SQL and Incomp?ete Data

A not so happy marriage

Dr Paolo Guagliardo

Applied Databases, Guest Lecture – 31 March 2016

SQL is efficient, correct and reliable

SQL is efficient, correct and reliable
. . . as long as data is **complete**

When data is missing, things get **ugly**

SQL is efficient, correct and reliable
. . . as long as data is **complete**

When data is missing, things get **ugly**

Very poor design choices in the SQL standard

NULL: all-purpose marker to represent (different kinds of) missing information
Main source of problems and inconsistencies

SQL is efficient, correct and reliable
. . . as long as data is **complete**

When data is missing, things get **ugly**

Very poor design choices in the SQL standard

`NULL`: all-purpose marker to represent (different kinds of) missing information
Main source of problems and inconsistencies

> *". . . this topic cannot be described in a manner that is simultaneously both comprehensive and comprehensible"*
>
> *"Those SQL features are . . . fundamentally at odds with the way the world behaves"*
>
> — *C. Date & H. Darwen,* A Guide to SQL Standard

What's in a Null?

Two types of missing information:

- **Unknown** – there is a value, but we do not know which one
- **Non-Applicable** – a value does not exist

What's in a Null?

Two types of missing information:

- **Unknown** – there is a value, but we do not know which one
- **Non-Applicable** – a value does not exist

The problem is that **`NULL`** can represent both without distinction

PERSON

Name	**NINo**
Jane	**`NULL`**
John	**`NULL`**

What's in a Null?

Two types of missing information:

- **Unknown** – there is a value, but we do not know which one
- **Non-Applicable** – a value does not exist

The problem is that **NULL** can represent both without distinction

PERSON

Name	**NINo**
Jane	**NULL**
John	**NULL**

Does Jane have a National Insurance number?

What's in a Null?

Two types of missing information:

- **Unknown** – there is a value, but we do not know which one
- **Non-Applicable** – a value does not exist

The problem is that **NULL** can represent both without distinction

PERSON

Name	NINo
Jane	NULL
John	NULL

Does Jane have a National Insurance number?
Does John have a National Insurance number?

Unknown Vs. Non-Applicable Values

Person

Name	NINo
Jane	NULL
John	NULL

No information on whether Jane/John have a NINo (albeit its value is unknown) or whether they do not have a NINo at all

Unknown Vs. Non-Applicable Values

PERSON

Name	NINo
Jane	NULL
John	NULL

No information on whether Jane/John have a NINo (albeit its value is unknown) or whether they do not have a NINo at all

PERSONMOREINFO

Name	HasNINo	NINo
Jane	Yes	NULL
John	No	NULL

Unknown Vs. Non-Applicable Values

Person

Name	NINo
Jane	NULL
John	NULL

No information on whether Jane/John have a NINo (albeit its value is unknown) or whether they do not have a NINo at all

PersonMoreInfo

Name	HasNINo	NINo	
Jane	Yes	NULL	← unknown
John	No	NULL	

Unknown Vs. Non-Applicable Values

Person

Name	NINo
Jane	**NULL**
John	**NULL**

No information on whether Jane/John have a NINo (albeit its value is unknown) or whether they do not have a NINo at all

PersonMoreInfo

Name	HasNINo	NINo	
Jane	Yes	**NULL**	← unknown
John	No	**NULL**	← non-applicable

Unknown Vs. Non-Applicable Values

Person

Name	NINo
Jane	NULL
John	NULL

No information on whether Jane/John have a NINo (albeit its value is unknown) or whether they do not have a NINo at all

PersonMoreInfo

Name	HasNINo	NINo	
Jane	Yes	NULL	← unknown
John	No	NULL	← non-applicable

Person = **SELECT** Name, NINo **FROM** PersonMoreInfo

Getting Rid of Non-Applicable Values

PersonWithNINo

Name	NINo
Jane	NULL

PersonWithoutNINo

Name
John

Getting Rid of Non-Applicable Values

PersonWithNINo

Name	NINo
Jane	NULL

```
SELECT Name, NINo
FROM   PersonMoreInfo
WHERE  HasNINo = 'Yes'
```

PersonWithoutNINo

Name
John

```
SELECT Name
FROM   PersonMoreInfo
WHERE  HasNINo = 'No'
```

Getting Rid of Non-Applicable Values

PersonWithNINo

Name	NINo
Jane	**NULL**

```
SELECT Name, NINo
FROM   PersonMoreInfo
WHERE  HasNINo = 'Yes'
```

