Null Semantics for Subqueries and Atomic Predicates

Narongrit Waraporn, and Kriengkrai Porkaew

***Abstract*— SQL is one of the major languages to manipulate and retrieve data in the databases. It was standardized by the corporation among researchers and commercial database industries into many versions. SQL provides NULL values for attributes which are unknown to the user. There are three meanings of NULL but they are not classified by database engines. Results of SQL queries show different meaning when they are produced by subqueries or atomic predicates. This paper explains the meaning of them with suggestions proposing to database community.**

***Index Terms*—Database Engines, Null, Predicates, SQL, Subquery**

I. Introduction

SQL was proposed in 1986 and revised into different versions; SQL-89, SQL-92, SQL:1999, SQL:2003, and recently SQL: 2006 which includes XML specifications. Null value, also, has a long history of its own problems attached to SQL standards and has been discussed since 1970s [16], [3], [5], [6] about handling the missing data. Many approaches to evaluate its miserable logical value have been proposed including multi-valued logic such as 3-valued logic in [18], and 4-valued logic in [9], [4] and others. Its effects are still recently discussed in various issues of database such as normalization in [13] providing axiomatization to normalize databases over the multivalued dependencies with null values, and language semantics in [10] between SQL and XQuery that treats an empty sequence in the same manner of NULL in SQL by returning the empty sequence if any of operators is an empty sequence.

Understanding the semantics of NULL values in various case is important because the side effects of mishandling null when retrieving them from a database to an application may lead to other problems such as, program crashing due to unusable values in [1] whose authors introduce a method called origin tracking to record program location where unusable values are assigned in a form of value piggybacking whereas other various techniques are demonstrated such as null dereference analysis in [7] and reference-counting garbage collection in [11].

Manuscript received July 22, 2008.

N. Waraporn is with the School of Information Technology, King Mongkut's University of Technology Thonburi, Thungkru, Bangkok, 10140 Thailand (phone: 662-470-9909; fax: 662-872-7145; e-mail: narongrit@sit.kmutt.ac.th).

K. Porkaew is with the School of Information Technology, King Mongkut's University of Technology Thonburi, Thungkru, Bangkok, 10140 Thailand (e-mail: kk@sit.kmutt.ac.th).

The problem origination of an application receiving NULL values from a database is from SQL that can be tackled with different strategies. [15] suggests an adopting of Modified Condition Decision Coverage (MCDC) for SQL conditions to switch Boolean logic to a three-valued logic whereas [2] proposes an improvement of SQL query optimization in their version of nested relational algebra that allows null values and duplicate values.

During the time we were ETL our data from a relational database to our data warehouse, we found that writing SQL statement with subqueries may cause various problems when encountering NULL values whereas other researchers found different problems such as negations cause a switch from a sure answer to a potential answer and vice versa [12]. Many execution strategies for SQL subqueries are discussed in [8] using a mapping technique to deal with quantified comparison in the presence of NULL values.

In this paper, we present semantics of NULL values in a nested SQL query. The fundamental meanings and logical facts of NULL values are explained in section II. In section III, we illustrate how the comparison between NULL and atomic values is done. The meanings of NULL values in different cases according to the locality of NULL are demonstrated in section IV and V while section VI presents other cases such as semantic of NULL values in the aggregate functions and their equality comparisons. Section VII suggests practical approaches when dealing with NULL for designing of database and application. Section VII, also, suggests an extension for database engines to cope with 3-value logic of NULL values. At last, we conclude and discuss other issues of NULL values in section VIII.

II. Semantics of NULL in Database

We can understand NULL values in two aspects; their meanings, and their logical facts. The meanings are classified by considering the acquiring of data into the table while the logical facts are defined from the three-valued logic.

A. Meanings of NULL Values

NULL values in DBMS are classified in three meanings:

Missing: data supposes to be in the column but has not yet filled in. For example, NULL for date of birth of an employee is missing because every person was born on a date. Other examples of missing for NULL values are name, and gender.

Not-applicable: data cannot be applied to the column at this time. For example, GPA for freshmen, who just enter the college, is not valid during the first semester. The NULL value for GPA column is not applicable for at least one semester.

Unknown: data can be either missing or not-applicable values for NULL. For example, most students will have GPA after the final exam. However, a NULL value for the GPA of a sophomore may be considered in two situations. It is missing data because of data loss during processing at the registrar office. The second case is when student dropped all course before the deadline, therefore the student still has no GPA after the first year. The second case represents the not-applicable meaning. Therefore the NULL value for GPA of students who passed the first semester could be unknown. Other examples of unknown meaning for NULL values are supervisor column which can be missing or not-applicable if he/she is the CEO, and address column which can be missing or not-applicable for a person who may be temporarily homeless after hurricane season.

In this paper, the case that a customer does not provide date of birth, or salary due to privacy information, can be considered in another meaning but it is out of scope for this paper.

