Local nulls in summarised mobile and distributed databases

D. Chan & J. F. Roddick
School of Informatics and Engineering,
Flinders University of South Australia, Adelaide, Australia

Abstract

The concept and semantics of null values in relational databases has been discussed widely since the introduction of the relational data model in the late 1960s. With the introduction of highly mobile, distributed databases, in order to preserve the accepted soundness and completeness criteria, the semantics of the null value needs to expand to reflect a localised lack of information that may not be apparent for the global database. This paper discusses an extension to the notion of nulls to include the semantics of 'local' nulls. The paper introduces local nulls in terms of amendments to the relational algebra and examines its impact on query languages.

1 Previous research and motivation

Much of the research on the semantics of null values in relational databases dates back to the 1970s and 1980s [1–7]. The two definitions of nulls as given by Codd are missing and applicable, and missing and inapplicable [1] and Zaniolo [4] later proposed a third definition as, essentially, a lack of knowledge about the attribute's applicability, or *no information*.

To handle null values, various logical approaches have been developed. For example, the commonly-used three value logic includes true, false (often by virtue of a value's absence - q.v. the closed-world assumption [8]), and a maybe value which indicates that the results may be true [9, 10]. A four value logic has also been proposed which includes an additional truth value, which represents the outcome of evaluating expressions which have inapplicable values [11, 12]. Approaches to accommodating null values in practical systems include the work of Motro [13] who uses the ideas of conceptual closeness *fill the vacancies* represented by a null value and Roth *et al.* who aim to include nulls in NF^2 databases [5]. Null values have also been studied in relation to schema evolution and integration [14, 15].

WIT Transactions on Information and Communication Technologies, Vol 37, © 2006 WIT Press
www.witpress.com, ISSN 1743-3517 (on-line)
doi:10.2495/DATA060411

However, there has been little research undertaken in terms of missing data / nulls in distributed and mobile databases. Much of the reason for this is that there is little common agreement as to how to deal with such values. However, some of the techniques introduced to deal with null values in relational database are also applicable to distributed databases. That is, in many cases the evaluation of any query on a distributed database system as a whole is viewed as an evaluation of that query on a single relational database with all the data of the distributed one. Thus, in some cases, although not all, specialised techniques to handle nulls in distributed databases can be avoided. Moreover, there is research that introduces techniques to allow for the approximation of incomplete data using fuzzy rules [16]. It should be noted that many of the techniques for handling data (including nulls) in distributed databases are often applicable to mobile databases, especially they are viewed as extensions to distributed databases. In such cases, many designers are able avoid developing new techniques to dealing with problems in mobile databases specifically. Similarly, modifications to techniques developed for mobile systems are often applicable to distributed systems.

We adopt an ongoing motivating example of a medical practitioner visiting patients. The practitioner carries a PDA which stores a summarised version of the main database. We assume that the main database contains all information available, and the summarised database stores an optimum subset of (what has been estimated to be) relevant information. Note that 'relevant information' can be determined using the context-sensitive framework discussed in an earlier paper [17] or by some other means. Should the main database lack information these are then stored into the main database as null values. Thus, within the summary database, the same information, if selected for inclusion, would also be represented as null values. Since the summary database is created using relevancy, irrelevant data is excluded and, in the absence of local nulls, these missing data would also be represented by null values. In such cases, while the summarised database is disconnected from the clinic's network, it will not be possible to differentiate between data that are available somewhere in the network and data that are globally unavailable. Since the existence, or otherwise, of the data is known when the summary database is populated, by introducing the concept of a 'local null', a disconnected summary database would be able to distinguish between them.

There are a number of benefits to this. For example, in a mobile, wireless or large area distributed environment, where the cost of communication are relatively high, by knowing which data is available it is possible to request only specific data, and thus reducing the traffic over the low bandwidth communication network. That is, by examining the query, such as one that displays the patient's details, it is possible to create sub-queries that resolve the local nulls. These sub-queries may then be forwarded to the main database, thus allowing the practitioner to vary the accuracy of the results. That is, lower accuracy would result in faster response time.

2 Notation

Maier examined the presence of unknown values in relational databases [3]. In this section we adopt these conventions as a basis for our modifications to include the notion of local null values. We discuss these amended notations and provides a comparison to Maier's work. For the purposes of this paper, local and global null values will be represented as φ and ω, respectively.