PersonWithoutNINo

Name
John

```
SELECT Name
FROM   PersonMoreInfo
WHERE  HasNINo = 'No'
```

PersonMoreInfo can be reconstructed as follows:

```
SELECT Name, 'Yes' AS HasNINo, NINo
FROM   PersonWithNINo
UNION
SELECT Name, 'No', NULL
FROM   PersonWithoutNINo
```

Nulls as Non-Applicable Values

Can be avoided with a better schema design

- Makes data semantics clearer (and queries easier to write)
- Saves storage space

Nulls as Non-Applicable Values

Can be avoided with a better schema design

- Makes data semantics clearer (and queries easier to write)
- Saves storage space

Occur naturally in **outer joins**

A	B
0	1
1	2
2	3

NATURAL LEFT JOIN

A	C
0	a
2	b

=

A	B	C
0	1	a
1	2	**NULL**
2	3	b

Nulls as Non-Applicable Values

Can be avoided with a better schema design

- Makes data semantics clearer (and queries easier to write)
- Saves storage space

Occur naturally in **outer joins**

A	B
0	1
1	2
2	3

NATURAL LEFT JOIN

A	C
0	a
2	b

=

A	B	C
0	1	a
1	2	**NULL**
2	3	b

If you cannot do without them,
at least be aware of what you are dealing with

Nulls in Selection Conditions (1)

What is the answer to

Q_1: `SELECT * FROM R, S WHERE R.A = S.A`

Q_2: `SELECT * FROM R, S WHERE R.A <> S.A`

Q_3: `SELECT * FROM R, S WHERE R.A = S.A OR R.A <> S.A`

when $R = \{1, \textbf{NULL}\}$ and $S = \{\textbf{NULL}\}$?

Nulls in Selection Conditions (1)

What is the answer to

Q_1: `SELECT * FROM R, S WHERE R.A = S.A`

Q_2: `SELECT * FROM R, S WHERE R.A <> S.A`

Q_3: `SELECT * FROM R, S WHERE R.A = S.A OR R.A <> S.A`

when $R = \{1,$ **NULL**$\}$ and $S = \{$**NULL**$\}$?

R.A
1
NULL

$\times$

S.A
NULL

$=$

R.A	S.A
1	NULL
NULL	NULL

Nulls in Selection Conditions (1)

What is the answer to

Q_1: **SELECT** * **FROM** R, S **WHERE** R.A = S.A

Q_2: **SELECT** * **FROM** R, S **WHERE** R.A <> S.A

Q_3: **SELECT** * **FROM** R, S **WHERE** R.A = S.A **OR** R.A <> S.A

when $R = \{1, \textbf{NULL}\}$ and $S = \{\textbf{NULL}\}$?

R.A
1
NULL

×

S.A
NULL

=

R.A	S.A
1	**NULL**
NULL	**NULL**

Answer to all three queries: $\{\}$

Evaluation of Selection Conditions

SQL uses three truth values: **true** (t), **false** (f), **unknown** (u)

Evaluation of Selection Conditions

SQL uses three truth values: **true** (t), **false** (f), **unknown** (u)

1. Every comparison (except **`IS NULL`** and **`IS NOT NULL`**) where one of the arguments is **`NULL`** evaluates to unknown

Evaluation of Selection Conditions

SQL uses three truth values: **true** (t), **false** (f), **unknown** (u)

1. Every comparison (except **`IS NULL`** and **`IS NOT NULL`**) where one of the arguments is **`NULL`** evaluates to unknown
2. The truth values assigned to each comparison are propagated using the following tables:

AND	t	f	u
t	t	f	u
f	f	f	f
u	u	f	u

OR	t	f	u
t	t	t	t
f	t	f	u
u	t	u	u

	NOT
t	f
f	t
u	u

Evaluation of Selection Conditions

SQL uses three truth values: **true** (t), **false** (f), **unknown** (u)

1. Every comparison (except **IS NULL** and **IS NOT NULL**) where one of the arguments is **NULL** evaluates to unknown
2. The truth values assigned to each comparison are propagated using the following tables:

AND	t	f	u
t	t	f	u
f	f	f	f
u	u	f	u

OR	t	f	u
t	t	t	t
f	t	f	u
u	t	u	u

	NOT
t	f
f	t
u	u

3. The rows for which the condition evaluates to true are returned

Nulls in Selection Conditions (2)

What is the answer to

Q_1: `SELECT * FROM R, S WHERE R.A = S.A`

Q_2: `SELECT * FROM R, S WHERE R.A <> S.A`

Q_3: `SELECT * FROM R, S WHERE R.A = S.A OR R.A <> S.A`

when $R = \{1, \textbf{NULL}\}$ and $S = \{\textbf{NULL}\}$?

