	W	S		24FWS
30	M	W	M	=	30MWM
36	F	B	M		36FBM
66	F	B	M		66FBM
84	M	B	M		84MBM

Table 1 is actually over-constrained: some of the constraints can be dropped while preserving a unique solution. For example, dropping statistic 2A, 2B, 2C or 2D still yields a single solution, but dropping 2A *and* 2B increases the solution universe to eight satisfying solutions. All of these solutions contain the reconstructed microdata records 8FBS, 36FBM, 66FBM and 84MBM. This means that even if statistics 2A and 2B are censored, we can still infer that these four microdata must be present.

Statistical agencies have long used suppression (censoring) in an attempt to provide privacy to those whose attributes are presented in the microdata, although the statistics that they typically drop are those that are based on a small number of persons. How effective is this approach?

Reviewing Table 1, statistic 4A is an obvious candidate for suppression—especially given that statistics 4B, 4C and 4D have already been suppressed to avoid an inappropriate statistical disclosure.

Removing the constraints for statistic 4A increases the number of solutions from 1 to 2:

Solutions without 4A	
Solution #1	Solution #2
8FBS	2FBS
18MWS	12MWS
24FWS	24FWM
30MWM	30MBM
36FBM	36FWS
66FBM	72FBM
84MBM	90MBM

did you include suppression based constraints in deriving these solutions?

3 Defending Against a DRA

There are three approaches for defending against a database reconstruction: publish less statistical data, and apply noise (random changes) to the statistical data being tabulated, or apply noise to the results after the tabulation. We consider them in order below.

Although it might seem that publishing less statistical data is a reasonable defense against the DRA, this choice may severely limit the number of tabulations that can be published. A related problem is that, with even a moderately large population, it may be computationally infeasible to determine when the intersection of all possible reconstructions identifies individuals.

you can refer to papers on auditing where they show that making such decisions can be computationally hard.

A second approach is to apply noise to the data before tabulation. This approach is called *input noise infusion*. For example, each respondent's age might be randomly altered by a small amount. Input noise infusion doesn't prevent database reconstruction, but it limits the value of the reconstructed data by creating uncertainty for each of the reconstructed values.

need to define reconstruction (see comment in intro)

For example, if a random offset in the range of $-2 \ldots +2$ is added to each record of our census and the reconstruction results in individuals of ages (7, 17, 22, 29, 36, 66, 82) or (6, 18, 26, 31, 34, 68, 82). An attacker would presumably take this into account, but they would have no way of knowing if the true age of the youngest person is 6, 7, 8, 9 or 10. Randomness could also be applied to the sex, race and marital status variables. Clearly, the more noise that is added, the better privacy is protected, but the less accurate are the resulting statistics. Considering statistic 1A, input noise infusion might result with a median $28 \ldots 32$ and a mean $36 \ldots 40$.

you may mention that other noise infusion techniques exist (adding noise in the computation).

The third approach is to add noise to the statistics themselves. This is called *output noise infusion*. Whereas input noise infusion applies noise to the microdata directly, output noise infusion applies output to the statistical publications. Output noise infusion impacts database reconstruction in two ways. First, the resulting statistical publication is likely no longer consistent, so the reconstruction of *any* database may no longer be possible. Second, even if a database is reconstructed, it is likely not the correct database.

this is inaccurate, as the reconstruction attack can take the noise infusion process into account!

4 Related Work

In 2003 Irit Dinur and Kobbi Nissim [2] showed that the underlying confidential data of a statistical database can be reconstructed with a surprisingly small number of queries. In practice, most statistical agencies perform these queries themselves when they release statistical tables. Thus, Dinur and Nissim's primary finding is that a statistical agency that publishes too many accurate statistical tables drawn from the same confidential dataset risks inadvertently compromising that dataset unless it takes specific protective measures. This paper is a demonstration of that finding.

Statistical tables create the possibility of database reconstruction because they form a set of constraints for which there is ultimately only one exact solution. Restricting the number or specific types of queries—for example, by suppressing results from a small number of respondents—is often insufficient to prevent access to indirectly identifying information, because the system's refusal to answer a "dangerous" query itself provides the attacker with information. Dinur and Nissim found that, if a database is modeled as a string of n bits, then at least $\sqrt{n}$ bits must be modified by adding noise to protect persons from being identified. In our example, each record is contains 11 bits of data, so the confidential database has 77 bits of information. Each statistic in Table 3 can be

The fact you're getting an exact solution is nice, but reconstruction need not be exact.

this is an inaccurate statement of the DiNi result. I can explain.

modeled as a 4 bit of count, a 7 bit of median, and a 7 bit of mean, for a total of 18 bits; Table 3 releases 126 bits of information.