A tuple in a local relation t is a *locally complete tuple* (or an LC-tuple) if it holds all data that also exists in the equivalent attributes in the global database. Conversely, a *locally partial* tuple (or an LP-tuple) contains one or more local nulls. LC-tuples will not contain any local nulls but may, of course, contain global nulls. LP-tuples may contain global ω and local φ nulls.

The designation of LC and LP-tuples is extended to include Maier's notation [3] where a tuple is designated as a *partial* tuple or a *total* tuple depending on whether global nulls exists. When an attribute value, $t(A)$, is not a local null it is considered identical to values in the global database, $t(A)^L \downarrow$, where $t(A) \downarrow$ defines an attribute value that is not a global null. Thus, for a set of attributes X, $t(X)^L \downarrow$ implies $t(A)^L \downarrow$ for every attribute $A \in X$. A simplified notation, $t^L \downarrow$, is then used to define t as a locally complete tuple, while a complete tuple consists of no nulls of either type. Similarly, for tuples, t and u defined over the same schema, t *locally subsumes* u, $t \geq^L u$ if $\forall\, u(A)^L \downarrow, u(A) = t(A)$. If $t \geq^L u$ and $t^L \downarrow$, then t is a *local extension* of u, $t \downarrow \geq^L u$. For example, the tuple $< a, \varphi, \varphi >$ is locally subsumed by $< a, b, \omega >$, and in this case, the tuple is also locally extended by $< a, b, \omega >$. In comparison to Maier's work, tuple $< a, b, \omega >$ is subsumed and extended by $< a, b, c >$.

A relation r is a *locally complete relation* (an LC-relation), $r \downarrow^L$, when all its tuples are locally complete tuples, and a *locally partial* relation (an LP-relation) when its tuples contains one or more local nulls. For a relation scheme R, $Rel \uparrow (R)^L$ is a set of all locally partial relations over R, while $Rel(R)^L$ is the set of all locally complete relations over R. For relations r and s over R, r *locally subsumes* s, denoted $r \geq^L s$, if $\forall t_s \in s, \exists t_r \in r : t_r \geq^L t_s$. If r is locally complete, then r is a *local extension* of s if every tuple of s is subsumed by at least one tuple of r, denoted $r \downarrow \succeq^L s$, and is a *local completion* of s if every tuple of s is subsumed by exactly one tuple of r, denoted $r \downarrow \geq^L s$. In Table 1, r is a local extension of s as both tuple $< d, e, f >$ and $< d, e, n >$ subsume $< d, e, \varphi >$ while p is a local completion of s.

Constraints on global nulls are usually such that they do not appear in any component of a candidate key [3]. Similarly, any component of a candidate key for any summary databases should not contain any local nulls. This is important as primary keys are used not only as object identifiers but as indexes to retrieve information when it is lacking from the summary database.

Table 1: Example of local extension and local completion. r is a local extension of s, while p is a local completion of s.

s(A B C)		
a	b	c
d	e	φ
k	l	m
g	φ	i

r(A B C)		
a	b	c
d	e	ω
d	e	f
k	l	m
g	h	i

p(A B C)		
a	b	c
d	e	ω
k	l	m
g	h	i

Table 2: Union compatible tables.

r(A B C)		
a	b	c
d	j	φ
φ	l	m
g	φ	i

r(A B C)		
a	φ	φ
φ	j	φ
g	h	φ

Table 3: Operations using null substitution principle.

$r \cup s$(A B C)		
a	b	c
d	j	φ
φ	l	m
g	φ	i

$r \cap s$(A B C)		
a	φ	φ
φ	j	φ
g	h	φ

$r - s$(A B C)		
φ	l	m

3 Local nulls and relational algebra

The relational algebra provides a set of operations used in the manipulation and retrieval of data from a database [1]. As with global nulls, the presence of local nulls requires an extension to the current relational algebra. Two global relations, $R(A_1, A_2, ..., A_n)$ and $S(B_1, B_2, ..., B_n)$, are considered union compatible if they have the same degree of n and if $\forall i \in n : dom(A_i) = dom(B_i)$. Given this, set operations can be applied to these relations. For locally partial relations where local nulls exist, we need to show that the use of set operators are plausible and that the results gracefully degrade as local nulls are resolved. Since local nulls are extensions of the traditional null concept, we can assume that set theory operations should be possible for locally partial relations. Union $\cup$, intersection $\cap$ and difference $-$ operate over two union compatible tables r and s:

$$\text{union:} r \cup s = \{t | t \in r \vee t \in s\} \quad (1)$$

$$\text{intersection:} r \cap s = \{t | t \in r \wedge t \in s\} \quad (2)$$

$$\text{set difference:} r - s = \{t | t \in r \wedge t \notin s\} \quad (3)$$

For these three operations, only duplicate tuples will be removed from the final relation. However, for relations with global nulls, Codd introduced the *null substitution principle* to reduce redundancy in the relation [10]. This was further discussed in [3, 4] for global nulls. For locally partial relations, this principle is similarly applicable to remove redundancy with respect to local nulls instead of global nulls. That is, if for two tuples t and u, $t \geq^L u$, then u will be removed from the relation (see Table 3 for examples).

Table 4: Keyed set operations.

r(A B C)	r(A B C)	$r \cup_A s$(**A** B C)	$r \cap_A s$(**A** B C)	$r -_A s$(**A** B C)
a b c	a φ φ	a b c	a b c	k l m
d j φ	d j φ	d j φ	d j φ	
k l m	g h φ	g h i	g h i	
g φ i		k l m		

Table 5: Value evaluation of $x \cap_k y$ and $x \cup_k y$.

		x			
		Value 'A'	Value 'B'	φ	ω
y	Value 'A'	A	Conflict	A	A
	Value 'B'	Conflict	B	B	B
	φ	A	B	φ	ω
	ω	A	B	ω	ω

In cases where the unique primary keys of a relation is available, it is possible to exploit the keys and introduce 'keyed' operations. Essentially, the keyed operations make it explicit when we can assume that two tuples represent the same object. For example, suppose there are two observations of a white car travelling at speed. One observer states that there were two passengers while the other is unsure. Given that white cars are common, we are unable to assume there was only the one car and thus merge the two observations. However, if the registration number of the cars were recorded we can make such an assumption and merge the two observations.

Before we define key operations, we first define $\in^L$ as an extension of $\in$ for global nulls. That is for a relation r and its corresponding LC-relation, $r \downarrow^L$, $t \in^L r$ iff there exists a tuple $t_1 \in r \downarrow^L$ such that $t_1 \geq^L t$. Now, for a set of n tuples, $\{t_1, t_2, ..., t_n\}$, we may represent a unique key set as $\{t_1, t_2, ..., t_n\}^k$ where $t_i(k) \neq t_j(k) \forall i, j$ in n and k is the set of primary key attributes. Thus for any two tuples in different union-compatible relations defined over the schema R with the same primary keys:

Conflicting tuples may result from such operations if unique keys are not identified in union compatible relations. That is, tuples conflict when, for a primary attribute A_p, the primary keys for two tuples are the same while some or all of the other non-null values are not (see Table 5). Since the tuple conflicts, the relations would then be considered non-union compatible and the keyed operation would fail. This union compatibility is different in that it is a data-centric compatibility instead of structural compatibility.

'Keyed' operations is seen as a generalised form of conventional set operation. That is, if r is an LC-relation, from observation we can see that $t \in^L r$ implies $t \in r$ and $t \notin^L r$ implies $t \notin r$. Similarly, if r consists of set n tuples, then $\{t_1, t_2, ..., t_n\}^k$ implies $\{t_1, t_2, ..., t_n\}$. Thus,

$$\begin{aligned} r \downarrow^L \cup_k s \downarrow^L &= \{t | t \in^L r \downarrow^L \vee t \in^L s \downarrow^L\}^k \\ &= \{t | t \in r \downarrow^L \vee t \in s \downarrow^L\} \\ &= \{t | t \in r \downarrow^L \vee t \in s \downarrow^L\} \end{aligned} \tag{4}$$

$$r \downarrow^L \cap_k s \downarrow^L = \{t|t \in^L r \downarrow^L \wedge t \in^L s \downarrow^L\}^k$$
$$= \{t|t \in^L r \downarrow^L \wedge t \in^L s \downarrow^L\}$$
$$= \{t|t \in r \downarrow^L \wedge t \in s \downarrow^L\} \quad (5)$$

$$r \downarrow^L -_k s \downarrow^L = \{t|t \in^L r \downarrow^L \wedge t \notin^L s \downarrow^L\}^k$$
$$= \{t|t \in^L r \downarrow^L \wedge t \notin^L s \downarrow^L\}$$
$$= \{t|t \in r \downarrow^L \wedge t \notin s \downarrow^L\} \quad (6)$$

3.1 Select and project operations

In general, the select operation selects a tuple or a group of tuples that satisfies a certain condition, as denoted by $\sigma_{<condition>}(R)$. With locally partial relations, the same form is used, i.e $\sigma_{<condition>}(R)$ where R is a locally partial relation.