Nulls in Selection Conditions (2)

What is the answer to

Q_1: **SELECT** * **FROM** R, S **WHERE** R.A = S.A

Q_2: **SELECT** * **FROM** R, S **WHERE** R.A <> S.A

Q_3: **SELECT** * **FROM** R, S **WHERE** R.A = S.A **OR** R.A <> S.A

when $R = \{1,$ **NULL**$\}$ and $S = \{$**NULL**$\}$?

R.A	S.A
1	**NULL**
NULL	**NULL**

Nulls in Selection Conditions (2)

What is the answer to

Q_1: `SELECT * FROM R, S WHERE R.A = S.A`

Q_2: `SELECT * FROM R, S WHERE R.A <> S.A`

Q_3: `SELECT * FROM R, S WHERE R.A = S.A OR R.A <> S.A`

when $R = \{1, \textbf{NULL}\}$ and $S = \{\textbf{NULL}\}$?

R.A	S.A
1	**NULL**
NULL	**NULL**

c_1
u
u

Nulls in Selection Conditions (2)

What is the answer to

Q_1: `SELECT * FROM R, S WHERE R.A = S.A`

Q_2: `SELECT * FROM R, S WHERE R.A <> S.A`

Q_3: `SELECT * FROM R, S WHERE R.A = S.A OR R.A <> S.A`

when $R = \{1, \textbf{NULL}\}$ and $S = \{\textbf{NULL}\}$?

R.A	S.A
1	NULL
NULL	NULL

c_1
u
u

c_2
u
u

Nulls in Selection Conditions (2)

What is the answer to

Q_1: `SELECT * FROM R, S WHERE R.A = S.A`

Q_2: `SELECT * FROM R, S WHERE R.A <> S.A`

Q_3: `SELECT * FROM R, S WHERE R.A = S.A OR R.A <> S.A`

when $R = \{1, \textbf{NULL}\}$ and $S = \{\textbf{NULL}\}$?

R.A	S.A
1	**NULL**
NULL	**NULL**

c_1
u
u

c_2
u
u

c_3
u
u

Nulls and Set Operations

What is the answer to

Q_1: `SELECT * FROM R UNION     SELECT * FROM S`

Q_2: `SELECT * FROM R INTERSECT SELECT * FROM S`

Q_3: `SELECT * FROM R EXCEPT    SELECT * FROM S`

when $R = \{1, \textbf{NULL}\}$ and $S = \{\textbf{NULL}\}$?

- Answer to Q_1:
- Answer to Q_2:
- Answer to Q_3:

Nulls and Set Operations

What is the answer to

```
Q1: SELECT * FROM R UNION     SELECT * FROM S
Q2: SELECT * FROM R INTERSECT SELECT * FROM S
Q3: SELECT * FROM R EXCEPT    SELECT * FROM S
```

when $R = \{1, \textbf{NULL}\}$ and $S = \{\textbf{NULL}\}$?

- Answer to Q_1: $\{1, \textbf{NULL}\}$
- Answer to Q_2:
- Answer to Q_3:

Nulls and Set Operations

What is the answer to

Q_1: **SELECT** * **FROM** R **UNION** **SELECT** * **FROM** S

Q_2: **SELECT** * **FROM** R **INTERSECT** **SELECT** * **FROM** S

Q_3: **SELECT** * **FROM** R **EXCEPT** **SELECT** * **FROM** S

when $R = \{1, \textbf{NULL}\}$ and $S = \{\textbf{NULL}\}$?

- Answer to Q_1: $\{1, \textbf{NULL}\}$
- Answer to Q_2: $\{\textbf{NULL}\}$
- Answer to Q_3:

Nulls and Set Operations

What is the answer to

Q_1: `SELECT * FROM R UNION SELECT * FROM S`

Q_2: `SELECT * FROM R INTERSECT SELECT * FROM S`

Q_3: `SELECT * FROM R EXCEPT SELECT * FROM S`

when $R = \{1, \textbf{NULL}\}$ and $S = \{\textbf{NULL}\}$?