Kasiviswanathan, Rudelson, and Smith [4] introduced the concept of the linear reconstruction attack, which underlies the DRA. The key concept is that, given nonsensitive data such as zip code or gender, an attacker can construct a matrix of linear equalities that can often be solved in polynomial time. The paper also analyzes a common reconstruction technique known as least squares decoding, where the attacker sets up a goal function to minimize the square of the distance between two databases in order to reconstruct the original database. This paper does not use that attack technique, but instead creates a system of constraints and solves them with a solver that can solve NP-hard problems. Although such a solver requires exponential time in the worst case, in this case it is quite fast.

I think you should also mention some other papers.

5 Conclusion

With the dramatic improvement in both computer speeds and the efficiency of solvers in the last decade, database reconstruction attacks on statistical databases are no longer a theoretical danger. The vast quantity of data products published by statistical agencies each year may give a determined attacker more than enough constraints to reconstruct some or all of a target database and breach the privacy of millions of people. Although a suite of traditional disclosure avoidance technique is often sufficient to defend against a cursory attack, cell suppression and generalization are not secure against this kind of attack. To protect the privacy of respondent data, statistical agencies must use some kind of noise infusion. Formal privacy methods, especially differential privacy, were specifically designed to to control this risk and, as both of our noise-infusion examples illustrate, they do so by systematically expanding the universe of solutions to the DBA constraints. In this expanded universe, the real confidential data are but a single solution, and no evidence in the published data can improve an attacker's guess about which solution is the correct one.

Again, while computing power and efficiency of solvers help, I dont think this is the real reason reconstruction is possible. Probably the main relevant advancement is in a theoretical understanding of privacy.

6 References

References

1. Biere, A.: Picosat essentials. Journal on Satisfiability, Boolean Modeling and Computation (JSAT) (2008)
2. Dinur, I., Nissim, K.: Revealing information while preserving privacy. pp. 202–210. ACM, New York, USA (2003)
3. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Proceedings of the Third Conference on Theory of Cryptography. pp. 265–284. TCC'06, Springer-Verlag, Berlin, Heidelberg (2006), http://dx.doi.org/10.1007/11681878_14

4. Kasiviswanathan, S.P., Rudelson, M., Smith, A.: The power of linear reconstruction attacks. In: Proceedings of the Twenty-fourth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1415–1433. SODA '13, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2013), http://dl.acm.org/citation.cfm?id=2627817.2627919
5. Kong, S., Malec, D.: Cook-levin theorem. University Lecture, University of Wisconsin (2007)
6. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning sat solvers. In: Handbook of Satisfiability. pp. 131–153. IOS Press, Amsterdam, Netherlands (2009)
7. McCarthy, J.: Recursive functions of symbolic expressions and their computation by machine, part i. Commun. ACM 3(4), 184–195 (Apr 1960), http://doi.acm.org/10.1145/367177.367199
8. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling finite linear csp into sat. Constraints 14(2), 254–272 (2009), http://bach.istc.kobe-u.ac.jp/sugar/

7 Appendices

7.1 SAT and SAT Solvers

The Boolean satisfiability problem (SAT) was the first problem to be proven NP-complete [5]. This problem asks, for a given Boolean formula, whether replacing each variable with either True or False can make the formula evaluate to True. A consequence of SAT being NP-complete is that many problems involving constraints on variables can be reduced in polynomial time (i.e., quickly) to an instance of the SAT problem. Once reduced, many such problems can be solved through the use of general-purpose heuristics. Although the SAT problem is not solvable by algorithms in polynomial time, researchers have found many heuristic techniques for expediting this process. SAT solvers combine a variety of these techniques into one complex process, resulting in polynomial time solutions for the SAT problem in many cases.

Modern SAT solvers use a heuristic technique called Conflict-Driven Clause Learning, commonly referred to as CDCL [6]. Briefly, CDCL algorithm is:

1. Assign a value to a variable arbitrarily.
2. Use this assignment to determine values for the other variables in the formula (a process known as unit propagation).
3. If a conflict is found, backtrack to the clause that made the conflict occur and undo variable assignments made after that point.
4. Add the negation of the conflict-causing clause as a new clause to the master formula and resume from step 1.

This process is fast at solving SAT problems because adding conflicts as new clauses has the potential to avoid wasteful "repeated backtracks." Additionally, CDCL and its predecessor algorithm, DPLL, are both provably complete algorithms and will always return either a solution or "Unsatisfiable" if given enough time and memory. Another advantage is that CDCL solvers reuse past work when producing the universe of all possible solutions.

There are a wide variety of SAT solvers available to the public for minimal or no cost. Although a SAT solver requires the user to translate the problem into Boolean formulae before use, programs such as Naoyuki Tamura's Sugar facilitate this process by translating user-input mathematical and English constraints into Boolean formulae automatically.