B. Logical Facts of NULL values

Three valued logic, 3VL, for DBMS is defined by three facts; true, false, and unknown, which is NULL, values. [17] defined truth tables of 3VL in nine out of eighty one possible pairs of conjunction and disjunction tables. Out of nine pairs, SQL99 uses two pairs to define the elementary truth tables shown in the Tables I and II by applying 3VL for NULL values. Table I is the truth table for the logical AND while Table II is for the logical OR. SQL99 demonstrated in [14] also applied 3VL for the logical negation and is defined in the truth table shown in Table III.

Table I. Truth Table for Conjunction

AND	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

Table II. Truth Table for Disjunction

OR	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

Table III. Truth Table for Negation

P	Not P
True	False
False	True
Unknown	Unknown

Due to the fact of logical AND that its predicate is true if both facts are known as true whereas the third meaning of NULL value is unknown, the value of truth table of logical AND is unknown if another fact is unknown. However, if at least one of the facts is false, regardless of another, the predicate of logical AND returns false. This is analogous to the fact of logical OR in two valued logic, if at least one of the facts is true, regardless whether we know the second fact or not, the OR predicate is true. On the other hand, if one of the facts is false while another is unknown, we cannot determine the OR predicate.

Because of 3VL in SQL where the facts of truth table could be unknown, the results of condition in WHERE clause may mislead the programmer when writing a SQL statement against NULL values.

III. PREDICATES AGAINST NULL

When comparing values in an expression against NULL, SQL standard suggests the use of IS NULL predicate. The syntax is "value-expression IS NULL". An IS NULL predicate tests whether the value-expression is an empty value or not. The surprise of IS NULL predicate occurs when the value-expression is a tuple containing more than one column such as WHERE (SALARY, SUPERVISOR) IS NULL.

According to SQL99 standard in [14], the NULL predicate semantics is summarized in Table IV. The surprise of two-value logic for IS NULL predicate is when n-value expression containing some null values and some not-null values in an expression is false if the predicate compares with IS NULL while comparing with IS NOT NULL, the n-value expression is also false. This is due to the fact of IS NULL predicates that it is true if and only if every value in the expression is null. Similarly, the IS NOT NULL predicate is true if and only if every value in the expression must be not null.

Table IV. NULL Predicate Semantics

P: n-value expression	P IS NULL	P IS NOT NULL	NOT P IS NULL	NOT P IS NOT NULL
n=1: null	True	False	False	True
n=1: not null	False	True	True	False
n>1: all null	True	False	False	True
n>1: some null	False	**False**	True	True
n>1: none null	False	True	True	False

DB2, and MySQL do not allow the value-expression as a n-value tuple to compare with IS NULL. They return syntax error when comparing such tuple with IS NULL predicate. However, they allow the comparison of the value-expression as an n-value tuple with an n-value tuple, but not the case of MS-Access. One reason that commercial databases does not allow this type of predicate may due to the complicated semantics of NULL when using IS NULL or IS NOT NULL predicates as shown in Table IV. Alternatively, programmer can rewrite the comparison of an n-value expression with IS NULL predicate into n predicates comprehending to the intended semantics of the query such as "SALARY IS NULL AND SUPERVISOR IS NULL", or "SALARY IS NULL OR SUPERVISOR IS NOT NULL".

Since NULL means unknown in 3VL, the basic comparisons such as =, < >, <, <=, >, and >= return unknown value to the predicate containing NULL value. An example of truth tables for equality is shown in Table V.

For example, if we want to find two employees who are supervised by the same supervisor, Mike and Ann in Table

VII should not be in the result because we don't know who their supervisors are. Therefore, NULL = NULL comparison should be unknown.

Table V. Truth Table for Equality

=	X	~X	Unknown
X	True	False	Unknown
~X	False	True	Unknown
Unknown	Unknown	Unknown	Unknown

However, the fact in Table V might not be perfect. Using data in Table VII, if the condition is "SUPERVISOR = SUPERVISOR", every database engine does not include rows 1 and 5 due to NULL = NULL comparison is unknown. But shouldn't everyone have the same supervisor to him/herself?

If we intend the comparison of NULL = NULL to be true as in the previous paragraph, we may add an additional predicate such as "SUPERVISOR = SUPERVISOR OR SUPERVISOR IS NULL". So, the case of NULL = NULL comparison will be excluded. A similar example is also presented in section VIII.

Another contradiction of NULL comparison of 3VL in SQL is the case of aggregate functions in GROUP BY clause. Let say, if we would like to count the number of employee for each supervisor such as "Select count(*) from employee group by supervisor" from the data in Table VII, every database engine considers that Mike and Ann are in the same group and returns 2 as the number of employees whose supervisor is unknown. If NULL = NULL comparison is unknown, database engine should not consider them into the same group when applying an aggregate function.

In the case of aggregate functions, if we can define NULL values into either missing or not-applicable, then SQL can group the data properly. In the case of missing data for the supervisor column, SQL should not group them together because these missing supervisors might or might not be the same person. If the case of NULL in supervisor column means not-applicable, then SQL should group them together into the same group because the count of the number of employees for each supervisor is unambiguous comparing to the meaning of missing data. This supports the proposal of four-valued logic for databases in [4].