A selection over Table 6 $\sigma_{B=b}(R)$ would result in rows 1 and 2 being selected, while rows 3-5 are not.

Examples

Table 6: Relation

R	A	B	C	D
	a	b	c	d
	g	b	e	φ
	f	φ	e	d
	k	h	φ	φ
	p	h	e	φ

Table 7: Projection.

(C	D)
c	d
e	φ
e	d
φ	φ
e	φ

Table 8: Multiple conditions.

(B	C)
b	e
φ	e

To include row 3 into the result, a new operator '?=' is introduced. The operator '?=' states that, for two attribute values A and B,

$$A\ ?= B \qquad \text{iff} \qquad (A = B) \vee (A = \varphi) \vee (B = \varphi) \quad (7)$$

Projection produces a new relation which includes only those attributes specified. Since this does not include an evaluation of local nulls against a value, it is possible to include local nulls that exist within those columns into the new relation. However, there is also a possibility that the new relation will now contain duplicate tuples. For example, a projection, $\pi_{(C,D)}$ over the relation in Table 6 will give the following relation, Table 7. This relation consists of a duplicate tuple <e, φ >. Since our evaluation of local null against another local null is that they can be the same, it is possible to delete the duplicating tuples. This is reasonable since including the extra duplicating tuples may not necessarily provide any additional information when compared to removing it. This is the same for the tuple, < φ, φ >, and is removed. Additionally, the evaluation may require multiple conditions which are joined together by NOT, AND and OR operators. For example $\pi_{(B,C)}\sigma_{(B?=b \wedge C?=e)}$, produces Table 8, in which the first row is true while the second row is locally unknown.

Table 9: Join example.

R(A B C D)			
a	b	c	d
g	b	e	φ
f	φ	e	d
k	h	φ	φ
p	h	e	φ

S(E F G H)			
l	a	d	k
m	a	h	φ
n	φ	e	d
o	ω	φ	φ
q	k	e	φ

Table 10: Conventional.

(A B C D E F G H)							
a	b	c	d	l	a	d	k
a	b	c	d	m	a	h	φ
k	h	φ	φ	q	k	e	φ

Table 11: Generalised.

(A B C D E F G H)							
a	b	c	d	l	a	d	k
a	b	c	d	m	a	h	φ
k	h	φ	φ	q	k	e	φ
a	b	c	d	o	ω	φ	φ
g	b	e	φ	o	ω	φ	φ
f	φ	e	d	o	ω	φ	φ
k	h	φ	φ	o	ω	φ	φ
p	h	e	φ	o	ω	φ	φ

Table 12: Local.

(A B C D E F G H)							
a	b	c	d	l	a	d	k
a	b	c	d	m	a	h	φ
g	b	e	φ	o	g	φ	φ
k	h	φ	φ	q	k	e	φ
a	b	c	d	n	φ	e	d
g	b	e	φ	n	φ	e	d
f	φ	e	d	n	φ	e	d
k	h	φ	φ	n	φ	e	d
p	h	e	φ	n	φ	e	d

3.2 Join operations

Unlike set theory operations where the relations being used must be compatible, these operations allow the joining of relations which can be defined over different schemata. Join operations usually involve the combination of two relations over compatible attributes. A conventional join of two locally partial relations will produce a new relation whose tuples are true with respect to the conditions of the join. Assuming a conventional join over the attributes A and F, (ie. $R \bowtie_{A=F} S$), the relation in Table 10 would be produced.