7.2 Sugar Input

Sugar input is given in a standard Constraint Satisfaction Problem (CSP) file format. A constraint must be given on a single line of the file, but here we separate most constraints into multiple lines for readability. Constraint equations are separated by comments describing what statistics they encode.

Input for the model in this paper is as follows:

```
;; -*- mode: lisp; -*-
;; Define variables and their domains
;; Households (either 1 or 2)
;;
;; HOUSEHOLD  SEX   AGE  RACE  MARITAL   RELATION
;;
;;     1      S=M   A=18 R=W   M=SA        #1
;;     1      S=F   A=24 R=W   M=SA        SPO

;;     2      S=M   A=30 R=W   M=MA        #1
;;     2      S=F   A=36 R=B   M=MA        SPO
;;     2      S=M   A=84 R=B   M=MA        PAR
;;     2      S=F   A=8  R=B   M=SC        CHI
;;     2      S=F   A=66 R=B   M=MA        PAR

;; But we will report this sorted by age:


;; Problem setup
#define USE_2A
#define USE_2B
#define USE_2C
#define USE_2D
#define USE_3B
#define USE_4A
;;#define USE_SOLUTION   // for debugging

;; Sexes (0:female  1:male)
#define FEMALE 0
#define MALE   1
(int S1 FEMALE MALE) (int S2 FEMALE MALE)
(int S3 FEMALE MALE) (int S4 FEMALE MALE)
```

```
(int S5 FEMALE MALE) (int S6 FEMALE MALE)
(int S7 FEMALE MALE)

;; Ages (between 0 and 115 years old)
#define MIN_AGE 1
#define MAX_AGE 115
(int A1 MIN_AGE MAX_AGE) (int A2 MIN_AGE MAX_AGE)
(int A3 MIN_AGE MAX_AGE) (int A4 MIN_AGE MAX_AGE)
(int A5 MIN_AGE MAX_AGE) (int A6 MIN_AGE MAX_AGE)
(int A7 MIN_AGE MAX_AGE)

;; Races (0:black  1:white)
#define BLACK 0
#define WHITE 1
(int R1 BLACK WHITE) (int R2 BLACK WHITE) (int R3 BLACK WHITE) (int R4 BLACK WHITE)
(int R5 BLACK WHITE) (int R6 BLACK WHITE) (int R7 BLACK WHITE)

;; Marital Status (0:single  1:married)
#define SINGLE 0
#define MARRIED 1
(int M1 SINGLE MARRIED) (int M2 SINGLE MARRIED)
(int M3 SINGLE MARRIED) (int M4 SINGLE MARRIED)
(int M5 SINGLE MARRIED) (int M6 SINGLE MARRIED)
(int M7 SINGLE MARRIED)

;; Structural Zeros:
;;; Married people must be over 14. Set the minimum age based on the marriage flag
(< (if (= M1 MARRIED) 14 0) A1)
(< (if (= M2 MARRIED) 14 0) A2)
(< (if (= M3 MARRIED) 14 0) A3)
(< (if (= M4 MARRIED) 14 0) A4)
(< (if (= M5 MARRIED) 14 0) A5)
(< (if (= M6 MARRIED) 14 0) A6)
(< (if (= M7 MARRIED) 14 0) A7)


;; Assure that the output is sorted by age. This does a good job
;; eliminating dupliate answers that simply have swapped records.
;; This is called "breaking symmetry" in the literature.
(<= A1 A2)
(<= A2 A3)
(<= A3 A4)
(<= A4 A5)
(<= A6 A7)
```

```
;; Reported tables

;; Statistic 1A: Total Pop: 7, median=30, mean=38

;; Median age 30
;; The ages are sorted, so A4 must be 30.
(= A4 30)

; mean age: 38
(= (+ A1 A2 A3 A4 A5 A6 A7)
   (* 7 38))

;; Statistic 2A: Female:  n=4, median=30, mean=33.5
#ifdef USE_2A
(= (+ S1 S2 S3 S4 S5 S6 S7) 3)  ;; 4 female (0=female, 1=male)

;; Median age of female is 30
;;
;; To solve this, we create some temporary variables:
;; FEMALEID1 FEMALEID2 FEMALEID3 FEMALEID4
;;                 - the ID number of each female, in order of ages
;; FEMALE_AGE1 FEMALE_AGE2 FEMALE_AGE3 FEMALE_AGE4
;;                 - the age of each female, in order of ages
;;
;; So we kow that the average of FEMALE_AGE2 and FEMALEA3 is 30
;;