IV. NULL INTERPRETATION IN SUBQUERIES

We create tables and run queries in four database engines; DB2 Express-C 9.0, MS-Access 2007, MySQL 5.0.18, and Oracle 10g. The department and employee data in Tables VI and VII will be used throughout the paper. WORKDEP column in the EMPLOYEE table has a foreign key referencing to DEPNO column in the DEPARTMENT table.

To compare with a NULL value, SQL offers IS comparison operator in WHERE clause. For example:

Query I

```
SELECT ENAME FROM EMPLOYEE
WHERE SALARY IS NULL
```

A record containing NULL for salary will be shown in the result of query 1. As for this case, Ann is shown in every database engine.

To find a value that is not in a relation, SQL offers a NOT IN atomic/set comparison operator to compare with a subquery. A record whose searching column is not in the subquery is retrieved in the result. For example:

Query II

```
SELECT ENAME FROM EMPLOYEE
WHERE EMPID NOT IN (SELECT MANAGER
      FROM DEPARTMENT)
```

Employee who is not a manager of any department will be listed which is Tom and Ann.

However, a similar query using NOT IN operator to compare with NULL values, does not return results as expected. We illustrate the comparison between NULL values and subquery into three cases; comparing a not-null value with a subquery containing NULL values, a NULL value with a subquery without any NULL, and a NULL value with a subquery containing NULL values.

A. Subquery Containing NULL Values

When a pointer points to a record in the main query, values in the record are passed to the WHERE condition for comparison. For this case, values of the column comparing to the subquery, in every record are not NULL, but, in contrast, they are used to compare with a subquery returning NULL values.

Table VI. DEPARTMENT Table

Depno	Dname	Location	Manager
1	IT	New York	111
2	HR	London	112
3	Sale	New York	113

Table VII. EMPLOYEE Table with Foreign Key on WORKDEP to DEPARTMENT Table

EmpID	Ename	Salary	Supervisor	workdep
111	Mike	50000	NULL	1
112	John	250000	111	2
113	Jake	120000	112	2
114	Tom	40000	115	NULL
115	Ann	NULL	NULL	1

For example, query III finds a department that has no employee working at:

Query III

```
SELECT DEPNO FROM DEPARTMENT
WHERE DEPNO NOT IN (SELECT WORKDEP
                    FROM EMPLOYEE)
```

Every testing database engines returns no records whereas DEPNO 3 is not in the WORKDEP column of DEPARTMENT table. However, the query II returns 114, and 115 which is not in the MANAGER column of EMPLOYEE table. This is due to the NULL values in the WORKDEP column. The database engines treat NULL values as missing values. It cannot conclude whether DEPNO 3 is not in the WORKDEP column because the NULL can be 3 or other values.

Considering it as a set, the WHERE condition of query II compares

111 ∉ { 111, 112, 113 } which is false.

115 ∉ { 111, 112, 113 } which is true.

whereas the WHERE condition of query III compares

3 ∉ {1, x, 2, 1, y} which cannot be true or false because x and y are variables whose values are undetermined as NULL.

Conversely, if an employee 114 is a new employee and has no department assigned to, the meaning of NULL in WORKDEP column for employee 114 is not-applicable. For this case, department number 3 has no one working at. Therefore the database engines do not treat NULL for the NOT IN operation as not-applicable because the query III does not return DEPNO 3. However, SQL offer EXISTS quantifier for an alternative for IN. Another way to find a department that has no employee working at can be written in SQL as query IV.

Query IV

```
SELECT DEPNO FROM DEPARTMENT
WHERE NOT EXISTS   ( SELECT *
                     FROM EMPLOYEE
                   WHERE DEPNO = WORKDEP)
```

We ran query IV in our database engines. Every database engine returns DEPNO 3. The reason is that subquery tests the WHERE condition that receives 3 from the main query and the equality of 3 to NULL is false. This includes other cases: 3=1 is false and 3=2 is false. The WHERE condition of query IV can be considered as

~ (∃ x / x = 1 or x = 2 or x = NULL)

As a result, the subquery returns empty relation when 3 is passed from the main query. Thus, NOT EXISTS quantifier is true and DEPNO 3 is shown in the result of query.

Therefore, the meaning of NULL for the NOT EXISTS quantifier is suitable for the case of not-applicable, but not for missing values.

To find only matched value excluding the NULL values, there are many ways to write such query. Using IN, opposite to NOT IN, is one of the options. Other options include Cartesian product operator, and EXISTS quantifier. Query V, VI and VII, demonstrate these options.

Query V

```
SELECT DEPNO FROM department
WHERE DEPNO IN (SELECT WORKDEP
                FROM EMPLOYEE)
```

Query VI

```
SELECT DEPNO
FROM DEPARTMENT, EMPLOYEE
WHERE DEPNO = WORKDEP
```

Query VII

```
SELECT DEPNO FROM DEPARTMENT
WHERE EXISTS (SELECT * FROM EMPLOYEE
              WHERE DEPNO = WORKDEP)
```

IN operator and EXISTS quantifier in Query V and VII shows records of the main query that matches in the subquery whereas query VI uses a Cartesian product to join two tables. Query V and VII show the same result. They include DEPNO 1, and 2 whereas query VI shows a slightly different answer which includes DEPNO 1, 2, and 1 due to the characteristic of Cartesian product to concatenate a record to all matched records.

