Figure 1: SQL Database

ORDERS	
ord_id	**ord_date**
ord1	2015-06-12
ord2	2015-07-11
ord3	2015-07-20

PAYMENTS		
pay_id	**ord_id**	**pay_date**
pay1	ord1	2015-06-14
pay2	NULL	2015-07-25

Figure 2: SQL Query

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

1.2 Correctness

In order to give a formal definition of correctness we need to give a few definitions. Much of the following is standard in the literature on databases with incomplete information, see, e.g., [[1], [8], [4], [7], [6]]. We consider incomplete databases with nulls interpreted as missing information. Databases are populated by two types of elements: constants and nulls, coming from countably infinite sets denoted by $Const$ and $Null$, respectively. Elements in $Null$ are denoted $\bot_t$, where t is a stamp, and behave accordingly to the marked nulls semantics. We call a valuation of nulls on an incomplete database D a map $v : Null(D) \to Const$ assigning a constant value to each null. According to this definition it's easy to understand that each element $\bot_t$ which might appears multiple times in the database will all be assign to the same constant by a valuation.

A typical definition of certain answers are those independent of the interpretation of missing information denoted $cert(Q, D) = \bigcap\{Q(v(D))|v \text{ is a valuation}\}$ where Q is a query and D a database. Such definition is a bit too restrictive as tuples with nulls can not be returned. A more general notion called certain answers with nulls, noted $cert_\bot(Q, D)$ is defined as :

$$cert_\bot(Q, D) = [\![a^n|\forall v, v(a) \in^n Q(v(D))]\!]$$

Those definition are closely related as if we remove tuples with null from $cert_\bot(Q, D)$ we obtain $cert(Q, D)$.

We say that a query evaluation algorithm, $Eval$, has correctness guarantees for a query Q if for every database D it returns a subset of $cert_\bot(Q, D)$.

We will focus on having an evaluation procedure for the basic fragment of SQL (i.e., first-order queries) which have correctness guarantee. Our evaluation algorithm will translate a query Q in Q' such that

$$\forall D, \forall a, (a \in^n Eval_{SQL}(Q', D) \implies a \in^n cert_\bot(Q, D))$$

.

1.3 SQL Query syntax

In this subsection we introduce a syntax in order to describe the basic fragment of SQL. Note that this is a personal choice as there is no real standard in the literature.

Definition. *We denote the set of well formed query without null test by* $[\![SQL]\!]$
We denote the Set of well formed query by $[\![SQL]\!]_\bot$

Definition. *Let's a Select query $Q \in [\![SQL]\!]_\perp$ a tuple (Σ, R, H, P) such that*

$$\Sigma \subseteq \{r_i.a_j | \exists r_i \in R \wedge \exists a_j \in r_i\}$$
$$P \subseteq \{p_i = r_j.a_k | r_j \notin R\} \textit{ a set of external parameter.}$$
$$R \textit{ a set of relation}$$
$$H \textit{ belongs to the following grammar.}$$

$$\begin{aligned} H ::= {} & r_i.a_j = c_k \mid r_i.a_j \neq c_k \mid r_i.a_j = r_k.a_l \mid r_i.a_j \neq r_k.a_l \mid r_i.a_j = p_k \mid r_i.a_j \neq p_k \mid \\ & r_i.a_j > c_k \mid r_i.a_j < c_k \mid r_i.a_j > r_k.a_l \mid r_i.a_j < r_k.a_l \mid r_i.a_j > p_k \mid r_i.a_j < p_k \mid \\ & null(r_i.a_j) \mid const(r_i.a_j) \mid null(p_i) \mid const(p_i) \mid \\ & exists(Q) \mid notexists(Q) \mid H \wedge H \mid H \vee H \end{aligned}$$

The query in 2 would be express as

$$(ord\_id, Orders, notexists((*, Payments, Payments.ord\_id = Order.ord\_id, \emptyset)), \emptyset)$$

Definition. *Let x a labeled-tuple we denote by $x[a]$ the value of the column labeled as a in the tuple.*
We denote $(\Sigma, R, H, P)[x]$ the query $(\Sigma, R, H, P \cup x)$.
We denote $(\Sigma, R, H, P)_*$ the query $(*, R, H, P)$.

$$\sigma_\Sigma(x) = (x[r_i.a_i] | r_i.a_i \in \Sigma)$$

$$\sigma_*(x) = x$$

$$\sigma_\Sigma(B) = [\![y^n | n = \sum_{x \in \{z | z \in \{B\} \wedge \sigma_\Sigma(z) = y\}} B(x)]\!]$$

Definition. *Let R be a set of relations then $R^\times$ denote the bag coming from the cartesian product of R.*

$$R^\times = \prod_{r \in R} r$$

Proposition 1. *Let R be a set of relation, and z a tuple then:*

$$(\forall v \textit{ valuation}, v(R^\times)(v(z)) \geq k) \Leftrightarrow R^\times(z) \geq k$$

This property is ensure due to the fact that a valuation v have to evaluate every marked null with the same stamp by the same constant that may not already appear in the database. (Here we assume that each attribute as an infinite number of possible value). The proof is straightforward but require a lot of notation, then it won't appear in this report. However this property does not hold for SQL nulls, which is one of the reason we choose to work with marked nulls.

1.4 SQL Evaluation semantic

In this sub-section we propose a reasonable semantic for the SQL evaluation. Keep in mind that it is only a reasonable one as it has yet to be proven that it fits the SQL evaluation. Such work have to be done either thanks to a deep code analysis, or more realistically an high number of practical tests.

Definition.

$$
\begin{aligned}
Eval_{SQL}((\Sigma,R,\emptyset,P),D) &= \sigma_\Sigma(R^\times)\\
Eval_{SQL}((\Sigma,R,H_1\wedge H_2,P),D) &= \sigma_\Sigma(Eval_{SQL}((*,R,H_1,P),D)\cap Eval_{SQL}((*,R,H_2,P),D))\\
Eval_{SQL}((\Sigma,R,H_1\vee H_2,P),D) &= \sigma_\Sigma(Eval_{SQL}((*,R,H_1,P),D)\cup Eval_{SQL}((*,R,H_2,P),D))\\
Eval_{SQL}((\Sigma,R,null(r_i.a_i),P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge\exists t,x[r_i.a_i]=\bot_t]\!])\\
Eval_{SQL}((\Sigma,R,const(r_i.a_i),P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge\forall t,x[r_i.a_i]\neq\bot_t]\!])\\
Eval_{SQL}((\Sigma,R,null(p_i),P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge\exists t,P[p_i]=\bot_t]\!])\\
Eval_{SQL}((\Sigma,R,const(p_i),P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge\forall t,P[p_i]\neq\bot_t]\!])\\
Eval_{SQL}((\Sigma,R,r_i.a_i=c_i,P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge x[r_i.a_i]=c_i]\!])\\
Eval_{SQL}((\Sigma,R,r_i.a_i=r_j.a_j,P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge x[r_i.a_i]=x[r_j.a_j]]\!])\\
Eval_{SQL}((\Sigma,R,r_i.a_i=p_i,P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge x[r_i.a_i]=P[p_i]]\!])\\
Eval_{SQL}((\Sigma,R,r_i.a_i\neq c_i,P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge x[r_i.a_i]\neq c_i\wedge\forall t,x[r_i.a_i]\neq\bot_t]\!])\\
Eval_{SQL}((\Sigma,R,r_i.a_i\neq r_j.a_j,P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge x[r_i.a_i]\neq x[r_j.a_j]\wedge\forall t,x[r_i.a_i]\neq\bot_t\wedge\forall t,x[r_j.a_j]\neq\bot_t]\!])\\
Eval_{SQL}((\Sigma,R,r_i.a_i\neq p_i,P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge x[r_i.a_i]\neq P[p_i]\wedge\forall t,x[r_i.a_i]\neq\bot_t\wedge\forall t,P[p_i]\neq\bot_t]\!])\\
Eval_{SQL}((\Sigma,R,r_i.a_i>c_i,P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge x[r_i.a_i]>c_i]\!])\\
Eval_{SQL}((\Sigma,R,r_i.a_i>r_j.a_j,P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge x[r_i.a_i]>x[r_j.a_j]]\!])\\
Eval_{SQL}((\Sigma,R,r_i.a_i>p_i,P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge x[r_i.a_i]>P[p_i]]\!])\\
Eval_{SQL}((\Sigma,R,r_i.a_i<c_i,P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge x[r_i.a_i]<c_i]\!])\\
Eval_{SQL}((\Sigma,R,r_i.a_i<r_j.a_j,P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge x[r_i.a_i]<x[r_j.a_j]]\!])\\
Eval_{SQL}((\Sigma,R,r_i.a_i<p_i,P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge x[r_i.a_i]<P[p_i]]\!])\\
Eval_{SQL}((\Sigma,R,exists(Q),P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge Eval_{SQL}(Q[x],D)\neq\emptyset]\!])\\
Eval_{SQL}((\Sigma,R,notexists(Q),P),D) &= \sigma_\Sigma([\![x^n|R^\times(x)=n\wedge Eval_{SQL}(Q[x],D)=\emptyset]\!])
\end{aligned}
$$

2 Evaluation with correctness guarantee

In order to compute a subset of certain answers with nulls for a Query $Q\in[\![SQL]\!]$, we translate it in $Q^+\in[\![SQL]\!]_\bot$ where the evaluation of Q^+ approximates certain answers. Consider the query such that $Q_1=(\Sigma,R,notexists(Q_2),P)$ we want a translation that ensure that we only obtain certain answers, then we have to be sure to also have a translation that guarantee to capture at least all certain answer in order to translate Q_2, otherwise we might create false positives. Let note $Q^?$ such translation.