Methods to generalise joins over attributes where global nulls exists had been discussed in the literature [2, 3, 4, 18]. These generalised join states that there may also exist tuples where it is globally unknown whether it is true, in addition to tuples that are true. That is, using the null substitution principle suggested by Codd [18], it is possible to include tuples where the equivalence evaluation over attribute values that are globally null indicates a globally unknown solution. These tuples are included since they may suggest that the solution may be true. For example, Table 11 shows one of the proposed generalised join, denoted $\bowtie^G$, over the attributes A and F, ie.

$$R \bowtie^G_{A=F} S \tag{8}$$

For this example, tuples 1 to 5 are global unknown solutions. In fact, $R \bowtie^G_{A=F} S$ is considered equivalent to a conventional join where $R \bowtie_{(A=F)\cup(A=\omega)\cup(F=\omega)} S$. The local join over the attributes A and F would evaluate the true solutions in addition to locally unknown solutions. That is,

$$\begin{aligned} R \bowtie^L_{A=F} S &\equiv R \bowtie_{(A?=F)} S \\ &\equiv R \bowtie_{(A=F)\cup(A=\varphi)\cup(F=\varphi)} S \end{aligned} \tag{9}$$

This can be seen in Table 12. Similarly, a generalised local join is also proposed, denoted $\bowtie^{GL}$. That is,

$$R \bowtie_{(A=F)\cup(A=\omega)\cup(F=\omega)\cup(A=\varphi)\cup(F=\varphi)} S \tag{10}$$

Outer joins are also possible for tables with local nulls. There exists two outer join operations, the left outer join and the right outer join. Both operations perform similar procedures whereby it includes the tuples which would results from a normal join in addition to any remaining tuples from one table padded with global null values since they are unable to be joined with the other table. The difference between the left and the right outer joins is that the tuples from the first or left table in the join are padded in the former, while the tuples in second or right table in the join are padded in the latter. A local outer join, denoted $\bowtie^{LLO}$ and $\bowtie^{RLO}$ for left and right respectively, is then proposed to allow attributes to join over local nulls, but still be padded with global nulls, ie.

$$R \bowtie^{LLO} S \equiv R \bowtie_{(A=F)\cup(A=\varphi)} S \tag{11}$$

$$R \bowtie^{RLO} S \equiv R \bowtie_{(A=F)\cup(F=\varphi)} S \tag{12}$$

A summary of join operations, including both conventional and the proposed local operations, can be found in [19]. Space also precludes a discussion of the division operator but full details can be found in the same report.

4 Local nulls and SQL

To take advantage of the notion of locality using local nulls, extensions to the query language are required and we give some examples in SQL. For example, it is now possible to extend SQL such that we are now able to query for tuples that are locally unknown.

```
SELECT  *       FROM R
        WHERE   A = LNULL
```

Where the new reserved word `LNULL` represents local null values. It is now also possible to allow queries that results in tuples that are true and locally unknown. Ie., using a new operator $?=$, a new query statement such as

```
SELECT  *       FROM R
        WHERE   A ?= 'b'
```

can be constructed. For local joins,

$$\begin{aligned} R \bowtie^{L} S &\equiv R \bowtie_{(A?=B)} S \\ &\equiv \sigma_{(A?=B)}(R \times S) \end{aligned} \tag{13}$$

In which case, we may now convert it to

```
SELECT  *       FROM R,S
        WHERE   A ?= B
```

Additionally, it is also possible to convert the '?=' operator further.

$$\begin{aligned} R \bowtie^{L} S &\equiv R \bowtie_{(A=B)\cup(A=\varphi)\cup(B=\varphi)} S \\ &\equiv \sigma_{(A=B)\cup(A=\varphi)\cup(B=\varphi)}(R \times S) \end{aligned} \tag{14}$$

which in SQL is equivalent to

```
SELECT  *       FROM R,S
        WHERE   A = B OR A = LNULL OR B = LNULL
```

This now provides a way to separate local unknown solutions from true solutions. Similarly, for generalised local joins,

$$R \bowtie^{GL} S \equiv R \bowtie_{(A=B)\cup(A=\omega)\cup(B=\omega)\cup(A=\varphi)\cup(B=\varphi)} S$$
$$\equiv \sigma_{(A=B)\cup(A=\omega)\cup(B=\omega)\cup(A=\varphi)\cup(B=\varphi)}(R \times S)$$

which in SQL is equivalent to

```
SELECT  *       FROM R,S
        WHERE   A = B
        OR      A = ω   OR B = ω
        OR      A = φ   OR B = φ
```

In our example, assuming that the medical practitioner requires the address of a patient, the following query can be submitted:

```
SELECT  NAME,ADDRESS,AGE,TELEPHONE
        FROM    PATIENT
        WHERE   NAME ?= 'John Doe' AND AGE ?= 30
```