;; This is a generic pattern that will be repeated for any cell size of 4

(int FEMALE_ID1 1 4)                     ; must leave room for 5, 6 and 7 to be female
(int FEMALE_ID2 2 5)                     ; must leave room for 1, 6 and 7 to be female
(int FEMALE_ID3 3 6)                     ; must leave room for 1, 2 and 7 to be female
(int FEMALE_ID4 4 7)                     ; must leave room for 1, 2 and 3 to be female

(< FEMALE_ID1 FEMALE_ID2)
(< FEMALE_ID2 FEMALE_ID3)
(< FEMALE_ID3 FEMALE_ID4)

;; Pigeon hole principle. If the IDs are in order, we only need to assure
;; that each ID maps to a female. We add the final equal and a -1 to force
;; an error if there are not enough females.

(= FEMALE_ID1
   (if (= S1 FEMALE) 1
     (if (= S2 FEMALE) 2
       (if (= S3 FEMALE) 3
        (if (= S4 FEMALE) 4
          -1)))))

(= FEMALE_ID2
   (if (and (= S2 FEMALE) (< FEMALE_ID1 2)) 2
     (if (and (= S3 FEMALE) (< FEMALE_ID1 3)) 3
       (if (and (= S4 FEMALE) (< FEMALE_ID1 4)) 4
         (if (and (= S5 FEMALE) (< FEMALE_ID1 5)) 5
           -1)))))

(= FEMALE_ID3
   (if (and (= S3 FEMALE) (< FEMALE_ID2 3)) 3
     (if (and (= S4 FEMALE) (< FEMALE_ID2 4)) 4
       (if (and (= S5 FEMALE) (< FEMALE_ID2 5)) 5
         (if (and (= S6 FEMALE) (< FEMALE_ID2 6)) 6
           (if (and (= S7 FEMALE) (< FEMALE_ID2 7)) 7
             -1))))))

(= FEMALE_ID4
   (if (and (= S4 FEMALE) (< FEMALE_ID3 4)) 4
     (if (and (= S5 FEMALE) (< FEMALE_ID3 5)) 5
       (if (and (= S6 FEMALE) (< FEMALE_ID3 6)) 6
         (if (and (= S7 FEMALE) (< FEMALE_ID3 7)) 7
           -1)))))

;; Create temporary variables for the ages of these females
;; This uses the Sugar
(int FEMALE_AGE1 MIN_AGE MAX_AGE)
(int FEMALE_AGE2 MIN_AGE MAX_AGE)
(int FEMALE_AGE3 MIN_AGE MAX_AGE)
(int FEMALE_AGE4 MIN_AGE MAX_AGE)

(< FEMALE_AGE1 FEMALE_AGE2)
(< FEMALE_AGE2 FEMALE_AGE3)
(< FEMALE_AGE3 FEMALE_AGE4)

;; Fix the female ages to the person ages
(element FEMALE_ID1 (A1 A2 A3 A4 A5 A6 A7) FEMALE_AGE1)
(element FEMALE_ID2 (A1 A2 A3 A4 A5 A6 A7) FEMALE_AGE2)
(element FEMALE_ID3 (A1 A2 A3 A4 A5 A6 A7) FEMALE_AGE3)
(element FEMALE_ID4 (A1 A2 A3 A4 A5 A6 A7) FEMALE_AGE4)

;; The average of these is 30, so their sum is 60
(= (+ FEMALE_AGE2 FEMALE_AGE3) 60)
```

```
;; end median calculation

;; average female age  = 33.5
(= (+ FEMALE_AGE1 FEMALE_AGE2 FEMALE_AGE3 FEMALE_AGE4) 134) ; 33.5 * 4
#endif

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 2B Male:  n=3, median=30, average=44
;; This is a generic pattern that will be repeated for any cell size of 3
;;

#ifdef USE_2B
;; there are three males
(= (+ (if (= S1 MALE) 1 0)
      (if (= S2 MALE) 1 0)
      (if (= S3 MALE) 1 0)
      (if (= S4 MALE) 1 0)
      (if (= S5 MALE) 1 0)
      (if (= S6 MALE) 1 0)
      (if (= S7 MALE) 1 0)
      )
   3)

;; constraints for median. There are only 3 men, so we know that Male #2 is 30
(int MALE_ID1 1 5)                     ; must leave room for 6 and 7 to be male
(int MALE_ID2 2 6)                     ; must leave room for 1 and 7 to be male
(int MALE_ID3 3 7)                     ; must leave room for 1 and 2 to be male

(< MALE_ID1 MALE_ID2)
(< MALE_ID2 MALE_ID3)

;; Pigeon hole principle. If the IDs are in order, we only need to assure
;; that each ID maps to a female. We add the final equal and a -1 to force