- Answer to Q_1: $\{1, \textbf{NULL}\}$
- Answer to Q_2: $\{\textbf{NULL}\}$
- Answer to Q_3: $\{1\}$

Nulls and Set Operations

What is the answer to

```
Q1: SELECT * FROM R UNION     SELECT * FROM S
Q2: SELECT * FROM R INTERSECT SELECT * FROM S
Q3: SELECT * FROM R EXCEPT    SELECT * FROM S
```

when $R = \{1, \textbf{NULL}\}$ and $S = \{\textbf{NULL}\}$?

- Answer to Q_1: $\{1, \textbf{NULL}\}$
- Answer to Q_2: $\{\textbf{NULL}\}$
- Answer to Q_3: $\{1\}$

In set operations **NULL** is treated like any other value

Nulls and Query Equivalence

Q_1

```
SELECT R.A FROM R
INTERSECT
SELECT S.A FROM S
```

Q_2

```
SELECT DISTINCT R.A
FROM   R, S
WHERE  R.A = S.A
```

On databases without nulls, Q_1 and Q_2 give the same answers

Nulls and Query Equivalence

Q_1

```
SELECT R.A FROM R
INTERSECT
SELECT S.A FROM S
```

Q_2

```
SELECT DISTINCT R.A
FROM    R, S
WHERE   R.A = S.A
```

On databases without nulls, Q_1 and Q_2 give the same answers

On databases **with nulls**, they do not

For example, when $R = S = \{$**NULL**$\}$

- Q_1 returns $\{$**NULL**$\}$
- Q_2 returns $\{\}$

Nulls and Arithmetic Operations

Every arithmetic operation that involves a **NULL** results in **NULL**

```
SELECT 1+NULL AS sum , 1-NULL AS diff,
       1*NULL AS mult, 1/NULL AS div
```

```
 sum  | diff | mult | div
------+------+------+------
 NULL | NULL | NULL | NULL
(1 row)
```

Nulls and Arithmetic Operations

Every arithmetic operation that involves a **NULL** results in **NULL**

```
SELECT 1+NULL AS sum , 1-NULL AS diff,
       1*NULL AS mult, 1/NULL AS div
```

```
 sum  | diff | mult | div
------+------+------+------
 NULL | NULL | NULL | NULL
(1 row)
```

Observe that `SELECT NULL/0` also returns **NULL** instead of throwing a DIVISION BY ZERO error!

Nulls and Aggregation (1)

Aggregate functions ignore nulls

Consider $R = \{0, 1, \textbf{NULL}\}$ on attribute A

```
SELECT MIN(A), MAX(A), COUNT(A), SUM(A),
       CAST( AVG(A) AS numeric(2,1) )
FROM   R
```

```
 min | max | count | sum | avg
-----+-----+-------+-----+-----
   0 |   1 |     2 |   1 | 0.5
(1 row)
```

Nulls and Aggregation (1)

Aggregate functions ignore nulls

Consider $R = \{0, 1, \textbf{NULL}\}$ on attribute A

Exception:

```
SELECT COUNT(*) FROM R
```

```
 count
-------
     3
(1 row)
```

Nulls and Aggregation (2)

Applying an aggregate function other than **COUNT** to an empty set results in **NULL**

Consider $R = \{0, 1, \textbf{NULL}\}$ on attribute A

```
SELECT MIN(A), MAX(A), SUM(A), AVG(A), COUNT(A)
FROM   R
WHERE  A = 2
```

```
 min  | max  | sum  | avg  | count
------+------+------+------+-------
 NULL | NULL | NULL | NULL |     0
(1 row)
```

The semantics of these nulls is that of **non-applicable** values

Possible Worlds

An incomplete database can represent (infinitely) many complete databases, depending on how the missing values are interpreted

Each of these **possible worlds** corresponds to a substitution of actual values for the nulls

Name	**Age**
Jane	**NULL**
John	**NULL**
Mary	27

Possible Worlds

An incomplete database can represent (infinitely) many complete databases, depending on how the missing values are interpreted

Each of these **possible worlds** corresponds to a substitution of actual values for the nulls

Name	**Age**
Jane	45
John	1
Mary	27

Possible Worlds

An incomplete database can represent (infinitely) many complete databases, depending on how the missing values are interpreted