We can write WHERE condition of query V, where x and y represent NULL, in set comparisons as:

1 ∈ {1, x, 2, 1, y} which is true.

2 ∈ {1, x, 2, 1, y} which is true.

and 3 ∈ {1, x, 2, 1, y} which is false when we executed in each testing database.

The example reveals that NULL values for this case does not mean missing because if it is missing, the database engines cannot determine whether 3 is a member or not. In contrast, the database engines determine NULL values as not-applicable meaning that employee 114 is not working at DEPNO 3. Thus, 3 is not a member of subquery and is false for the IN comparison operator.

This raises an issue that IN operation interprets NULL values into two meanings; missing for NOT IN and not-applicable for IN.

Query VII shows the same result as of query V. We can write WHERE condition of query VII as

(. ∃ x / x = 1 or x = 2 or x = NULL) where x is a value passed from the main query.

When x is 3, the condition returns false because 3 is not equal to 1, 2 or NULL. Thus, 3 is not a result of the main query. Therefore NULL for EXISTS quantifier has the meaning of not-applicable, but not missing. The reason is that if NULL value is a missing value, 3 may or may not equal to a missing value. It would be nondeterministic. If NULL value means not-applicable, 3 will not equal to something that has no value. This is also true in our example that DEPNO 3 does not exist in the subquery.

These results show us that IN and NOT IN operators interpret the meaning of NULL value differently but not the case of EXISTS and NOT EXISTS quantifier.

The next two queries, VIII and IX, demonstrate the equivalency of the meaning of NULL values between set comparison operators when NULL values are compared:

a) IN versus " =SOME" and

b) NOT IN versus "!=ALL"

Query VIII

```
SELECT DEPNO FROM DEPARTMENT
WHERE DEPNO =SOME (SELECT WORKDEP
                   FROM EMPLOYEE)
```

Result of query VIII is the same as of query V. We can write the WHERE condition of query VIII as

(. ∃y/y∈WORKDEP and x = y) where x is DEPNO

For our example, it compares

(x = 1 or x = 2 or x = NULL)

When DEPNO is 1 or 2, the subquery is true, but not for 3. This shows us that =SOME operator uses the meaning of not-applicable which is the same as of IN operator.

Query IX

```
SELECT DEPNO FROM DEPARTMENT
WHERE DEPNO !=ALL (SELECT WORKDEP
                   FROM EMPLOYEE)
```

(In MS-Access 2007 not-equal operator is < >.)

Result of query IX is empty which is the same as of NOT IN operator in query III. The WHERE condition of query IX can be written as

(∀y/ y∈WORKDEP and x!=y) where x is DEPNO

For our example, it compares

(x ! = 1 and x ! = 2 and x != NULL)

When x is 1, the condition is false. It is the same to 2. However when x is 3, 3 != 1 is true, 3 != 2 is true, and 3 != NULL is false. (When comparing equality between NULL and a constant, it is always false.) Therefore, when DEPNO is 3, the WHERE condition returns false. DEPNO 3 is not an answer for the query IX. This is consistency with NOT IN operator in query III. Hence, the meaning of NULL values applied with != operator is missing.

B. NULL Comparing with Subquery without NULL

This case occurs when a value of the record in the main query is a NULL value and it is used to compare with a subquery containing no NULL value. We consider only the cases expressing the different meanings of NULL which are NOT IN, and IN operators and the EXISTS quantifier.

Query X

```
SELECT * FROM EMPLOYEE
WHERE WORKDEP NOT IN (SELECT DEPNO
                      FROM DEPARTMENT)
```

Query X returns no record, not even Tom who has no value of DEPNO. NULL value in this case means missing because it implied that Tom is working in a department either DEPNO 1, 2, or 3 but we do not have his information. If we have an employee who works in department 4 but not in the existing department, he/she would be listed in the result. This is not the case when we have a foreign key referencing from EMPLOYEE to DEPARTMENT tables. If the meaning of NULL in the DEPNO for Tom means not-applicable, Tom would not have a department to work at. So, he should be listed, but it is not the case for the result of query X.

The database engines interpret NOT IN operator of query X as

$x \notin \{ 1, 2, 3 \}$ where x is an unknown variable. The database engines return false because of NULL comparison. This shows that it is not true that x is not a member of set {1, 2, 3}. Logically, x is a member of the set. Therefore NULL value comparing with a relation by NOT IN operator means a missing value.

Query XI

```
SELECT * FROM EMPLOYEE
WHERE WORKDEP IN (SELECT DEPNO
                  FROM DEPARTMENT)
```

Opposite to NOT IN, IN operator in query XI returns Mike, John, Jake, and Ann but not Tom. This is obvious that four of them are working at a department saving in the DEPNO column of DEPARTMENT table but Tom is not. In this case, "Tom is working in department 1, 2, or 3" is false so, the result does not include Tom.

$x \in \{ 1, 2, 3 \}$ is false where x is an unknown variable.

It implies that x cannot be 1, 2 or 3. Therefore, the meaning of NULL value for IN operator in this case is not-applicable. This cannot be missing because x cannot be 1, 2, or 3, otherwise $x \in \{1, 2, 3\}$ would not be false, and x cannot be 4 due to foreign key between EMPLOYEE and DEPARTMENT tables.