Figure 3: Translation $Q\to(Q^+,Q^?)$

$$
\begin{aligned}
(\Sigma,R,H,P)^+ &\to (\Sigma,R,H^*,P)\\
(\Sigma,R,H,P)^? &\to (\Sigma,R,H^{**},P)
\end{aligned}
$$

Figure 4: Translation $H \rightarrow H^*$

$$
\begin{aligned}
(H_1 \wedge H_2)^* &\rightarrow H_1^* \wedge H_2^* \\
(H_1 \vee H_2)^* &\rightarrow H_1^* \vee H_2^* \\
(r_i.a_i = c_i)^* &\rightarrow r_i.a_i = c_i \\
(r_i.a_i \neq c_i)^* &\rightarrow r_i.a_i \neq c_i \wedge const(r_i.a_i) \\
(r_i.a_i > c_i)^* &\rightarrow r_i.a_i > c_i \\
(r_i.a_i < c_i)^* &\rightarrow r_i.a_i < c_i \\
(r_i.a_i = r_j.a_j)^* &\rightarrow r_i.a_i = r_j.a_j \\
(r_i.a_i \neq r_j.a_j)^* &\rightarrow r_i.a_i \neq r_j.a_j \wedge const(r_i.a_i) \wedge const(r_j.a_j) \\
(r_i.a_i > r_j.a_j)^* &\rightarrow r_i.a_i > r_j.a_j \\
(r_i.a_i < r_j.a_j)^* &\rightarrow r_i.a_i < r_j.a_j \\
(r_i.a_i = p_j)^* &\rightarrow r_i.a_i = p_j \\
(r_i.a_i \neq p_j)^* &\rightarrow r_i.a_i \neq p_j \wedge const(r_i.a_i) \wedge const(p_j) \\
(r_i.a_i > p_j)^* &\rightarrow r_i.a_i > p_j \\
(r_i.a_i < p_j)^* &\rightarrow r_i.a_i < p_j \\
exists(Q)^* &\rightarrow exists(Q^+) \\
notexists(Q)^* &\rightarrow notexists(Q^?)
\end{aligned}
$$

Figure 5: Translation $H \rightarrow H^{**}$

$$
\begin{aligned}
(H_1 \wedge H_2)^{**} &\rightarrow H_1^{**} \wedge H_2^{**} \\
(H_1 \vee H_2)^{**} &\rightarrow H_1^{**} \vee H_2^{**} \\
(r_i.a_i = c_i)^{**} &\rightarrow r_i.a_i = c_i \vee null(r_i.a_i) \\
(r_i.a_i \neq c_i)^{**} &\rightarrow r_i.a_i \neq c_i \\
(r_i.a_i > c_i)^{**} &\rightarrow r_i.a_i > c_i \vee null(r_i.a_i) \\
(r_i.a_i < c_i)^{**} &\rightarrow r_i.a_i < c_i \vee null(r_i.a_i) \\
(r_i.a_i = r_j.a_j)^{**} &\rightarrow r_i.a_i = r_j.a_j \vee null(r_i.a_i) \vee null(r_j.a_j) \\
(r_i.a_i \neq r_j.a_j)^{**} &\rightarrow r_i.a_i \neq r_j.a_j \\
(r_i.a_i > r_j.a_j)^{**} &\rightarrow r_i.a_i > r_j.a_j \vee ((null(r_i.a_i) \vee null(r_j.a_j)) \wedge r_i.a_i \neq r_j.a_j) \\
(r_i.a_i < r_j.a_j)^{**} &\rightarrow r_i.a_i < r_j.a_j \vee ((null(r_i.a_i) \vee null(r_j.a_j)) \wedge r_i.a_i \neq r_j.a_j) \\
(r_i.a_i = p_j)^{**} &\rightarrow r_i.a_i = p_j \vee null(r_i.a_i) \vee null(p_j) \\
(r_i.a_i \neq p_j)^{**} &\rightarrow r_i.a_i \neq p_j \\
(r_i.a_i > p_j)^{**} &\rightarrow r_i.a_i > p_j \vee ((null(r_i.a_i) \vee null(p_j)) \wedge r_i.a_i \neq p_j) \\
(r_i.a_i < p_j)^{**} &\rightarrow r_i.a_i < p_j \vee ((null(r_i.a_i) \vee null(p_j)) \wedge r_i.a_i \neq p_j) \\
exists(Q)^{**} &\rightarrow exists(Q^?) \\
notexists(Q)^{**} &\rightarrow notexists(Q^+)
\end{aligned}
$$

The key to understand this algorithm, is that $Q^?$ actually compute every answers that might be a certain answer. Indeed as soon as there exist a valuation such that $x \in Q(v(D))$ then $x \in Q^?(D)$. This property is ensure by the fact that the equality is replace with a disjunction checking if the attributes are nulls. As soon as one of the attribute is null then a valuation might exists which validate the equality. Conversely Q^+ ensure that difference are verify for every valuation, by replacing them with a conjunction ensuring that no nulls are involve. Then it's easy to understand that we can demonstrate that: if we assume that Q^+ evaluation only return certain answers, then $Q^?$ return at least all of them. And if we assume that $Q^?$ computes at least every certain answers, then Q^+ will only return certain answers.

Lemma 1.

$$\forall Q \in [\![SQL]\!], \sigma_\Sigma(cert_\perp(Q_*, D)) = cert_\perp(Q, D)$$

Proof.

$$\begin{aligned}
Q &= (\Sigma, R, H, P) \\
x \in^n \sigma_\Sigma(cert_\perp(Q_*, D)) &\Rightarrow \sum_{z \in \{y | y \in cert_\perp(Q_*, D) \wedge \sigma_\Sigma(y) = x\}} cert_\perp(Q*, D)(z) \geq n \\
Moreover \\
\forall g, cert_\perp(Q_*, D) &\subseteq Eval(Q_*, g(D)) \\
Then \\
x \in^n \sigma_\Sigma(cert_\perp(Q_*, D)) &\Rightarrow \forall g, \sum_{z \in \{y | y \in Eval(Q_*, g(D)) \wedge \sigma_\Sigma(y) = x\}} Eval(Q_*, g(D))(z) \geq n \\
&\Rightarrow \forall g, Eval(Q, g(D))(x) \geq n \\
&\Rightarrow x \in^n cert_\perp(Q, D)
\end{aligned}$$

$$\begin{aligned}
Q &= (\Sigma, R, H, P) \\
x \notin \sigma_\Sigma(cert_\perp(Q_*, D)) &\Rightarrow \sum_{z \in \{y | y \in cert_\perp(Q_*, D) \wedge \sigma_\Sigma(y) = x\}} cert_\perp(Q_*, D)(z) = 0 \\
&\Rightarrow \forall y, \sigma_\Sigma(y) \neq x \vee cert_\perp(Q_*, D)(y) = 0 \\
&\Rightarrow \forall y, \sigma_\Sigma(y) \neq x \vee \exists v, v(y) \notin Eval_{SQL}(Q_*, v(D)) \\
&\Rightarrow \exists v, v(x) \notin \sigma_\Sigma(Eval_{SQL}(Q_*, v(D))) \\
&\Rightarrow \exists v, v(x) \notin Eval_{SQL}(Q, D) \\
&\Rightarrow x \notin cert_\perp(Q, D)
\end{aligned}$$

□

Proposition 2.

$$\forall Q \in [\![SQL]\!], Eval_{SQL}(Q^+, D) \subseteq cert_\perp(Q, D)$$

Proposition 3.

$$\forall Q \in [\![SQL]\!], cert_\perp(Q, D) \subseteq Eval_{SQL}(Q^?, D)$$

Proof. The proof of (2) is made by induction over the query Q assuming only (3).