Each of these **possible worlds** corresponds to a substitution of actual values for the nulls

Name	**Age**
Jane	18
John	90
Mary	27

Possible Worlds

An incomplete database can represent (infinitely) many complete databases, depending on how the missing values are interpreted

Each of these **possible worlds** corresponds to a substitution of actual values for the nulls

Name	**Age**
Jane	28
John	29
Mary	27

Possible Worlds

An incomplete database can represent (infinitely) many complete databases, depending on how the missing values are interpreted

Each of these **possible worlds** corresponds to a substitution of actual values for the nulls

Name	**Age**
Jane	27
John	27
Mary	27

Certain Answers

Answers independent of the interpretation of missing information

Certain Answers

Answers independent of the interpretation of missing information

For a query Q and a database D, they are defined as

$$\text{certain}(Q, D) = \bigcap Q(D')$$

over all possible worlds D' described by D

Certain Answers

Answers independent of the interpretation of missing information

For a query Q and a database D, they are defined as

$$\text{certain}(Q, D) = \bigcap Q(D')$$

over all possible worlds D' described by D

D:

Name	Age
Jane	**NULL**
John	**NULL**
Mary	27

Q: “people over 18”

certain$(Q, D) = \{$(Mary, 27)$\}$

Certain Answers

Answers independent of the interpretation of missing information

For a query Q and a database D, they are defined as

$$\text{certain}(Q, D) = \bigcap Q(D')$$

over all possible worlds D' described by D

D:

Name	Age
Jane	**NULL**
John	**NULL**
Mary	27

Q: "people over 18"

$\text{certain}(Q, D) = \{(\text{Mary}, 27)\}$

Provide the notion of **correctness** on incomplete databases

SQL and Correctness

> *"If you have any nulls in your database, you're getting wrong answers to some of your queries. What's more, you have no way of knowing, in general, just which queries you're getting wrong answers to; all results become suspect. You can never trust the answers you get from a database with nulls"*
>
> — *C. Date*, Database in Depth

Wrong Answers in SQL

Orders

ord_id	price
ord1	30
ord2	15
ord3	50

Payments

ord_id	pay_date
ord1	2015-10-12
NULL	2015-12-11

Q: list unpaid orders

```
SELECT O.ord_id
FROM   Orders O
WHERE  NOT EXISTS (
  SELECT *
  FROM   Payments P
  WHERE  P.ord_id = O.ord_id )
```

Wrong Answers in SQL

ORDERS

ord_id	price
ord1	30
ord2	15
ord3	50

PAYMENTS

ord_id	pay_date
ord1	2015-10-12
NULL	2015-12-11

Q: list unpaid orders

```
SELECT O.ord_id
FROM   Orders O
WHERE  NOT EXISTS (
  SELECT *
  FROM   Payments P
  WHERE  P.ord_id = O.ord_id )
```

Answer: $\{$ord2, ord3$\}$

Wrong Answers in SQL

ORDERS

ord_id	price
ord1	30
ord2	15
ord3	50

PAYMENTS

ord_id	pay_date
ord1	2015-10-12
NULL	2015-12-11

Q: list unpaid orders

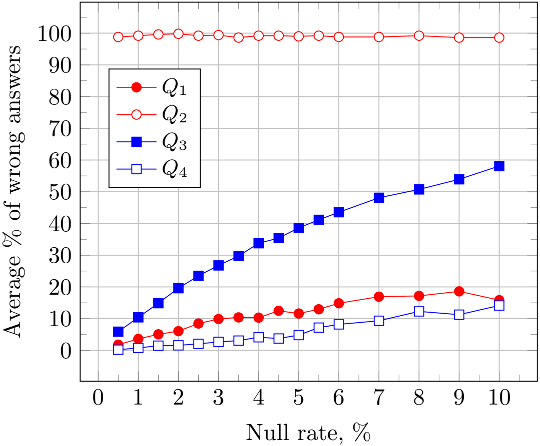
```
SELECT O.ord_id
FROM   Orders O
WHERE  NOT EXISTS (
  SELECT *
  FROM   Payments P
  WHERE  P.ord_id = O.ord_id )
```

Answer: $\{\text{ord2}, \text{ord3}\}$ **incorrect!** $\text{certain}(Q, D) = \{\}$

Is This a Real Problem?