Query XII

```
SELECT * FROM EMPLOYEE
WHERE NOT EXISTS (SELECT *
                  FROM DEPARTMENT
                  WHERE DEPNO = WORKDEP)
```

Unlike query X, query XII returns Tom. Query XII finds an employee who is not working in any department. If NULL of WORKDEP means missing, Tom would work at either DEPNO 1, 2, or 3 and he would not be listed in the result. Therefore, the meaning of NOT EXISTS quantifier of query XII is not-applicable which is consistent with the meaning of NULL value in query IV. The WHERE condition can be rewritten as

~ (∃x / x = 1 or x = 2 or x = 3) where x = {1, 2, NULL}

When x is 1 or 2, the proposition is false. When x is NULL, NULL=1 is false, NULL=2 is false, and NULL=3 is false. The (. ∃x / x = 1 or x = 2 or x = 3) is false. So the proposition is true. Therefore, Tom is a result of query XII.

C. NULL Comparing with Subquery Containing NULL

Queries XIII and XIV are the combination of the previous cases; subquery returns NULL values and the main query sends NULL values to compare with the subquery. Suppose the DEPNO 2 has no manager as shown in Table VIII.

Query XIII

```
SELECT SUPERVISOR FROM EMPLOYEE
WHERE SUPERVISOR NOT IN
        (SELECT MANAGER
        FROM DEPARTMENT)
```

Query XIV

```
SELECT SUPERVISOR FROM EMPLOYEE
WHERE NOT EXISTS (SELECT *
      FROM DEPARTMENT
      WHERE SUPERVISOR = MANAGER)
```

Both queries simply mean "find supervisor who is not a manager". When we ran the queries, query XIII returned no record, but query XIV returned 112, 115 and two NULL values.

In query XIII, if a NULL value of MANAGER column means not-applicable, the 112 and 115 must be listed. Similar to NULL values in SUPERVISOR column, their meanings are missing. Employees 111, and 115 have a supervisor but their supervisors are managers, so their supervisors are not listed in the same way as of employees 112, 113, and 114. If employees 111, and 115 do not have a supervisor (not-applicable), then it is true that their supervisors are not in the list of managers. So, their records should be listed, but they are not. Therefore the meaning of NULL values for SUPERVISOR in query XIII is, only, missing.

Table VIII. DEPARTMENT Table with a NULL Value in the MANAGER Column

Depno	Dname	Location	Manager
1	IT	New York	111
2	HR	London	NULL
3	Sale	New York	113

Because query XIV showed 112, and 115, the meaning of NULL values in MANAGER column could be not-applicable. However it can be missing if its value is 116. This is because of foreign key. The effects of foreign key are explained in section V. This is also applied to SUPERVISOR column.

In case of IN operators, and EXISTS quantifier, the meaning of NULL values in MANAGER and SUPERVISOR columns are not-applicable with the same reasons as explain for query XIV.

V. UNKNOWN MEANING IN SUBQUERIES

In this section, we demonstrate the case of NULL values when two tables have no reference integrity. The example is shown in Table IX, when the WORKDEP column of EMPLOYEE table does not have a foreign key to the DEPARTMENT table. Some employees may work in the department that is not in the DEPARTMENT table such as John.

Query III returns the same result for both having-foreign key in section IV and no-foreign key in this section. Therefore the meaning of NULL value in the case of no-foreign key is missing as the case of having-foreign key. The value of missing NULL can be 1, 2, or 3 but not 4 or others because if its value is 4 or other, the query III would show DEPNO 3. The meaning of NULL cannot be not-applicable with the same reason as the case of having foreign key.

Query X returns, by all database engines, only John, who is working in DEPNO 4, but not Tom who, has no department to work at this time. It is slightly different from the same case of having-foreign key which has no result return. NULL value in this case is also interpreted as missing. It implied that Tom is working in a department either DEPNO 1, 2, or 3 but not 4, and we do not have his info. Unlike John who is working in DEPNO 4, John is listed in the result. The meaning cannot be not-applicable with the same reason of query X in section IV.

Table IX. EMPLOYEE Table without Foreign Key on WORKDEP Column to the DEPARTMENT Table

EmpID	Ename	Salary	Supervisor	workdep
111	Mike	50000	NULL	1
112	John	250000	111	**4**
113	Jake	120000	112	2
114	Tom	40000	115	NULL
115	Ann	NULL	NULL	1

In case of having-foreign key, the query X is meaningless because the query always returns no value. In case of no-foreign key, it means "finding employee who works outside the company (not in the list of departments of the company)". However, for both cases, the meaning of NULL value is missing.

The database engines interpret NOT IN operator of query X as

$4 \notin \{ 1, 2, 3 \}$ which is true.

$x \notin \{ 1, 2, 3 \}$ where x is an unknown variable. The database engines return false because of NULL comparison. This shows that it is not true that x is not a member of set {1, 2, 3}. So, x must be a member of the set which also means that x cannot be 4 as of John.