We only detail critical case a more complete proof can be found in appendix.

$$
\begin{aligned}
&Q = (\Sigma, R, notexists(Q'), P) \\
&\quad x \in^n Eval_{SQL}(Q^+, D) \Rightarrow \sum_{z \in \{y | y \in Eval_{SQL}(Q^+_*, D) \wedge \sigma_\Sigma(y) = x\}} Eval_{SQL}(Q^+_*, D)(z) \geq n \\
&\quad Moreover \\
&\quad Eval_{SQL}(Q^+_*, D)(z) \geq k \Rightarrow Eval_{SQL}((*, R, notexists(Q')^*, P), D)(z) \geq k \\
&\quad \Rightarrow Eval_{SQL}((*, R, notexists(Q'^?), P), D)(z) \geq k \\
&\quad \Rightarrow R^\times(z) \geq k \wedge Eval_{SQL}(Q'[z]^?, D) = \emptyset \\
&\quad \Rightarrow R^\times(z) \geq k \wedge cert_\perp(Q'[z], D) = \emptyset \\
&\quad \Rightarrow R^\times(z) \geq k \wedge \forall h, \forall w, h(w) \notin Eval_{SQL}(Q'[h(z)], h(D)) \\
&\quad \Rightarrow R^\times(z) \geq k \wedge \forall h, Eval_{SQL}(Q'[h(z)], h(D)) = \emptyset \\
&\quad \Rightarrow R^\times(z) \geq k \wedge \forall h, h(z) \in Eval_{SQL}((*, R, notexists(Q'), h(P)), h(D)) \\
&\quad \Rightarrow cert_\perp(Q_*, D)(z) \geq k \\
&\quad Then \\
&\quad x \in^n Eval_{SQL}(Q^+, D) \Rightarrow \sum_{z \in \{y | y \in Eval_{SQL}(Q^+_*, D) \wedge \sigma_\Sigma(y) = x\}} cert_\perp(Q_*, D)(z) \geq n \\
&\quad \Rightarrow x \in^n cert_\perp(Q, D)
\end{aligned}
$$

□

Proof. The proof of (3) is made by induction over the query Q assuming only (2).
We only detail critical case a more complete proof can be found in appendix.

$$
\begin{aligned}
&Q = (\Sigma, R, notexists(Q'), P) \\
&\quad x \in^n cert_\perp(Q, D) \Rightarrow \sum_{z \in \{y | y \in cert_\perp(Q_*, D) \wedge \sigma_\Sigma(y) = x\}} cert_\perp(Q_*, D)(z) \geq n \\
&\quad Moreover \\
&\quad cert_\perp(Q_*, D)(z) \geq k \Rightarrow R^\times(z) \geq k \wedge \forall h, h(z) \in Eval_{SQL}((*, R, notexists(Q'), h(P)), h(D)) \\
&\quad \Rightarrow R^\times(z) \geq k \wedge \forall h, Eval_{SQL}(Q'[h(z)], h(D)) = \emptyset \\
&\quad \Rightarrow R^\times(z) \geq k \wedge cert_\perp(Q'[z], D) = \emptyset \\
&\quad \Rightarrow R^\times(z) \geq k \wedge Eval(Q'[z]^+, D) = \emptyset \\
&\quad \Rightarrow R^\times(z) \geq k \wedge z \in Eval_{SQL}((*, R, notexists(Q'^+), P), D) \\
&\quad \Rightarrow R^\times(z) \geq k \wedge z \in Eval_{SQL}((*, R, notexists(Q')^{**}, P), D) \\
&\quad \Rightarrow Eval_{SQL}((*, R, notexists(Q'), P)^?, D)(z) \geq k \\
&\quad Then \\
&\quad x \in^n cert_\perp(Q, D) \Rightarrow \sum_{z \in \{y | y \in posi_\perp(Q_*, D) \wedge \sigma_\Sigma(y) = x\}} Eval_{SQL}(Q^?_*, D)(z) \geq n \\
&\quad \Rightarrow x \in^n Eval_{SQL}(Q^?, D)
\end{aligned}
$$

□

We have proven that the evaluation algorithm such that we translate a query Q in Q^+ and then evaluate Q^+ with standard SQL semantics has *correctness guarantee*. Indeed we have even proven that such translation is valid as long as Q^+ computes a subset of certain answers and $Q^?$ computes at least all of them. It's easy to understand that work may be done in order to improve the over approximation of the certain answers without any loose.

3 Removing redundant null check

The evaluation of a query Q^+ which come from our translation may have a huge cost, as most DBMS such as Postgres do not behave well with disjunction. In our translation we introduce disjunctions however some of them might be redundant.

Definition. *For each Query and nested Query we maintain a set of attributes that have to be not null in order for the query to be true resp. false we denote this set* $\perp_Q^T$ *resp.*$\perp_Q^F$ *without taking null check in account.*

$$x \in Eval_{SQL}(Q_*, D) \Rightarrow \forall r_i.a_i \in \perp_Q^T, x[r_i.a_i] \neq \perp$$

$$x \notin Eval_{SQL}(Q_*, D) \Rightarrow \forall r_i.a_i \in \perp_Q^F, x[r_i.a_i] \neq \perp$$

(see appendix for a proper inductive definition, which show how to compute those).

Definition. *For a query Q we also compute the set of constraint which have to be true resp.false , in order for the tuples to be returned. We denote this set* $nested^+(Q)$ *resp.* $nested^-(Q)$.

$$x \in Eval_{SQL}(Q, D) \Rightarrow \forall c \in nested^+(Q), x \vdash c$$

$$x \in Eval_{SQL}(Q, D) \Rightarrow \forall c \in nested^-(Q), x \vdash \neg c$$

(see appendix for a proper inductive definition, which show how to compute those).

Now we can offer a translation $Q \rightarrow Q_Q^\perp$

Figure 6: Translatation $(H, Q) \rightarrow H_Q^\perp$

$$(\Sigma, R, H \vee null(r_i.a_i), P)_Q^\perp \rightarrow (\Sigma, R, H_Q^\perp, P)$$
$$\text{if } \exists Q' \in nested^+(Q), (H \vee null(r_i.a_i)) \in nested(Q'), r_i.a_i \in \perp_{Q'}^T$$
$$\text{if } \exists Q' \in nested^-(Q), (H \vee null(r_i.a_i)) \in nested(Q'), r_i.a_i \in \perp_{Q'}^F$$
$$(\Sigma, R, H \vee null(p_i), P)_Q^\perp \rightarrow (\Sigma, R, H_Q^\perp, P)$$
$$\text{if } \exists Q' \in nested^+(Q), (H \vee null(p_i)) \in nested(Q'), p_i \in \perp_{Q'}^T$$
$$\text{if } \exists Q' \in nested^-(Q), (H \vee null(p_i)) \in nested(Q'), p_i \in \perp_{Q'}^F$$
$$(\Sigma, R, H \wedge const(r_i.a_i), P)_Q^\perp \rightarrow (\Sigma, R, H_Q^\perp, P)$$
$$\text{if } \exists Q' \in nested^+(Q), (H \wedge const(r_i.a_i)) \in nested(Q'), r_i.a_i \in \perp_{Q'}^T$$
$$\text{if } \exists Q' \in nested^-(Q), (H \wedge const(r_i.a_i)) \in nested(Q'), r_i.a_i \in \perp_{Q'}^F$$
$$(\Sigma, R, H \wedge const(p_i), P)_Q^\perp \rightarrow (\Sigma, R, H_Q^\perp, P)$$
$$\text{if } \exists Q' \in nested^+(Q), (H \wedge const(p_i)) \in nested(Q'), p_i \in \perp_{Q'}^T$$
$$\text{if } \exists Q' \in nested^-(Q), (H \wedge const(p_i)) \in nested(Q'), p_i \in \perp_{Q'}^F$$

To give an intuition to understand why such translation preserve the evaluation, one have to understand that in order for a tuple to be return each sub-conditions have to be fulfill. Then if some part of the query require an attribute to be a constant, we know that this attribute can not be null in the whole query. Then we are allowed to remove any null test as they can not be involved in the computation of a valid tuple. (Actually it is a bit more complicated due to the presence of disjunction but that's the main idea).

Proposition 4.

$$\forall Q \in [\![SQL]\!]_\perp Eval_{SQL}(Q, D) = Eval_{SQL}(Q_Q^\perp, D)$$

Proof. In order to prove the property we have to introduce an analogue query translation when we assume that the query is false. Such translations would verify:

Lemma 2.

$$\forall Q, \forall Q' \in nested^-(Q), Eval(Q, D) = Eval(Q_{Q'}^{\perp^F}, D)$$

Lemma 3.

$$\forall Q, \forall Q' \in nested^+(Q), Eval(Q, D) = Eval(Q_{Q'}^\perp, D)$$

Then we can prove inductively that assuming 2, 3 holds and assuming 3, 2 holds. (see appendix for a proper inductive definitions,and a full proof). Then as prop 4 is only a restriction of lemma 2 it also holds. □

This rewriting has prove to be efficient, as it reduce the number of disjunction is the query moreover it allow us to take care of SQL constraint such as NOT NULL attributes, indeed if we add those attributes to $\perp_Q^T$ every null check on those attributes will be remove.