Experiment

- Data from TPC-H Benchmark (models a business scenario)
- 4 queries with negation: 2 from TPC-H, 2 from a textbook
- Approximate algorithms for detecting wrong answers



Correctness in SQL (1)

It can be achieved for a restricted subset of the language:

- No duplicate elimination (**DISTINCT** not allowed)
- No set operations (i.e., **UNION**, **INTERSECT**, **EXCEPT**)
- No grouping/aggregation
- No explicit **NOT** in **WHERE** conditions
- Semijoins (**EXISTS**)
- Antijoins (**NOT EXISTS**)
- Restricted theta-joins, allowed only inside:
  - a projection on non-nullable attributes, or
  - the right-hand side of a semijoin or antijoin

Correctness in SQL (2)

Main idea: translate a query Q into a pair $Q^+, Q^?$ where

- Q^+ approximates correct answers
- $Q^?$ represents potential answers

$$
\begin{aligned}
R^+ &= R \\
\big(\sigma_\theta(Q)\big)^+ &= \sigma_{\theta^+}(Q^+) \\
\big(\pi_\alpha(Q)\big)^+ &= \pi_\alpha(Q^+) \\
(Q_1 \bowtie_\theta Q_2)^+ &= Q_1^+ \bowtie_{\theta^+} Q_2^+ \\
(Q_1 \ltimes_\theta Q_2)^+ &= Q_1^+ \ltimes_{\theta^+} Q_2^+ \\
(Q_1 \overline{\ltimes}_\theta Q_2)^+ &= Q_1^+ \overline{\ltimes}_{\theta^?} Q_2^?
\end{aligned}
\qquad
\begin{aligned}
R^? &= R \\
\big(\sigma_\theta(Q)\big)^? &= \sigma_{\theta^?}(Q^?) \\
\big(\pi_\alpha(Q)\big)^? &= \pi_\alpha(Q^?) \\
(Q_1 \bowtie_\theta Q_2)^? &= Q_1^? \bowtie_{\theta^?} Q_2^? \\
(Q_1 \ltimes_\theta Q_2)^? &= Q_1^? \ltimes_{\theta^?} Q_2^? \\
(Q_1 \overline{\ltimes}_\theta Q_2)^? &= Q_1^? \overline{\ltimes}_{\theta^+} Q_2^+
\end{aligned}
$$

Key task: translating selection/join conditions θ

Translation of Conditions

$$(A \,\mathsf{op}\, B)^{+} = (A \,\mathsf{op}\, B) \wedge \mathsf{not\_null}(A) \wedge \mathsf{not\_null}(B)$$

$$(A \,\mathsf{op}\, c)^{+} = (A \,\mathsf{op}\, c) \wedge \mathsf{not\_null}(A) \qquad (c \text{ is a constant})$$

$$(\theta_1 \wedge \theta_2)^{+} = \theta_1^{+} \wedge \theta_2^{+}$$

$$(\theta_1 \vee \theta_2)^{+} = \theta_1^{+} \vee \theta_2^{+}$$

$$(A \,\mathsf{op}\, B)^{?} = (A \,\mathsf{op}\, B) \vee \mathsf{null}(A) \vee \mathsf{null}(B)$$

$$(A \,\mathsf{op}\, c)^{?} = (A \,\mathsf{op}\, c) \vee \mathsf{null}(A) \qquad (c \text{ is a constant})$$

$$(\theta_1 \wedge \theta_2)^{?} = \theta_1^{?} \wedge \theta_2^{?}$$

$$(\theta_1 \vee \theta_2)^{?} = \theta_1^{?} \vee \theta_2^{?}$$

Example of Translation

```
SELECT o_orderkey
FROM   orders
WHERE  NOT EXISTS (
       SELECT *
       FROM   lineitem
       WHERE  l_orderkey = o_orderkey


         AND  l_suppkey <> $supp_key

)
```

In relational algebra: $\pi_{\texttt{o\_orderkey}}\big(\texttt{orders}\ \overline{\ltimes}_{\theta}\ \texttt{lineitem}\big)$
where θ is the condition in the **WHERE** clause

Example of Translation

```sql
SELECT o_orderkey
FROM   orders
WHERE  NOT EXISTS (
       SELECT *
       FROM   lineitem
       WHERE  ( l_orderkey = o_orderkey
                OR l_orderkey IS NULL
                OR o_orderkey IS NULL )
         AND  ( l_suppkey <> $supp_key
                OR l_suppkey IS NULL )
)
```