In our running example, the practitioner may be satisfied with the result. Moreover, since the 'Telephone' attribute of ID 1 returns a NULL value, no additional query is required from the main database. If the data for ID 2 is required, then a further query of the main database will be required. The subquery required to be transmitted to main database will be more specific reducing the time and communication bandwidth required. It is interesting to note that a local null appearing in the `SELECT` clause of an SQL statement is generally less of an issue that one appearing in a `WHERE` clause. For example, given the SQL example above, a local null in `AGE` would mean that the selection of the complete tuple is conditional, whereas a local null in `ADDRESS` would merely mean that the value is unknown but the presence of the tuple is unconditional.

5 Discussion

In this paper, we introduced the concept of local nulls in relations. We showed that it is possible to manipulate relations with these nulls through the use of the relational algebra with some modification. It is then possible to extend query languages to identify locally unknown solutions and, importantly, to provide users with a definable response to queries that include them. The concept of local nulls are primarily applicable within a distributed or mobile database system, where a particular piece of information may exist in multiple databases within that system. Because local nulls are considered to be temporary nulls that replace actual values within a database in order to conserve storage capacity, it is possible for attribute values to be null locally, while other databases may have its actual value. In mobile databases, where storage capacity is limited, it is useful to be able to store only parts of the database that are used often.

Acknowledgement

This research was partially supported by Australian Research Council SPIRT Grant C00107533, held in collaboration with HP Labs.

References

[1] Codd, E., A relational model of data for large shared data banks. *CACM*, **13**(6), pp. 377387, 1970.

[2] Lacroix, M. & Pirotte, A., Generalized joins. *SIGMOD Rec*, **8**(3), pp. 1415, 1976.

[3] Maier, D., The Theory of Relational Databases. Pitman Publishing, 1983.

[4] Zaniolo, C., Database relations with null values. *Journal of Computer and System Sciences*, **28**(1), pp. 142166, 1984.

[5] Roth, M.A., Korth, H.F. & Silberschatz, A., Null values in nested relational databases. *Acta Inf*, **26**(7), pp. 615642, 1989.

[6] Imielinski, T. & Lipski, W., Incomplete information in relational databases. *JACM*, **31**(4), pp. 761791, 1984.

[7] Vassiliou, Y., Null values in data base management a denotational semantics approach. *SIGMOD 79*, ACM Press: New York, NY, USA, pp. 162169, 1979.

[8] Reiter, R., On closed world databases. *Logic and Databases*, eds. H. Gallaire & J. Minker, Plenum Press: New York, pp. 5576, 1978.

[9] Yue, K.B., A more general model for handling missing information in relational databases using a 3-valued logic. *SIGMOD Rec*, **20**(3), pp. 4349, 1991.

[10] Codd, E., Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, **4**(4), pp. 397434, 1979.

[11] Codd, E., Missing information (applicable and inapplicable) in relational databases. *SIGMOD Rec*, **15**(4), 1986.

[12] Gessert, G.H., Four valued logic for relational database systems. *SIGMOD Rec*, **19**(1), pp. 2935, 1990.

[13] Motro, A., Vague: a user interface to relational databases that permits vague queries. *ACM Trans Inf Syst*, **6**(3), pp. 187214, 1988.

[14] Kim, W. & Seo, J., Classifying schematic and data heterogeneity in multidatabase systems. *Computer*, **24**(12), pp. 1218, 1991.

[15] Roddick, J.F., A survey of schema versioning issues for database systems. *Information and Software Technology*, **37**(7), pp. 383393, 1995.

[16] Chen, S.M. & Chen, H.H., Estimating null values in the distributed relational databases environment. *Cybernetics and Systems*, **31**(8), pp. 851871, 2000.

[17] Chan, D. & Roddick, J.F., Context-sensitive mobile database summarisation. *26th Aust. Computer Science Conf., Adelaide*, volume 16 of CRPIT, pp. 139149, 2003.

[18] Codd, E., Understanding relations (installment #7). *FDT-Bulletin of ACM Sigmod*, **7**(3), pp. 2328, 1975.

[19] Chan, D. & Roddick, J.F., Local nulls in mobile and distributed summary databases: Extended report. *Technical Report SIE-05-002*, Flinders University, 2005.