4 Query Optimization

In this section we offer rewriting methods that help the planner computing disjunctive query in a more efficient way. Such optimization rely on already existing DBMS like Postgres, those DBMS does not implement a marked nulls version yet, that is why we explain how our results can be used in the case of SQL nulls.

4.1 SQL null

The semantic usually used to deal with SQL nulls is that there should be no distinction when applying a marking function on the database nulls, and when applying a marking function on the result of a query. Formally, let M be a marking function which mark each null value with a different stamp. Then there should exist an isomorphism between : $Q(M(D))$ and $M(Q(D))$. However it's not always the case if we consider the SQL-fragment we offer in our syntax.

Moreover as mention before the property : Let R be a set of relation, and z a tuple then:

$$(\forall v \text{ valuation}, v(R^\times)(v(z)) \geq k) \Leftrightarrow R^\times(z) \geq k$$

does not hold. Indeed Consider a relation $r = [\![\perp; \perp]\!]$ then $R^\times(\perp) \geq 2$ where as if $v(r) = [\![2; 3]\!]$ then $v(r)(v(\perp)) < 2$ Each SQL nulls can be evaluated differently by the valuation v. Then certain answers with nulls of a query $Q = (*, r, TRUE, \emptyset)$ would be $[\![\perp]\!]$ which does not seem to be satisfactory. A proper definition has not been found yet, and we are currently working on it.

We will only consider a fragment where the isomorphism holds. And we will compute the certain answers as if nulls in the answers where marked nulls, and that the multiplicity of them where the sum of each marked nulls. Just a few change to the translation (details in the appendix) allow us to have a valid evaluation algorithm.

4.2 Split the query

The main blow-up in computation time come from the fact that most DBMS give up using hash-join, as soon as they meet a disjunction, and choose to use nested-loop. In order to force the planner into using hash-join the way they should, we split the query in the various form it may have due to disjunction. Formally, the negative sub queries's conditions of a query Q are put in DNF, and each part are evaluate separately. We are aware that such algorithm have a exponential complexity however, it work well in practice.

Proposition 5. *Let* $Q = (\Sigma, R, notexists((\Sigma', R', (l_1 \vee l_2) \wedge H, P'), P)$

$Eval_{SQL}(Q, D) = Eval_{SQL}((\Sigma, R, notexists(\Sigma', R', (l_1 \wedge H), P') \wedge notexists(\Sigma', R', (l_2 \wedge H), P'), P), D)$

4.3 Optimize FROM relations

We can avoid computing the whole bag $R'^{\times}$. First we build the set of relation that are not linked to an upper-level query by any of there attributes denotes R'_s. Then we can rewrite the query Q'.

Proposition 6. *Let* $Q = (\Sigma, R, notexists((\Sigma', R', H_r \wedge H, P'), P)$ *where* H_r *denote every conditions which are on attributes of* r *and such that* $r \in R'_s$ *then :*

$$Eval_{SQL}(Q, D) = Eval_{SQL}((\Sigma, R, notexists(\Sigma', R' \setminus r, H \wedge exists(*, r, H_r, P'), P'), P), D)$$

We have to add an exists sub-query in order to check if the selection over the relation r is not empty. Indeed if the selection is empty then the Cartesian product $R'\times$ will be empty.

This methods help the DBMS because, it won't have to work on rows that are useless. Indeed as we are in a not exists condition, we only care about having one represent for each tuple that might be linked with the upper-level query.

4.4 Improve computation

The methods propose in this sub-section is the most efficient, when the tuples compute fit in the memory. Indeed it offers the fastest way to compute the query $Q^?$ however, it really computes each of them and have to store them. As soon as the Database involved does not fit in the ram memory, this method will be really expensive as each computation will involve a slow write on the hard drive. We won't be able to present the property formally here as it uses a fragment of SQL which is not supported in our syntax ie. (UNION ALL). The idea is to replace disjunctive query with UNION ALL, as soon as $H = l_1 \vee l_2$ verify the property such that:
$(l_1 \implies \neg l_2) \wedge (l_2 \implies \neg l_1)$ then we have :

$$Eval_{SQL}((\Sigma, R, D, P), D) = Eval_{SQL}((\Sigma, R, l_1, P), D) \text{ UNION ALL } Eval_{SQL}((\Sigma, R, l_2, P), D)$$

5 Results

Every result presented here are based on TCP-H database instance, in this report we present a simplified query coming from TCP-H. TCP-H database schema guarantee that $o\_orderkey, p\_partkey, l\_orderkey$ are not null attributes. Time were obtain on Postgres DBMS on a 10 Go instance with 0.5% of nulls randomly spread among each null-able attributes. Each step of the rewriting where done thanks to our Postgres Extension, Queries obtain after each steps can be found in appendix.

Figure 7: Initial SQL Query

```
SELECT o_orderkey
FROM orders
WHERE NOT EXISTS(
        SELECT *
        FROM lineitem ,part
        WHERE l_orderkey = o_orderkey
        AND l_partkey = p_partkey);
```

Figure 8: Query after Split and optimize FROM

```sql
SELECT o_orderkey
FROM orders
WHERE NOT EXISTS(
        SELECT *
        FROM lineitem ,part
        WHERE l_orderkey = o_orderkey
        AND l_partkey = p_partkey)
AND NOT EXISTS(
        SELECT *
        FROM lineitem
        WHERE l_orderkey = o_orderkey
        AND EXISTS(SELECT * FROM part WHERE l_partkey IS NULL));
```

After the applying of the translation we obtain a query without any optimization, when trying to evaluate this query PSQL estimate time is 10^3 times longer than for Q. Moreover when we actually run it, it seems to be true as we never manage to get to the end on big instances (nested-loop). The result is exactly the same after we remove redundant null checks, this result is explain by the fact that the planner try to compute the result in the exact same way because there is still disjunctions (nested-loop). As soon as we split the query, PSQL manage to compute the query and it take twice the time that we need to compute the initial query, it's easily understandable as we split the query in 2 different not exists (2 $\times$ Hash-Join). Finally on more complicated queries which might split in multiple not-exists queries, we try to mix UNION ALL and split. In worst case the computation takes 6 times longer than the computation of the initial query. However we did not have enough time to implement an heuristic to choose when to use UNION-ALL and when to split, this heuristic should use meta-data such as null-percent by attributes in order to predict if the tuples will fit the memory.

The result present before are not as reliable as they should be, they were produce on a restraint number of queries, and just a few different instances of TCP-H Database. Moreover PSQL was running on my personal computer which is not that stable. Proper benchmarks will be generated during next week, and will be presented during the defense.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[2] Serge Abiteboul, Paris Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78(1):159–187, 1991.

[3] Chris Date. *Database in depth: relational theory for practitioners*. " O'Reilly Media, Inc.", 2005.

[4] Amélie Gheerbrant, Leonid Libkin, and Cristina Sirangelo. Naïve evaluation of queries over incomplete databases. *ACM Transactions on Database Systems (TODS)*, 39(4):31, 2014.

[5] Paolo Guagliardo, Leonid Libkin, Leonid Libkin, Wim Martens, Domagoj Vrgoč, Marco Console, Pablo Barcelo, Diego Figueira, Amélie Gheerbrant, Cristina Sirangelo, et al. Making sql queries correct on incomplete databases: A feasibility study. *Journal of the ACM*, 63:2, 2016.

[6] Tomasz Imieliński and Witold Lipski Jr. Incomplete information in relational databases. *Journal of the ACM (JACM)*, 31(4):761–791, 1984.

[7] Leonid Libkin. Sql's three-valued logic and certain answers. *ACM Transactions on Database Systems (TODS)*, 41(1):1, 2016.

[8] Ron van der Meyden. Logical approaches to incomplete information: A survey. In *Logics for databases and information systems*, pages 307–356. Springer, 1998.