;; an error if there are not enough females.
(= MALE_ID1
   (if (= S1 MALE) 1
     (if (= S2 MALE) 2
       (if (= S3 MALE) 3
         (if (= S4 MALE) 4
           (if (= S5 MALE) 5
             -1))))))


(= MALE_ID2
   (if (and (= S2 MALE) (< MALE_ID1 2)) 2
     (if (and (= S3 MALE) (< MALE_ID1 3)) 3
       (if (and (= S4 MALE) (< MALE_ID1 4)) 4
         (if (and (= S5 MALE) (< MALE_ID1 5)) 5
           (if (and (= S6 MALE) (< MALE_ID1 6)) 6
             -1))))))

(= MALE_ID3
   (if (and (= S3 MALE) (< MALE_ID2 3)) 3
     (if (and (= S4 MALE) (< MALE_ID2 4)) 4
       (if (and (= S5 MALE) (< MALE_ID2 5)) 5
         (if (and (= S6 MALE) (< MALE_ID2 6)) 6
           (if (and (= S7 MALE) (< MALE_ID2 7)) 7
             -1))))))

;; Create temporary variables for the ages of these males
(int MALE_AGE1 MIN_AGE MAX_AGE)
(int MALE_AGE2 MIN_AGE MAX_AGE)
(int MALE_AGE3 MIN_AGE MAX_AGE)

;; make sure they are ordered
(< MALE_AGE1 MALE_AGE2)
(< MALE_AGE2 MALE_AGE3)

;; Fix the male ages to the person ages
(element MALE_ID1 (A1 A2 A3 A4 A5 A6 A7) MALE_AGE1)
(element MALE_ID2 (A1 A2 A3 A4 A5 A6 A7) MALE_AGE2)
(element MALE_ID3 (A1 A2 A3 A4 A5 A6 A7) MALE_AGE3)

;; The median is 30
(= MALE_AGE2 30)

; average male age: 44
(= (+ MALE_AGE1 MALE_AGE2 MALE_AGE3)    ; average male age = 44
   (* 3 44))

#endif

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 2C Black.  n=4  median=51  average=48.5
#ifdef USE_2C

;; 4 blacks
(= (+ (if (= R1 BLACK) 1 0)
      (if (= R2 BLACK) 1 0)
      (if (= R3 BLACK) 1 0)
      (if (= R4 BLACK) 1 0)
      (if (= R5 BLACK) 1 0)
      (if (= R6 BLACK) 1 0)
      (if (= R7 BLACK) 1 0)
      )
   4)

;;; Median age of blacks is 51
(int BLACK_ID1 1 4)                     ; must leave room for 5, 6 and 7 to be female
(int BLACK_ID2 2 5)                     ; must leave room for 1, 6 and 7 to be female
(int BLACK_ID3 3 6)                     ; must leave room for 1, 2 and 7 to be female
(int BLACK_ID4 4 7)                     ; must leave room for 1, 2 and 3 to be female

(< BLACK_ID1 BLACK_ID2)
(< BLACK_ID2 BLACK_ID3)
(< BLACK_ID3 BLACK_ID4)

;; Pigeon hole principle. If the IDs are in order, we only need to assure
;; that each ID maps to a female. We add the final equal and a -1 to force
;; an error if there are not enough females.

(= BLACK_ID1
   (if (= R1 BLACK) 1
     (if (= R2 BLACK) 2
       (if (= R3 BLACK) 3
         (if (= R4 BLACK) 4
           -1)))))

(= BLACK_ID2
   (if (and (= R2 BLACK) (< BLACK_ID1 2)) 2
     (if (and (= R3 BLACK) (< BLACK_ID1 3)) 3
       (if (and (= R4 BLACK) (< BLACK_ID1 4)) 4
         (if (and (= R5 BLACK) (< BLACK_ID1 5)) 5
           -1)))))

(= BLACK_ID3
   (if (and (= R3 BLACK) (< BLACK_ID2 3)) 3
     (if (and (= R4 BLACK) (< BLACK_ID2 4)) 4
       (if (and (= R5 BLACK) (< BLACK_ID2 5)) 5
         (if (and (= R6 BLACK) (< BLACK_ID2 6)) 6
           (if (and (= R7 BLACK) (< BLACK_ID2 7)) 7
             -1))))))

(= BLACK_ID4
   (if (and (= R4 BLACK) (< BLACK_ID3 4)) 4
     (if (and (= R5 BLACK) (< BLACK_ID3 5)) 5
       (if (and (= R6 BLACK) (< BLACK_ID3 6)) 6
         (if (and (= R7 BLACK) (< BLACK_ID3 7)) 7
           -1)))))

;; Create temporary variables for the black ages
(int BLACK_AGE1 MIN_AGE MAX_AGE)