In relational algebra: $\pi_{\texttt{o\_orderkey}}\left(\texttt{orders}\ \overline{\ltimes}_{\theta}\ \texttt{lineitem}\right)$
where θ is the condition in the **WHERE** clause

Example of Translation

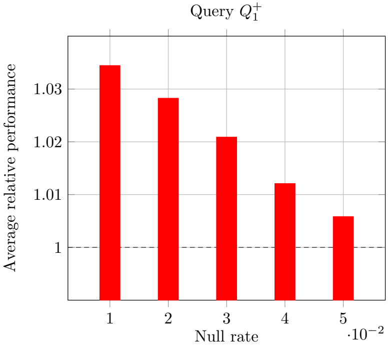
```
SELECT o_orderkey
FROM   orders
WHERE  NOT EXISTS (
       SELECT *
       FROM   lineitem
       WHERE  l_orderkey = o_orderkey


         AND  ( l_suppkey <> $supp_key
                OR l_suppkey IS NULL )
)
```

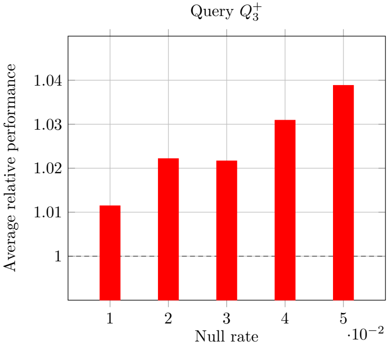
In relational algebra: $\pi_{\texttt{o\_orderkey}}\left(\texttt{orders}\ \overline{\ltimes}_{\theta}\ \texttt{lineitem}\right)$
where θ is the condition in the **WHERE** clause

Does It Work in Practice?

Yes, with mixed results

The good small overhead ($< 4\%$)

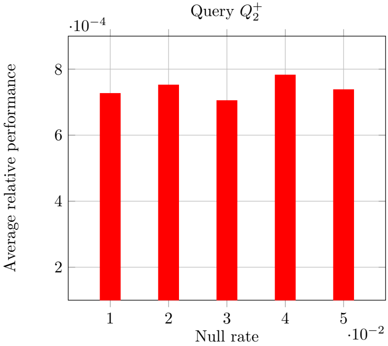




Does It Work in Practice?

Yes, with mixed results

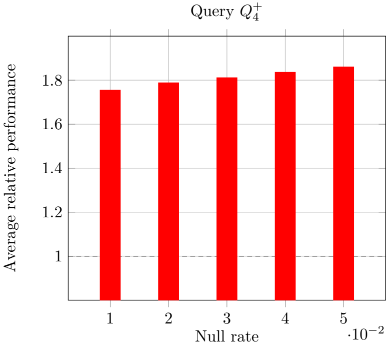
The fantastic significant speed-up (more than 10^3 times faster)



Does It Work in Practice?

Yes, with mixed results

The tolerable moderate slow-down (roughly half the speed)



Does It Work in Practice?

Yes, with mixed results

The **bad** and the **ugly** describe the status of query optimizers

Marked Nulls

Theoretical model where each missing value has an **identifier**

Person

Name	Age
Jane	**NULL:1**
John	**NULL:1**
Mary	27
Carl	**NULL:2**

We do not know what the age of Jane, John and Carl is
But we know that Jane and John have **the same** age

Marked Nulls

Theoretical model where each missing value has an **identifier**

Person

Name	Age
Jane	`NULL:1`
John	`NULL:1`
Mary	27
Carl	`NULL:2`

We do not know what the age of Jane, John and Carl is
But we know that Jane and John have **the same** age

- Allow cross-referencing of unknown values
- Indexable, seen as regular values by the optimizer

Conclusions

The way SQL handles incomplete data is **disastrous**

- Ambiguous (mixing different kinds of missing information)
- Inconsistent (nulls behave in conflicting ways)
- Incorrect (producing wrong answers)

Conclusions

The way SQL handles incomplete data is **disastrous**

- Ambiguous (mixing different kinds of missing information)
- Inconsistent (nulls behave in conflicting ways)
- Incorrect (producing wrong answers)

There is hope for a fix, but lots of work needs to be done
both theoretical and practical

- Aggregation
- Constraints
- Query optimization
- Marked nulls
- SQL-to-SQL translations
- More experiments