A Appendix

A.1 Bags

Definition. *We call a bag B a function $D \to \mathbb{N}$ such that $B(x)$ represents the multiplicity of x in the bag B.*

Definition.

$$
\begin{aligned}
\forall x, \emptyset(x) &= 0 \\
\forall x, (B_1 \cap B_2)(x) &= min(B_1(x), B_2(x)) \\
\forall x, (B_1 \cup B_2)(x) &= max(B_1(x), B_2(x)) \\
\forall x, (B_1 \uplus B_2)(x) &= B_1(x) + B_2(x) \\
\forall x, (B_1 \setminus B_2)(x) &= max(0, B_1(x) - B_2(x)) \\
\forall x, [\![a]\!](x) &= \begin{cases} 1 \textit{ if } x = a \\ 0 \textit{ otherwise} \end{cases} \\
\forall x, [\![a^n]\!](x) &= \begin{cases} n \textit{ if } x = a \\ 0 \textit{ otherwise} \end{cases} \\
\forall x, [\![y^n | P(y, n)]\!](x) &= max(\{i | P(x, i)\}) \\
x \in B &\iff B(x) \geq 1 \\
x \in^n B &\iff B(x) \geq n \\
x \notin B &\iff B(x) = 0 \\
B_1 = B_2 &\iff \forall x, B_1(x) = B_2(x) \\
B_1 \subseteq B_2 &\iff \forall x, B_1(x) \leq B_2(x) \\
\{B\} &= \{x | B(x) \geq 1\}
\end{aligned}
$$

A.2 Inductive definition

In this section we develop the definition, and building of the sets used in section 3.

Definition.

$$
\begin{aligned}
\perp^T_{(\Sigma,R,H_1\wedge H_2,P)} &= \perp^T_{(\Sigma,R,H_1,P)} \cup \perp^T_{(\Sigma,R,H_2,P)}\\
\perp^T_{(\Sigma,R,H_1\vee H_2,P)} &= \perp^T_{(\Sigma,R,H_1,P)} \cap \perp^T_{(\Sigma,R,H_2,P)}\\
\perp^T_{(\Sigma,R,r_i.a_i=c_i,P)} &= \{r_i.a_i\}\\
\perp^T_{(\Sigma,R,r_i.a_i\neq c_i,P)} &= \{r_i.a_i\}\\
\perp^T_{(\Sigma,R,r_i.a_i>c_i,P)} &= \{r_i.a_i\}\\
\perp^T_{(\Sigma,R,r_i.a_i<c_i,P)} &= \{r_i.a_i\}\\
\perp^T_{(\Sigma,R,r_i.a_i=r_j.a_j,P)} &= \emptyset\\
\perp^T_{(\Sigma,R,r_i.a_i\neq r_j.a_j,P)} &= \{r_i.a_i, r_j.a_j\}\\
\perp^T_{(\Sigma,R,r_i.a_i>r_j.a_j,P)} &= \emptyset\\
\perp^T_{(\Sigma,R,r_i.a_i<r_j.a_j,P)} &= \emptyset\\
\perp^T_{(\Sigma,R,r_i.a_i=p_i,P)} &= \emptyset\\
\perp^T_{(\Sigma,R,r_i.a_i\neq p_i,P)} &= \{r_i.a_i, p_i\}\\
\perp^T_{(\Sigma,R,r_i.a_i>p_i,P)} &= \emptyset\\
\perp^T_{(\Sigma,R,r_i.a_i<p_i,P)} &= \emptyset\\
\perp^T_{(\Sigma,R,exists(Q),P)} &= \perp^T_{Q[?]}\\
\perp^T_{(\Sigma,R,notexists(Q),P)} &= \perp^F_{Q[?]}
\end{aligned}
$$

$$
\begin{aligned}
\perp^F_{(\Sigma,R,H_1\wedge H_2,P)} &= \perp^F_{(\Sigma,R,H_1,P)} \cap \perp^F_{(\Sigma,R,H_2,P)}\\
\perp^F_{(\Sigma,R,H_1\vee H_2,P)} &= \perp^F_{(\Sigma,R,H_1,P)} \cup \perp^F_{(\Sigma,R,H_2,P)}\\
\perp^F_{(\Sigma,R,r_i.a_i=c_i,P)} &= \emptyset\\
\perp^F_{(\Sigma,R,r_i.a_i\neq c_i,P)} &= \emptyset\\
\perp^F_{(\Sigma,R,r_i.a_i>c_i,P)} &= \emptyset\\
\perp^F_{(\Sigma,R,r_i.a_i<c_i,P)} &= \emptyset\\
\perp^F_{(\Sigma,R,r_i.a_i=r_j.a_j,P)} &= \emptyset\\
\perp^F_{(\Sigma,R,r_i.a_i\neq r_j.a_j,P)} &= \emptyset\\
\perp^F_{(\Sigma,R,r_i.a_i>r_j.a_j,P)} &= \emptyset\\
\perp^F_{(\Sigma,R,r_i.a_i<r_j.a_j,P)} &= \emptyset\\
\perp^F_{(\Sigma,R,r_i.a_i=p_i,P)} &= \emptyset\\
\perp^F_{(\Sigma,R,r_i.a_i\neq p_i,P)} &= \emptyset\\
\perp^F_{(\Sigma,R,r_i.a_i>p_i,P)} &= \emptyset\\
\perp^F_{(\Sigma,R,r_i.a_i<p_i,P)} &= \emptyset\\
\perp^F_{(\Sigma,R,exists(Q),P)} &= \perp^F_{Q[?]}\\
\perp^F_{(\Sigma,R,notexists(Q),P)} &= \perp^T_{Q[?]}
\end{aligned}
$$

Definition.

$$\begin{aligned}
nested^+(H_1 \wedge H_2) &= \{H_1 \wedge H_2\} \cup nested^+(H_1) \cup nested^+(H_2)\\
nested^+(H_1 \vee H_2) &= \{H_1 \vee H_2\} \cup nested^+(H_1) \cup nested^+(H_2)\\
nested^+(r_i.a_i = c_i) &= \{r_i.a_i = c_i\}\\
nested^+(r_i.a_i \neq c_i) &= \{r_i.a_i \neq c_i\}\\
nested^+(r_i.a_i > c_i) &= \{r_i.a_i > c_i\}\\
nested^+(r_i.a_i < c_i) &= \{r_i.a_i < c_i\}\\
nested^+(r_i.a_i = r_j.a_j) &= \{r_i.a_i = r_j.a_j\}\\
nested^+(r_i.a_i \neq r_j.a_j) &= \{r_i.a_i \neq r_j.a_j\}\\
nested^+(r_i.a_i > r_j.a_j) &= \{r_i.a_i > r_j.a_j\}\\
nested^+(r_i.a_i < r_j.a_j) &= \{r_i.a_i < r_j.a_j\}\\
nested^+(r_i.a_i = p_j) &= \{r_i.a_i = p_j\}\\
nested^+(r_i.a_i \neq p_j) &= \{r_i.a_i \neq p_j\}\\
nested^+(r_i.a_i > p_j) &= \{r_i.a_i > p_j\}\\
nested^+(r_i.a_i < p_j) &= \{r_i.a_i < p_j\}\\
nested^+(exists(Q)) &= \{exists(Q)\} \cup nested^+(Q)\\
nested^+(notexists(Q)) &= \{notexists(Q)\} \cup nested^-(Q)
\end{aligned}$$

$$\begin{aligned}
nested^-(H_1 \wedge H_2) &= nested^-(H_1) \cup nested^-(H_2)\\
nested^-(H_1 \vee H_2) &= nested^-(H_1) \cup nested^-(H_2)\\
nested^-(r_i.a_i = c_i) &= \emptyset\\
nested^-(r_i.a_i \neq c_i) &= \emptyset\\
nested^-(r_i.a_i > c_i) &= \emptyset\\
nested^-(r_i.a_i < c_i) &= \emptyset\\
nested^-(r_i.a_i = r_j.a_j) &= \emptyset\\
nested^-(r_i.a_i \neq r_j.a_j) &= \emptyset\\
nested^-(r_i.a_i > r_j.a_j) &= \emptyset\\
nested^-(r_i.a_i < r_j.a_j) &= \emptyset\\
nested^-(r_i.a_i = p_j) &= \emptyset\\
nested^-(r_i.a_i \neq p_j) &= \emptyset\\
nested^-(r_i.a_i > p_j) &= \emptyset\\
nested^-(r_i.a_i < p_j) &= \emptyset\\
nested^-(exists(Q)) &= nested^-(Q)\\
nested^-(notexists(Q)) &= nested^+(Q)
\end{aligned}$$

$$nested(Q) = nested^-(Q) \cup nested^+(Q)$$

We do not prove that those inductive definitions verify the property needed in section , however such a proof is fairly straight forward with an induction.