When we ran query XII with employee that does not have foreign key, both Tom and John showed in the result. NULL value in this case can be interpreted either missing or not-applicable. In case of missing, Tom may work at DEPNO 4, 5, or others but not 1, 2, and 3. If he works at DEPNO 1, 2, or 3, he will not be listed, but he is listed. This is similar to the case of John. If the meaning of NULL in the DEPNO for Tom is not-applicable, Tom would not have a department to work at. Thus, he would be listed in the result and he is listed.

In case of having-foreign key, the query X means "finding employee who does not have a department to work at". In case of no-foreign key, it means "finding employee who does not have a department to work at or works outside the company (not in the list of department of company)". The first case is for not-applicable null whereas the second case is for both not-applicable and missing null. Therefore, it is unknown.

Query XV

```
SELECT * FROM EMPLOYEE
WHERE EXISTS (SELECT *FROM DEPARTMENT
      WHERE DEPNO = WORKDEP)
```

Query XV does not return John and Tom. As for meaning, Tom may not work in any department, as not applicable, or Tom may work in the department 4 with John (or other departments but not 1, 2, or 3) but we do not have his information about working department. Therefore, the meaning of NULL value in the EXISTS case is unknown which can be either not-applicable or missing. The meaning of NULL values as unknown are also applied to the queries IV, and VII for quantifier NOT EXISTS and EXISTS, respectively, and queries V, and XI for set comparison operator IN.

VI. OTHER ATOMIC PREDICATES COMPARING NULL

There are other cases when interpretation of NULL values is unclear in an atomic predicate. One of them is the case of DISTINCT keyword of SQL and another case is the equality between NULL and subquery.

A. DISTINCT and Aggregate Functions

Let consider the following queries with data in Table VII:

Query XVI

```
SELECT SUPERVISOR FROM EMPLOYEE
```

Query XVII

```
SELECT COUNT(*) FROM EMPLOYEE
```

Query XVIII

```
SELECT DISTINCT(SUPERVISOR)
  FROM EMPLOYEE
```

Query XIX

```
SELECT COUNT(DISTINCT(SUPERVISOR)
  FROM EMPLOYEE
```

Query XX

```
SELECT COUNT(SUPERVISOR)
  FROM EMPLOYEE
```

Query XVI returns five records which is consistent with query XVII that returns 5. In contrast, query XVIII returns four records by evaluating that two NULL values are not distinct while query XIX returns 3 by excluding the NULL values from the counting. However, query XIX is consistent with query XX that returns 3.

One reason that SQL holds the duplicate, which is opposite to the set theory, is the correctness of the computation of the aggregate functions. For example, if two employees have the same salary and SQL eliminates duplicates before computation, the summation of two identical salaries would not be twice of the salary. Count(*) in query XVII counts the number of rows which is equal to the number of rows of query XVI due to the duplicate holding of the SQL.

However, SQL does not include NULL values when applying them into aggregate functions, obviously with SUM, AVG, MAX, and MIN functions, because NULL is not zero. COUNT is also considered as an aggregate function even though it does not require computation among values. This leads the semantics of query XVIII different from the query XIX that user sees four values displayed but the number three when they are counted.

NULL for the COUNT function in query XX means not-applicable because if the NULL value means missing, the number of supervisors cannot be determined that there are exactly three supervisors. Every tested database engines confirms that there are exactly three supervisors. The NULL values in this column should not represent any value. Another word, these employees have no supervisor.

B. Equality between NULL and Subquery

In SQL, there is an exceptional case in WHERE clause, opposing to the set theory, that a programmer can write equality comparing an expression with a relation. It is only allowable if the relation returns only one tuple such as a subquery in query XXI.

Query XXI

```
SELECT * FROM EMPLOYEE
WHERE SALARY = ( SELECT MIN(SALARY)
                 FROM EMPLOYEE )
```

WHERE clause of Query XXI is valid for SQL but not for the set theory because, in set theory, 40000 = {40000} is not comparable. Since it is valid in SQL, the second unclear case of NULL value is that whether a NULL value is equal to an empty set and a NULL value is equal to a set containing NULL values. We consider the comparison between NULL and a set of NULL value(s) in three cases.

a) NULL = { }
b) NULL = { NULL }
c) NULL = { NULL, NULL }

Cases a) and b) are valid but case c) is error because the relation returns more than one value which is not the exceptional case in SQL. Let consider the following query

Query XXII

```
SELECT * FROM EMPLOYEE
WHERE SUPERVISOR = (SELECT SUPERVISOR
                    FROM EMPLOYEE
                    WHERE ENAME = 'BOB')
```

Query XXIII

```
SELECT * FROM EMPLOYEE
WHERE SUPERVISOR = (SELECT
      DISTINCT( SUPERVISOR) FROM EMPLOYEE
          WHERE ENAME = 'MIKE')
```

According to data in Table VII, Bob is not an employee. So, Query XXII is an example of case a). In contrast, Mike is an employee whose supervisor in the table is NULL, thus, query XXIII is for case b).