(int BLACK_AGE2 MIN_AGE MAX_AGE)
(int BLACK_AGE3 MIN_AGE MAX_AGE)
(int BLACK_AGE4 MIN_AGE MAX_AGE)

(< BLACK_AGE1 BLACK_AGE2)
(< BLACK_AGE2 BLACK_AGE3)
(< BLACK_AGE3 BLACK_AGE4)

;; Fix the black ages to the person ages
(element BLACK_ID1 (A1 A2 A3 A4 A5 A6 A7) BLACK_AGE1)
(element BLACK_ID2 (A1 A2 A3 A4 A5 A6 A7) BLACK_AGE2)
(element BLACK_ID3 (A1 A2 A3 A4 A5 A6 A7) BLACK_AGE3)
(element BLACK_ID4 (A1 A2 A3 A4 A5 A6 A7) BLACK_AGE4)

;; The median black age is 51.
;; The average of BLACK_AGE2 and BLACK_AGE3 is 51, so their sum is 51*2
(= (+ BLACK_AGE2 BLACK_AGE3) (* 51 2))


; average black age = 48.5 (x 4 = 194)
(= (+ (if (= R1 BLACK) A1 0)
      (if (= R2 BLACK) A2 0)
      (if (= R3 BLACK) A3 0)
      (if (= R4 BLACK) A4 0)
      (if (= R5 BLACK) A5 0)
      (if (= R6 BLACK) A6 0)
      (if (= R7 BLACK) A7 0)
      )
   194)
#endif

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; 2D. White. n=3 median=24, mean=24  (white=1)
#ifdef USE_2D

;; n=3 whites
(= (+ (if (= R1 WHITE) 1 0)
      (if (= R2 WHITE) 1 0)
      (if (= R3 WHITE) 1 0)
      (if (= R4 WHITE) 1 0)
      (if (= R5 WHITE) 1 0)
      (if (= R6 WHITE) 1 0)
      (if (= R7 WHITE) 1 0)
      ) 3)


;; constraints for median. There are only 3 men, so we know that Male #2 is 30
(int WHITE_ID1 1 5)                     ; must leave room for 6 and 7 to be male
(int WHITE_ID2 2 6)                     ; must leave room for 1 and 7 to be male
(int WHITE_ID3 3 7)                     ; must leave room for 1 and 2 to be male

(< WHITE_ID1 WHITE_ID2)
(< WHITE_ID2 WHITE_ID3)

;; Pigeon hole principle. If the IDs are in order, we only need to assure
;; that each ID maps to a female. We add the final equal and a -1 to force
;; an error if there are not enough females.
(= WHITE_ID1
   (if (= R1 WHITE) 1
     (if (= R2 WHITE) 2
       (if (= R3 WHITE) 3
         (if (= R4 WHITE) 4
           (if (= R5 WHITE) 5
             -1))))))


(= WHITE_ID2
   (if (and (= R2 WHITE) (< WHITE_ID1 2)) 2
     (if (and (= R3 WHITE) (< WHITE_ID1 3)) 3
       (if (and (= R4 WHITE) (< WHITE_ID1 4)) 4
         (if (and (= R5 WHITE) (< WHITE_ID1 5)) 5
           (if (and (= R6 WHITE) (< WHITE_ID1 6)) 6
             -1))))))

(= WHITE_ID3
   (if (and (= R3 WHITE) (< WHITE_ID2 3)) 3
     (if (and (= R4 WHITE) (< WHITE_ID2 4)) 4
       (if (and (= R5 WHITE) (< WHITE_ID2 5)) 5
         (if (and (= R6 WHITE) (< WHITE_ID2 6)) 6
           (if (and (= R7 WHITE) (< WHITE_ID2 7)) 7
             -1))))))
```

```
;; Create temporary variables for the ages of these males
(int WHITE_AGE1 MIN_AGE MAX_AGE)
(int WHITE_AGE2 MIN_AGE MAX_AGE)
(int WHITE_AGE3 MIN_AGE MAX_AGE)

;; make sure they are ordered
(< WHITE_AGE1 WHITE_AGE2)
(< WHITE_AGE2 WHITE_AGE3)

;; Fix the male ages to the person ages
(element WHITE_ID1 (A1 A2 A3 A4 A5 A6 A7) WHITE_AGE1)
(element WHITE_ID2 (A1 A2 A3 A4 A5 A6 A7) WHITE_AGE2)
(element WHITE_ID3 (A1 A2 A3 A4 A5 A6 A7) WHITE_AGE3)

;; The median white is 24
(= WHITE_AGE2 24)

; average white age: 24
(= (+ WHITE_AGE1 WHITE_AGE2 WHITE_AGE3)    ; average white age = 24
   (* 3 24))
#endif

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 3B. Married Adults
;;  n=4, median=51 mean=54

;; Count queries

#ifdef USE_3B
(= (+ (if (= M1 MARRIED) 1 0)    ; average age of single adults =