A.3 Proof of proposition 2

Proposition.

$$\forall Q \in [\![SQL]\!], Eval_{SQL}(Q^+, D) \subseteq cert_\perp(Q, D)$$

Proposition.

$$\forall Q \in [\![SQL]\!], cert_\perp(Q, D) \subseteq Eval_{SQL}(Q^?, D)$$

Proof. The proof of prop 2 is made by induction over the query Q assuming only prop 4.

$$\begin{aligned}
Q = (\Sigma, R, \emptyset, P)& \\
x \in^n Eval_{SQL}(Q^+, D) &\Rightarrow x \in^n \sigma_\Sigma(R^\times) \\
&\Rightarrow \forall v, v(x) \in^n \sigma_\Sigma(v(R)^\times) \text{ by } 1 \\
&\Rightarrow \forall v, v(x) \in^n \sigma_\Sigma(Eval_{SQL}(Q_*, v(D))) \\
&\Rightarrow x \in^n cert_\perp(Q, D)
\end{aligned}$$

$$\begin{aligned}
Q = (\Sigma, R, r_i.a_i \neq p_i, P)& \\
x \in^n Eval_{SQL}(Q^+, D) &\Rightarrow \sum_{z \in \{y \mid y \in Eval_{SQL}(Q^+_*, D) \wedge \sigma_\Sigma(y) = x\}} Eval_{SQL}(Q^+_*, D)(z) \geq n \\
Moreover& \\
Eval_{SQL}(Q^+_*, D)(z) \geq k &\Rightarrow Eval_{SQL}((*, R, (r_i.a_i \neq p_i)^*, P), D)(z) = k \\
&\Rightarrow Eval_{SQL}((*, R, r_i.a_i \neq p_i \wedge const(r_i, a_i) \wedge const(p_i), P), D)(z) = k \\
&\Rightarrow R^\times(x) \geq \wedge x[r_i.a_i] \neq P[p_i] \wedge \forall t, x[r_i.a_i] \neq \perp_t \wedge \forall t, P[p_i] \neq \perp_t \\
Moreover& \\
\forall t, x[r_i.a_i] \neq \perp_t \wedge \forall t, P[p_i] \neq \perp_t &\Rightarrow \forall h, h(x)[r_i.a_i] = x[r_i.a_i] \wedge h(P)[p_i] = P[p_i] \\
Then& \\
Eval_{SQL}(Q^+_*, D)(z) \geq k &\Rightarrow R^\times(x) \geq k \wedge \forall h, h(x)[r_i.a_i] \neq h(P)[p_i] \\
&\Rightarrow cert_\perp(Q_*, D)(z) \geq k \\
Then& \\
x \in^n Eval_{SQL}(Q^+, D) &\Rightarrow \sum_{z \in \{y \mid y \in Eval_{SQL}(Q^+_*, D) \wedge \sigma_\Sigma(y) = x\}} cert_\perp(Q_*, D)(z) \geq n \\
&\Rightarrow x \in^n cert_\perp(Q, D)
\end{aligned}$$

$$
\begin{aligned}
Q &= (\Sigma, R, H_1 \wedge H_2, P) \\
x \in^n Eval_{SQL}(Q^+, D) &\Rightarrow \sum_{z \in \{y | y \in Eval_{SQL}(Q_*^+, D) \wedge \sigma_\Sigma(y) = x\}} Eval_{SQL}(Q_*^+, D)(z) \geq n \\
Moreover & \\
Eval_{SQL}(Q_*^+, D)(z) \geq k &\Rightarrow Eval_{SQL}((*, R, H_1^* \wedge H_2^*, P), D)(z) \geq k \\
&\Rightarrow (Eval_{SQL}((*, R, H_1^*, P), D) \cap Eval_{SQL}((*, R, H_2^*, P), D))(z) \geq k \\
&\Rightarrow (Eval_{SQL}((*, R, H_1, P)^+, D) \cap Eval_{SQL}((*, R, H_2, P)^+, D))(z) \geq k \\
&\Rightarrow (Eval_{SQL}((*, R, H_1, P)^+, D))(x) \geq k \wedge (Eval_{SQL}((*, R, H_2, P)^+, D))(z) \geq k \\
&\Rightarrow (cert_\perp((*, R, H_1, P), D))(x) \geq k \wedge (cert_\perp((*, R, H_2, P), D))(z) \geq k \\
&\Rightarrow (cert_\perp((*, R, H_1, P), D) \cap cert_\perp((*, R, H_2, P), D))(z) \geq k \\
&\Rightarrow (cert_\perp((*, R, H_1 \wedge H_2, P), D))(z) \geq k \\
&\Rightarrow (cert_\perp(Q_*, D))(z) \geq k \\
Then & \\
x \in^n Eval_{SQL}(Q^+, D) &\Rightarrow \sum_{z \in \{y | y \in Eval_{SQL}(Q_*^+, D) \wedge \sigma_\Sigma(y) = x\}} cert_\perp(Q_*, D)(z) \geq n \\
&\Rightarrow x \in^n cert_\perp(Q, D)
\end{aligned}
$$

$$
\begin{aligned}
Q &= (\Sigma, R, H_1 \vee H_2, P) \\
x \in^n Eval_{SQL}(Q^+, D) &\Rightarrow \sum_{z \in \{y | y \in Eval_{SQL}(Q_*^+, D) \wedge \sigma_\Sigma(y) = x\}} Eval_{SQL}(Q_*^+, D)(z) \geq n \\
Moreover & \\
Eval_{SQL}(Q_*^+, D)(z) \geq k &\Rightarrow Eval_{SQL}((*, R, H_1^* \vee H_2^*, P), D)(z) \geq k \\
&\Rightarrow (Eval_{SQL}((*, R, H_1^*, P), D) \cup Eval_{SQL}((*, R, H_2^*, P), D))(z) \geq k \\
&\Rightarrow (Eval_{SQL}((*, R, H_1, P)^+, D) \cup Eval_{SQL}((*, R, H_2, P)^+, D))(z) \geq k \\
&\Rightarrow (Eval_{SQL}((*, R, H_1, P)^+, D))(x) \geq k \vee (Eval_{SQL}((*, R, H_2, P)^+, D))(z) \geq k \\
&\Rightarrow (cert_\perp((*, R, H_1, P), D))(x) \geq k \vee (cert_\perp((*, R, H_2, P), D))(z) \geq k \\
&\Rightarrow (cert_\perp((*, R, H_1, P), D) \cup cert_\perp((*, R, H_2, P), D))(z) \geq k \\
&\Rightarrow (cert_\perp((*, R, H_1 \vee H_2, P), D))(z) \geq k \\
&\Rightarrow (cert_\perp(Q_*, D))(z) \geq k \\
Then & \\
x \in^n Eval_{SQL}(Q^+, D) &\Rightarrow \sum_{z \in \{y | y \in Eval_{SQL}(Q_*^+, D) \wedge \sigma_\Sigma(y) = x\}} cert_\perp(Q_*, D)(z) \geq n \\
&\Rightarrow x \in^n cert_\perp(Q, D)
\end{aligned}
$$

$\square$

Proof. The proof of prop 4 is made by induction over the query Q assuming only prop 2.

$$\begin{aligned}
&Q = (\Sigma, R, \emptyset, P)\\
x \in^n cert_\perp(Q, D) &\Rightarrow \forall v, v(x) \in^n Eval_{SQL}(Q, v(D))\\
&\Rightarrow \forall v, v(x) \in^n \sigma_\Sigma(Eval_{SQL}(Q_*, v(D)))\\
&\Rightarrow \forall v, v(x) \in^n \sigma_\Sigma(v(R)^\times)\\
&\Rightarrow x \in^n \sigma_\Sigma(R^\times) \text{ by } 1\\
&\Rightarrow x \in^n Eval_{SQL}(Q^?, D)
\end{aligned}$$