For query XXIII, the comparison is evaluated in the same way of query XXI as an exceptional case whether NULL is equal to NULL. According to truth table in Table V, equality of two NULL's is unknown. The query XXIII returns an empty relation on every database engines. The semantics of NULL in the SUPERVISOR column for the main query in Query XXIII is unknown. Because his/her supervisor in the main query is missing, the query cannot determine whether he/she has the same supervisor as of Mike. However, this employee does not have a supervisor; which is not-applicable for the NULL semantics; therefore, this employee does not have the same supervisor as of Mike. Thus, the NULL in the SUPERVISOR column for the main query of Query XXIII means unknown.

For query XXII, x = { } is false in SQL where x is a constant. If x is null, regardless the meaning of missing or not-applicable, x = { } is false in every databases. Therefore, NULL in query XXII means unknown.

This is also consistent with other cases whether NULL is a member of a set containing NULL as in query XXIV.

Query XXIV

```
SELECT * FROM EMPLOYEE
WHERE SUPERVISOR IN (SELECT SUPERVISOR
                     FROM EMPLOYEE
                     WHERE ENAME = 'MIKE')
```

Query XXIV is valid due to set comparison operator IN. NULL $\in$ { NULL, NULL } is unknown for each tuple because it can be rewritten as "NULL = NULL or NULL = NULL".

VII. PRACTICAL APPROACHES

Table X is a summary of the meanings of NULL values in three cases; NULL in the subquery, NULL in the main query and NULL in both subquery and main query. The meanings are interpreted according the proposition of WHERE condition using IN comparison operator and EXISTS quantifier.

Summary of the meaning of NULL values when the tables of the main query and subquery have no foreign key referencing one to another is shown in Table XI.

To consider meanings of NULL values, we present two methodologies; designing of databases and applications to handle values of NULL and extending database engines to support the meaning of NULL values.

A. Design of Database and application

1. A column that cannot contain NULL, the designer should use "NOT NULL" constraint, when creating a table, to avoid missing value.

2. Database designers add two Boolean columns for columns that are critical for the meaning of NULL values. One additional column is for missing and not-applicable for another. For example, if the WORKDEP column is NULL and we know that it is a missing value, user (or application) must mark TRUE and FALSE into these two added columns respectively. When writing a query, programmer must check these new columns. The existing tables and applications require no change but only those that need to handle the meaning of NULL values.

3. There is no change at the design of database but programmer must consider the meaning of NULL when querying data. This includes the consideration of foreign key among tables in the query as shown in Tables X and XI. For example, if a query is intended for the meaning of NULL as not-applicable, and programmer wants to use NOT IN, the query must add IS NOT NULL in the subquery such as:

```
SELECT * FROM DEPARTMENT
WHERE DEPNO NOT IN (SELECT WORKDEP
        FROM EMPLOYEE
      WHERE WORKDEP IS NOT NULL)
```

Other features of SQL such as NULLIF and COALEASE expressions should be considered when handling NULL values.

4. NULL can be handled at the level of application programs when a database returns null values to a running process with various techniques; [1], [7] and [11] to avoid errors of the application.

Table X. Meanings of NULL values when there is a foreign key referencing between tables in the main query and subquery.

NULL Locality	Subquery	Main Query	Subquery and Main Queries
NOT IN	Missing	Missing	Missing
IN	Not-app.	Not-app.	Not-app.
NOT EXISTS	Not-app.	Not-app.	Not-app.
EXISTS	Not-app.	Not-app.	Not-app.

Table XI. Meanings of NULL values when there is no foreign key referencing between tables in the main query and subquery.

NULL Locality	Subquery	Main Query	Subquery and Main Queries
NOT IN	Missing	Missing	Missing
IN	Unknown	Unknown	Unknown
NOT EXISTS	Unknown	Unknown	Unknown
EXISTS	Unknown	Unknown	Unknown

Table XII. Two Additional Bits Adding to Nullable Column by Database Engines.

Value	M-bit	NA-bit	Meaning
X	0	0	Not-null
NULL	0	1	Not-applicable
NULL	1	0	Missing
NULL	1	1	Unknown

B. Extension of Database engines

Database engines can support three–valued NULL by adding two bits for each nullable column. The meanings of two additional bits are shown in Table XII.

The M-bit and NA-bit represent the meaning of missing and not-applicable respectively. It would be an option for database users to choose the intended meanings of their NULL values. This is a similar approach to [18] using A-mark, I-mark, and U-mark. Existing SQL of application is still valid where its NULL value means unknown. But SQL can accept two new values; Missing and Not-applicable, additional to the NULL only. [18], also, recommends the logical truth table among true, false, unknown, missing, and not-applicable. But it is out of scope of this paper and will be considered in our future work.

VIII. CONCLUSION AND DISCUSSIONS

A programmer who writing a standard SQL statement likely assumes that database engine will return correct answers without considering effects of NULL locality. Three meaning of NULL can be handled at the database design or application phase whereas database engines can provide this feature to programmer with the set theory and three-valued logic.

While writing a query or application, the programmer must consider both three meanings of NULL and the truth tables and the predicate semantics of the three-valued logic so when retrieving information from databases, they can expect the correct consequential result.