      (if (= M2 MARRIED) 1 0)
      (if (= M3 MARRIED) 1 0)
      (if (= M4 MARRIED) 1 0)
      (if (= M5 MARRIED) 1 0)
      (if (= M6 MARRIED) 1 0)
      (if (= M7 MARRIED) 1 0)
      )
   4)

;; Median age of married adults is = 51
;; There are 4, so there are 2 less than 51 and 2 more than 51

(int MARRIED_ADULT_ID1 1 4)                     ; must leave room for 5, 6 and 7 to be marri
```

```
(int MARRIED_ADULT_ID2 2 5)                     ; must leave room for 1, 6 and 7 to be marri
(int MARRIED_ADULT_ID3 3 6)                     ; must leave room for 1, 2 and 7 to be marri
(int MARRIED_ADULT_ID4 4 7)                     ; must leave room for 1, 2 and 3 to be marri

(< MARRIED_ADULT_ID1 MARRIED_ADULT_ID2)
(< MARRIED_ADULT_ID2 MARRIED_ADULT_ID3)
(< MARRIED_ADULT_ID3 MARRIED_ADULT_ID4)

;; Pigeon hole principle. If the IDs are in order, we only need to assure
;; that each ID maps to a female. We add the final equal and a -1 to force
;; an error if there are not enough females.

(= MARRIED_ADULT_ID1
   (if (= M1 MARRIED) 1
     (if (= M2 MARRIED) 2
       (if (= M3 MARRIED) 3
         (if (= M4 MARRIED) 4
           -1)))))

(= MARRIED_ADULT_ID2
   (if (and (= M2 MARRIED) (< MARRIED_ADULT_ID1 2)) 2
     (if (and (= M3 MARRIED) (< MARRIED_ADULT_ID1 3)) 3
       (if (and (= M4 MARRIED) (< MARRIED_ADULT_ID1 4)) 4
         (if (and (= M5 MARRIED) (< MARRIED_ADULT_ID1 5)) 5
           -1)))))

(= MARRIED_ADULT_ID3
   (if (and (= M3 MARRIED) (< MARRIED_ADULT_ID2 3)) 3
     (if (and (= M4 MARRIED) (< MARRIED_ADULT_ID2 4)) 4
       (if (and (= M5 MARRIED) (< MARRIED_ADULT_ID2 5)) 5
         (if (and (= M6 MARRIED) (< MARRIED_ADULT_ID2 6)) 6
           (if (and (= M7 MARRIED) (< MARRIED_ADULT_ID2 7)) 7
             -1))))))

(= MARRIED_ADULT_ID4
   (if (and (= M4 MARRIED) (< MARRIED_ADULT_ID3 4)) 4
     (if (and (= M5 MARRIED) (< MARRIED_ADULT_ID3 5)) 5
       (if (and (= M6 MARRIED) (< MARRIED_ADULT_ID3 6)) 6
         (if (and (= M7 MARRIED) (< MARRIED_ADULT_ID3 7)) 7
           -1)))))

;; Create temporary variables for the ages of these females
;; This uses the Sugar
(int MARRIED_ADULT_AGE1 MIN_AGE MAX_AGE)
(int MARRIED_ADULT_AGE2 MIN_AGE MAX_AGE)
```

```
(int MARRIED_ADULT_AGE3 MIN_AGE MAX_AGE)
(int MARRIED_ADULT_AGE4 MIN_AGE MAX_AGE)

(< MARRIED_ADULT_AGE1 MARRIED_ADULT_AGE2)
(< MARRIED_ADULT_AGE2 MARRIED_ADULT_AGE3)
(< MARRIED_ADULT_AGE3 MARRIED_ADULT_AGE4)

;; Fix the female ages to the person ages
(element MARRIED_ADULT_ID1 (A1 A2 A3 A4 A5 A6 A7) MARRIED_ADULT_AGE1)
(element MARRIED_ADULT_ID2 (A1 A2 A3 A4 A5 A6 A7) MARRIED_ADULT_AGE2)
(element MARRIED_ADULT_ID3 (A1 A2 A3 A4 A5 A6 A7) MARRIED_ADULT_AGE3)
(element MARRIED_ADULT_ID4 (A1 A2 A3 A4 A5 A6 A7) MARRIED_ADULT_AGE4)

;; The median age of the married adults is 51
;; so the average of these is 51, so their sum is 51*2
(= (+ MARRIED_ADULT_AGE2 MARRIED_ADULT_AGE3) (* 51 2))

;; end median calculation

;; mean married adult age  = 54
(= (+ MARRIED_ADULT_AGE1 MARRIED_ADULT_AGE2 MARRIED_ADULT_AGE3 MARRIED_ADULT_AGE4)
   (* 4 54))

#endif




;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; 4A. Black Females
;; n=3, median=36, mean=36.7

#ifdef USE_4A
;;; Three black females (R=0, S=0)
;; there are three black females
(= (+ (if (and (= R1 BLACK) (= S1 FEMALE)) 1 0)