$$\begin{aligned}
Q = (\Sigma, R, r_i.a_i = r_j.a_j, P)&\\
x \in^n cert_\perp(Q, D) &\Rightarrow \sum_{z \in \{y | y \in cert_\perp(Q_*, D) \wedge \sigma_\Sigma(y) = x\}} cert_\perp(Q_*, D)(z) \geq n\\
Moreover&\\
cert_\perp(Q_*, D)(z) \geq k &\Rightarrow \forall h, h(z) \in^k Eval_{SQL}(Q_*, h(D))\\
&\Rightarrow \forall h, \forall t, R^\times(z) \geq k \wedge h(z)[r_i.a_i] = h(z)[r_j.a_j] \wedge h(z)[r_i.a_i] \neq \perp_t \wedge h(z)[r_j.a_j] \neq \perp_t\\
Moreover&\\
\text{TRUE} &\Rightarrow \forall g, \forall t, g(z)[r_i.a_i] \neq \perp_t\\
\forall h, h(z)[r_i.a_i] = h(z)[r_j.a_j] &\Rightarrow \exists h, h(z)[r_i.a_i] = h(z)[r_j.a_j]\\
&\Rightarrow z[r_i.a_i] = z[r_j.a_j] \vee (\exists t, z[r_i.a_i] = \perp_t) \vee (\exists t, z[r_j.a_j] = \perp_t)\\
Then&\\
cert_\perp(Q_*, D)(z) \geq k &\Rightarrow R^\times(z) \geq k \wedge z[r_i.a_i] = z[r_j.a_j] \vee (\exists t, z[r_i.a_i] = \perp_t) \vee (\exists t, z[r_j.a_j] = \perp_t)\\
&\Rightarrow R^\times(z) \geq k \wedge z \in Eval_{SQL}((*, R, (r_i.a_i = r_j.a_j \vee null(r_i.a_i) \vee null(r_j.a_j)), P), D)\\
&\Rightarrow R^\times(z) \geq k \wedge z \in Eval_{SQL}((*, R, (r_i.a_i = r_j.a_j)^{**}, P), D)\\
&\Rightarrow Eval_{SQL}((*, R, r_i.a_i = r_j.a_j, P)^?, D)(z) \geq k\\
Then&\\
x \in^n cert_\perp(Q, D) &\Rightarrow \sum_{z \in \{y | y \in cert_\perp(Q_*, D) \wedge \sigma_\Sigma(y) = x\}} Eval_{SQL}(Q_*^?, D)(z) \geq n\\
&\Rightarrow x \in^n Eval_{SQL}(Q^?, D)
\end{aligned}$$

$$
\begin{aligned}
Q = (\Sigma, R, H_1 \wedge H_2, P) & \\
x \in^n cert_{\perp}(Q, D) &\Rightarrow \sum_{z \in \{y | y \in cert\perp(Q_*, D) \wedge \sigma_{\Sigma}(y) = x\}} cert_{\perp}(Q_*, D)(z) \geq n \\
Moreover & \\
cert_{\perp}(Q_*, D)(z) \geq k &\Rightarrow R^{\times}(z) \geq k \wedge \forall h, h(z) \in Eval_{SQL}((*, R, H_1 \wedge H_2, h(P)), h(D)) \\
&\Rightarrow R^{\times}(z) \geq k \wedge \forall h, h(z) \in Eval_{SQL}((*, R, H_1, h(P)), h(D)) \wedge h(z) \in Eval_{SQL}((*, R, H_1, h(P)), h \\
&\Rightarrow cert_{\perp}((*, R, H_1, P), D)(z) \geq k \wedge cert_{\perp}((*, R, H_2, P), D)(z) \geq k \\
&\Rightarrow Eval_{SQL}((*, R, H_1, P)^?, D)(z) \geq k \wedge Eval_{SQL}((*, R, H_2, P)^?, D)(z) \geq k \\
&\Rightarrow Eval_{SQL}((*, R, H_1^{**}, P), D)(z) \geq k \wedge Eval_{SQL}((*, R, H_2^{**}, P), D)(z) \geq k \\
&\Rightarrow Eval_{SQL}((*, R, H_1^{**} \wedge H_2^{**}, P), D)(z) \geq k \\
&\Rightarrow Eval_{SQL}((*, R, H_1 \wedge H_2, P)^?, D)(z) \geq k \\
&\Rightarrow Eval_{SQL}(Q_*^?, D)(z) \geq k \\
Then & \\
x \in^n cert_{\perp}(Q, D) &\Rightarrow \sum_{z \in \{y | y \in posi_{\perp}(Q_*, D) \wedge \sigma_{\Sigma}(y) = x\}} Eval_{SQL}(Q_*^?, D)(z) \geq n \\
&\Rightarrow x \in^n Eval_{SQL}(Q^?, D)
\end{aligned}
$$

□

A.4 Proof of proposition 4

Figure 9: Translation $(H, Q) \to H_Q^{\perp^F}$

$$
\begin{aligned}
(\Sigma, R, H \vee null(r_i.a_i), P)_Q^{\perp^F} &\to (\Sigma, R, H_Q^{\perp^F}, P) \\
&\text{if } \exists Q' \in nested^-(Q), (H \vee null(r_i.a_i)) \in nested(Q'), r_i.a_i \in \perp_{Q'}^T \\
&\text{if } \exists Q' \in nested^+(Q), (H \vee null(r_i.a_i)) \in nested(Q'), r_i.a_i \in \perp_{Q'}^F \\
(\Sigma, R, H \vee null(p_i), P)_Q^{\perp^F} &\to (\Sigma, R, H_Q^{\perp^F}, P) \\
&\text{if } \exists Q' \in nested^-(Q), (H \vee null(p_i)) \in nested(Q'), p_i \in \perp_{Q'}^T \\
&\text{if } \exists Q' \in nested^+(Q), (H \vee null(p_i)) \in nested(Q'), p_i \in \perp_{Q'}^F \\
(\Sigma, R, H \wedge const(r_i.a_i), P)_Q^{\perp^F} &\to (\Sigma, R, H_Q^{\perp^F}, P) \\
&\text{if } \exists Q' \in nested^-(Q), (H \wedge const(r_i.a_i)) \in nested(Q'), r_i.a_i \in \perp_{Q'}^T \\
&\text{if } \exists Q' \in nested^+(Q), (H \wedge const(r_i.a_i)) \in nested(Q'), r_i.a_i \in \perp_{Q'}^F \\
(\Sigma, R, H \wedge const(p_i), P)_Q^{\perp^F} &\to (\Sigma, R, H_Q^{\perp^F}, P) \\
&\text{if } \exists Q' \in nested^-(Q), (H \wedge const(p_i)) \in nested(Q'), p_i \in \perp_{Q'}^T \\
&\text{if } \exists Q' \in nested^+(Q), (H \wedge const(p_i)) \in nested(Q'), p_i \in \perp_{Q'}^F
\end{aligned}
$$

Lemma.

$$\forall Q' \in nested^+(Q), Eval(Q, D) = Eval(Q_{Q'}^{\perp}, D)$$

Lemma.
$$\forall Q' \in nested^{-}(Q), Eval(Q,D) = Eval(Q^{\perp^F}_{Q'}, D)$$

Proof. Let suppose $H_1 \wedge H_2 \in nested^{+}(Q)$.
The critical case is when $\exists r_i.a_i \in (\perp^T_{H_1} \setminus \perp^T_{H_2})$ and $(H \vee null(r_i.a_i) \in nested(H_2)$.
Let's assume there exists x such that $x \in Eval_{SQL}(Q,D)$ and $x \notin Eval(Q^{\perp}_{H_1 \wedge H_2}, D)$. Moreover we know by induction that $x \in Eval(Q^{\perp}_{H_1}, D) \wedge x \in Eval(Q^{\perp}_{H_2}, D)$. Then if $x[r_i.a_i] = \perp$ we know that $x \notin Eval_{SQL}(Q,D)$ as $x \in \perp^T_{H_1}$ and $H_1 \in nested^{+}(Q)$. So $x[r_i.a_i] \neq \perp$ then the rewriting is correct.
Let's assume there exists x such that $x \notin Eval_{SQL}(Q,D)$ and $x \in Eval(Q^{\perp}_{H_1 \wedge H_2}, D)$. Moreover we know by induction that $x \in Eval(Q^{\perp}_{H_1}, D) \wedge x \in Eval(Q^{\perp}_{H_2}, D)$. Then if $x[r_i.a_i] = \perp$ we know that $x \notin Eval_{SQL}(H_1, D)$ as $x \in \perp^T_{H_1}$. So $\forall H, x \notin Eval_{SQL}(H_1 \wedge H, D)$ especially $Eval(Q^{\perp}_{H_1 \wedge H_2}, D)$. Then $x[r_i.a_i] \neq \perp$ then the rewriting is correct.

Let suppose $H_1 \vee H_2 \in nested^{-}(Q)$.
The critical case is when $\exists r_i.a_i \in (\perp^F_{H_1} \setminus \perp^F_{H_2})$ and $(H \vee null(r_i.a_i) \in nested(H_2)$.
Let's assume there exists x such that $x \in Eval_{SQL}(Q,D)$ and $x \notin Eval(Q^{\perp}_{H_1 \vee H_2}, D)$. Morover we know by induction that $x \in Eval(Q^{\perp}_{H_1}, D) \wedge x \in Eval(Q^{\perp}_{H_2}, D)$. Then if $x[r_i.a_i] = \perp$ we know that $x \notin Eval_{SQL}(Q,D)$ as $x \in \perp^F_{H_1}$ and $H_1 \in nested^{-}(Q)$. So $x[r_i.a_i] \neq \perp$ then the rewriting is correct.
Let's assume there exists x such that $x \notin Eval_{SQL}(Q,D)$ and $x \in Eval(Q^{\perp}_{H_1 \vee H_2}, D)$. Morover we know by induction that $x \in Eval(Q^{\perp}_{H_1}, D) \wedge x \in Eval(Q^{\perp}_{H_2}, D)$. Then if $x[r_i.a_i] = \perp$ we know that $x \in Eval_{SQL}(H_1, D)$ as $x \in \perp^F_{H_1}$. So $\forall H, x \in Eval_{SQL}(H_1 \vee H, D)$ especially $Eval_{SQL}(Q^{\perp}_{H_1 \vee H_2}, D)$. Then $x[r_i.a_i] \neq \perp$ then the rewriting is correct.