There are other cases that SQL is not perfect when considering the meaning of NULL with the result of a SQL query. For example, FALSE returning, when comparing an expression with a NULL value, misleads the results from the meaning. Such as

```
SELECT * FROM EMPLOYEE E1, EMPLOYEE E2
WHERE E1.EMPID  = E2.EMPID AND
      E1.WORKDEP = E2.WORKDEP
```

The result of this query does not return Tom whose WORKDEP is null, because NULL = NULL is false. However other employees are listed because they are working at the same place as themselves. One may argue that if Tom does not assign to a department, then Tom does not work in the same place with himself. It is a paradox.

Another example of NULL misleading is:

```
SELECT * FROM EMPLOYEE
WHERE SALARY > 100 OR SALARY <= 100
```

The result does not show Ann whose salary column is NULL. Salary of Ann cannot be missing. Otherwise, it would be true in one of the condition. However, it cannot be not-applicable too because if Ann does not have salary, she would not get paycheck which is the same to a person who receives salary zero. In this case, the second condition is true and Ann should be in the result.

NULL is handled well in XML due to the fact that XML provide programmers arbitrary elements and attributes. Programmers can define their own ways to handle NULL values in XML schemas. XML schemas provide basic features for null such as “nillable” and xsi:nil attributes for

provision of null values. However, some database engines may not support all features of XML. XML is still evolving. There are issues that XML has to gain its grounds before database vendors fully adopt them.

At last, the key factor of a success application is up to application programmers who are responsible to understand business logic, database design and query analysis because submitting a query to the database engine. Application programmers must be aware of null values when submitting a query to the database and handle them well when a database returns them during the run-time to avoid errors of the application. Therefore, the responsibility is not only for the database designers but also the application programmers to tackle problem of the semantics of NULL values.

REFERENCES

[1] M. D. Bond, N. Nethercote, S. W. Kent, S. Z. Guyer, K. S. McKinley, "Tracking bad apples: reporting the origin of null and undefined value errors", Proc. 22nd annual ACM SIGPLAN conf. on OOP Sys. and App. Montreal, Quebec, Canada, 25-27 Oct 2007, pp. 405 – 422

[2] B. Cao, A. Badia, "SQL query optimization through nested relational algebra", ACM Tran. on DB. Sys., Vol. 32, No. 3, Art. 18, Aug. 2007

[3] E. F. Codd, "More commentary on missing information (applicable and inapplicable information)", SIGMOD RECORD 16(1), Mar 1987, pp. 42-27

[4] E. F. Codd, "The relational model for database management: version 2", Addison-Wesley Longman Publishing., January 1990

[5] C. J. Date, "A critique of the SQL database language", Dec. 1983.

[6] C. J. Date, "Null Values in Database Management. In Relational Databases: Selected Writings", Addison-Wesley, Mass. 1986.

[7] I. Dillig, T. Dillig, A. Aiken, "Static error detection using semantic inconsistency inference", Proc. 2007 ACM SIGPLAN Conf. on Prog. Lang. Design and Imp. PLDI '07, San Diego, CA, USA Vol. 42 Issue 6 June 2007, pp. 435 - 445

[8] M. Elhemali, C. A. Galindo-Legaria, T. Grabs, M. M. Joshi, "Execution strategies for SQL subqueries", SIGMOD'07 June 12-14 2007, Beijing, Chaina, pp. 993-1004

[9] G. H. Gessert, "Four valued logic for relational database systems", SIGMOD RECORD, Vol.19, No. 1, Mar. 1990, pp.29-35

[10] A. Halverson, V. Josifovski, G. Lohman, H. Pirahesh, M. Morschel, "ROX: Relaional over XML" Proc. 30th VLDB Conf., Toronto, Canada, 2004, pp. 264-275

[11] P. G. Joisha, "Formal semantics and static analysis: compiler optimizations for nondeferred reference: counting garbage collection", Proc. 5th Inter. Symp. on Mem. Man. Ottawa, Ontario, Canada June 2006, pp. 150 - 161

[12] H. J. Klein, "How to modify SQL queries in order to guarantee sure answers", SIGMOD, Vol. 23, No. 3, September 1994, pp.14 – 20

[13] S. Link, "On the logical implication of multivalued dependencies with null values", 12th Computing: The Australasian Theory Symposium (CATS), Hobart, Tasmania. Conferences in Research and Practice in Info. Tech., Vol.51. 2006

[14] J. Melton, A. R. Simon, "SQL:1999 Understanding Relational Language Components", Morgan Kaufmann Publishers, 2002

[15] J. Tuya, M. J. Suarez-Cabal, C. Riva, "A practical guide to SQL white-box testing", ACM SIGPLAN Notices Vol. 41 (4), Apr. 2006, pp. 36-41

[16] Y. Vassiliou, Null values in data base management A denotational semantics approach", ACM 1979

[17] E. A. Walker, "Stone Algebras, Conditional Events, and Three Valued Logic", IEEE Trans. Sys. Man, and Cyb. Vol. 24, No. 12, Dec 1994 pp. 1699 -1707

[18] K. Yue, "A more general model for handling missing information in relational databases using a 3-valued logic", SIGMOD Vol.20, No. 3, Sept. 1991, pp.43-49