      (if (and (= R2 BLACK) (= S2 FEMALE)) 1 0)
      (if (and (= R3 BLACK) (= S3 FEMALE)) 1 0)
      (if (and (= R4 BLACK) (= S4 FEMALE)) 1 0)
      (if (and (= R5 BLACK) (= S5 FEMALE)) 1 0)
      (if (and (= R6 BLACK) (= S6 FEMALE)) 1 0)
      (if (and (= R7 BLACK) (= S7 FEMALE)) 1 0)
      ) 3)
```

```
;; constraints for median. There are only 3 men, so we know that Male #2 is 30
(int BLACK_FEMALE_ID1 1 5)                      ; must leave room for 6 and 7 to be male
(int BLACK_FEMALE_ID2 2 6)                      ; must leave room for 1 and 7 to be male
(int BLACK_FEMALE_ID3 3 7)                      ; must leave room for 1 and 2 to be male

(< BLACK_FEMALE_ID1 BLACK_FEMALE_ID2)
(< BLACK_FEMALE_ID2 BLACK_FEMALE_ID3)

;; Pigeon hole principle. If the IDs are in order, we only need to assure
;; that each ID maps to a female. We add the final equal and a -1 to force
;; an error if there are not enough females.
(= BLACK_FEMALE_ID1
   (if (and (= R1 BLACK) (= S1 FEMALE)) 1
     (if (and (= R2 BLACK) (= S2 FEMALE)) 2
       (if (and (= R3 BLACK) (= S3 FEMALE)) 3
         (if (and (= R4 BLACK) (= S4 FEMALE)) 4
           (if (and (= R5 BLACK) (= S5 FEMALE)) 5
             -1))))))


(= BLACK_FEMALE_ID2
   (if (and (= R2 BLACK) (= S2 FEMALE) (< BLACK_FEMALE_ID1 2)) 2
     (if (and (= R3 BLACK) (= S3 FEMALE) (< BLACK_FEMALE_ID1 3)) 3
       (if (and (= R4 BLACK) (= S4 FEMALE) (< BLACK_FEMALE_ID1 4)) 4
         (if (and (= R5 BLACK) (= S5 FEMALE) (< BLACK_FEMALE_ID1 5)) 5
           (if (and (= R6 BLACK) (= S6 FEMALE) (< BLACK_FEMALE_ID1 6)) 6
             -1))))))

(= BLACK_FEMALE_ID3
   (if (and (= R3 BLACK) (= S3 FEMALE) (< BLACK_FEMALE_ID2 3)) 3
     (if (and (= R4 BLACK) (= S4 FEMALE) (< BLACK_FEMALE_ID2 4)) 4
       (if (and (= R5 BLACK) (= S5 FEMALE) (< BLACK_FEMALE_ID2 5)) 5
         (if (and (= R6 BLACK) (= S6 FEMALE) (< BLACK_FEMALE_ID2 6)) 6
           (if (and (= R7 BLACK) (= S7 FEMALE) (< BLACK_FEMALE_ID2 7)) 7
             -1))))))

;; Create temporary variables for the ages of these males
(int BLACK_FEMALE_AGE1 MIN_AGE MAX_AGE)
(int BLACK_FEMALE_AGE2 MIN_AGE MAX_AGE)
(int BLACK_FEMALE_AGE3 MIN_AGE MAX_AGE)

;; make sure they are ordered
(< BLACK_FEMALE_AGE1 BLACK_FEMALE_AGE2)
(< BLACK_FEMALE_AGE2 BLACK_FEMALE_AGE3)
```

```
;; Fix the male ages to the person ages
(element BLACK_FEMALE_ID1 (A1 A2 A3 A4 A5 A6 A7) BLACK_FEMALE_AGE1)
(element BLACK_FEMALE_ID2 (A1 A2 A3 A4 A5 A6 A7) BLACK_FEMALE_AGE2)
(element BLACK_FEMALE_ID3 (A1 A2 A3 A4 A5 A6 A7) BLACK_FEMALE_AGE3)

;; The median is 36
(= BLACK_FEMALE_AGE2 36)

; media black female age: 36.7
(= (+ BLACK_FEMALE_AGE1 BLACK_FEMALE_AGE2 BLACK_FEMALE_AGE3)    ;
   110)
#endif

;; Statistic 1A. This is a sugar bug; we should not have to put it here.
;(= A4 30)

;; Solution. To verify constraints, uncomment these and everything should satisfy!
;; female=0        black=0   single=0
;; male  =1        white=1   married=1
;; Note that this is the sorted by age
#ifdef USE_SOLUTION
(= S1 0) (= A1  8) (= R1 0) (= M1 0)
(= S2 1) (= A2 18) (= R2 1) (= M2 0)
(= S3 0) (= A3 24) (= R3 1) (= M3 0)
(= S4 1) (= A4 30) (= R4 1) (= M4 1)
(= S5 0) (= A5 36) (= R5 0) (= M5 1)
(= S6 0) (= A6 66) (= R6 0) (= M6 1)
(= S7 1) (= A7 84) (= R7 0) (= M7 1)
#endif
```