Other case are trivial as, $\perp^T$ resp.$\perp^F$ only increase with this$\wedge$ resp $\vee$. In case of negation we only have to assume that lemma 3 is true.
The proof for lemma 3 is analogue □

A.5 Proof of proposition 5

Proposition. *Let* $Q = (\Sigma, R, notexists((\Sigma', R', (l_1 \vee l_2) \wedge H, P'), P)$

$$Eval_{SQL}(Q,D) = Eval_{SQL}((\Sigma, R, notexists(\Sigma', R', (l_1 \wedge H), P') \wedge notexists(\Sigma', R', (l_2 \wedge H), P'), P), D)$$

Proof.

$$\begin{aligned}
x \in^n Eval_{SQL}(Q,D) &\Leftrightarrow x \in^n Eval_{SQL}((\Sigma, R, notexists((\Sigma', R', (l_1 \wedge H) \vee (l_2 \wedge H), P'), P), D) \\
&\Leftrightarrow R^{\times}(\sigma^{-1}_{\Sigma}(x)) \geq n \wedge Eval_{SQL}((\Sigma', R', (l_1 \wedge H) \vee (l_2 \wedge H), P' \cup x]), D) = \emptyset \\
&\Leftrightarrow R^{\times}(\sigma^{-1}_{\Sigma}(x)) \geq n \wedge Eval_{SQL}((\Sigma', R', l_1 \wedge H, P' \cup x]), D) \\
&\cup Eval_{SQL}((\Sigma', R', l_1 \wedge H, P' \cup x]), D) = \emptyset \\
&\Leftrightarrow R^{\times}(\sigma^{-1}_{\Sigma}(x)) \geq n \wedge Eval_{SQL}((\Sigma', R', l_1 \wedge H, P' \cup x]), D) = \emptyset \\
&\wedge Eval_{SQL}((\Sigma', R', l_1 \wedge H, P' \cup x]), D) = \emptyset \\
&\Leftrightarrow x \in^n Eval_{SQL}((\Sigma, R, notexists(\Sigma', R', (l_1 \wedge H), P'), P), D) \\
&\wedge x \in^n Eval_{SQL}((\Sigma, R, notexists(\Sigma', R', (l_2 \wedge H), P'), P), D) \\
&\Leftrightarrow x \in^n Eval_{SQL}((\Sigma, R, notexists(\Sigma', R', (l_1 \wedge H), P') \wedge notexists(\Sigma', R', (l_2 \wedge H), P'), P), D)
\end{aligned}$$

□

A.6 Proof of proposition 6

Proposition. *Let* $Q = (\Sigma, R, notexists((\Sigma', R', H_r \wedge H, P'), P)$ *where* H_r *denote every conditions which are on attributes of r and such that* $r \in R'_s$ *then :*

$$Eval_{SQL}(Q, D) = Eval_{SQL}((\Sigma, R, notexists(\Sigma', R' \setminus r, H \wedge exists(*, r, H_r, P'), P'), P), D)$$

Proof. If $Eval_{SQL}((*, r, H_r, P'), D) = \emptyset$ then the proof is immediate. As $Eval_{SQL}(Q, D)$ will be equal to $\sigma_\Sigma(R^\times)$ in deed $Eval_{SQL}((\Sigma', R', H_r \wedge H, P'), D) = \emptyset$ as there is no element of r which are able to fulfill H_r condition. We found exactly the same for $Eval_{SQL}((\Sigma, R, notexists(\Sigma', R' \setminus r, H \wedge exists(*, r, H_r, P'), P'), P), D)$ in deed $exists(*, r, H_r, P')$ will never be true.

If $Eval_{SQL}((*, r, H_r, P'), D) \neq \emptyset$ then $exists(*, r, H_r, P')$ will always be true. Then $Eval_{SQL}((\Sigma, R, notexists(\Sigma', R' \setminus r, H \wedge exists(*, r, H_r, P'), P'), P), D) = Eval_{SQL}((\Sigma, R, notexists(\Sigma', R' \setminus r, H, P'), P), D)$ Moreover as r is not connected to any other relation. We have $\{Eval_{SQL}((\Sigma', R' \setminus r, H, P'), D)\} = \{Eval_{SQL}(\Sigma', R', H, P'), D)\}$. Indeed we are only multiplying the number of row by the number of tuples in r. Then as we are only checking if the set is empty it does not change anything because each attributes linked with a upper-level query will still have at least one representative. □

A.7 SQL nulls Translation

Here we present the translation for SQL Nulls. The only variation is due to the fact that $\bot_t = \bot_t$ is evaluate to true in case of marked nulls, while $\bot = \bot$ is evaluate to unknown in case of SQL nulls.

Figure 10: Translatation $Q \to (Q^+, Q^?)$

$$(\Sigma, R, H, P)^+ \to (\Sigma, R, H^*, P)$$
$$(\Sigma, R, H, P)^? \to (\Sigma, R, H^{**}, P)$$

Figure 11: Translatation $H \to H^*$

$$(H_1 \wedge H_2)^* \to H_1^* \wedge H_2^*$$
$$(H_1 \vee H_2)^* \to H_1^* \vee H_2^*$$
$$(r_i.a_i = c_i)^* \to r_i.a_i = c_i$$
$$(r_i.a_i \neq c_i)^* \to r_i.a_i \neq c_i$$
$$(r_i.a_i = r_j.a_j)^* \to r_i.a_i = r_j.a_j$$
$$(r_i.a_i \neq r_j.a_j)^* \to r_i.a_i \neq r_j.a_j$$
$$null(r_i.a_i)^* \to null(r_i.a_i)$$
$$const(r_i.a_i)^* \to const(r_i.a_i)$$
$$exists(Q)^* \to exists(Q^+)$$
$$notexists(Q)^* \to notexists(Q^?)$$

Figure 12: Translatation $H \to H^{**}$

$$
\begin{aligned}
(H_1 \wedge H_2)^{**} &\to H_1^{**} \wedge H_2^{**} \\
(H_1 \vee H_2)^{**} &\to H_1^{**} \vee H_2^{**} \\
(r_i.a_i = c_i)^{**} &\to r_i.a_i = c_i \vee null(r_i.a_i) \\
(r_i.a_i \neq c_i)^{**} &\to r_i.a_i \neq c_i \vee null(r_i.a_i) \\
(r_i.a_i = r_j.a_j)^{**} &\to r_i.a_i = r_j.a_j \vee null(r_i.a_i) \vee null(r_j.a_j) \\
(r_i.a_i \neq r_j.a_j)^{**} &\to r_i.a_i \neq r_j.a_j \vee null(r_i.a_i) \vee null(r_j.a_j) \\
null(r_i.a_i)^{**} &\to null(r_i.a_i) \\
const(r_i.a_i)^{**} &\to const(r_i.a_i) \\
exists(Q)^{**} &\to exists(Q^{?}) \\
notexists(Q)^{**} &\to notexists(Q^{+})
\end{aligned}
$$

A.8 Query rewriting step

Figure 13: Translated Query

```
SELECT o_orderkey
FROM orders
WHERE NOT EXISTS(
        SELECT *
        FROM lineitem , part
        WHERE l_orderkey = o_orderkey OR l_orderkey IS NULL OR o_orderkey IS NULL
        AND l_partkey = p_partkey OR l_partkey IS NULL OR p_partkey IS NULL);
```

Figure 14: Query after remove redundant

```
SELECT o_orderkey
FROM orders
WHERE NOT EXISTS(
        SELECT *
        FROM lineitem , part
        WHERE l_orderkey = o_orderkey
        AND l_partkey = p_partkey OR l_partkey IS NULL);
```

Figure 15: Query after split

```
SELECT o_orderkey
FROM orders
WHERE NOT EXISTS(
        SELECT *
        FROM lineitem , part
        WHERE l_orderkey = o_orderkey
        AND l_partkey = p_partkey)
AND NOT EXISTS(
        SELECT *
        FROM lineitem , part
        WHERE l_orderkey = o_orderkey
        AND l_partkey IS NULL);
```

Figure 16: Query after optimize FROM

```
SELECT o_orderkey
FROM orders
WHERE NOT EXISTS(
        SELECT *
        FROM lineitem , part
        WHERE l_orderkey = o_orderkey
        AND l_partkey = p_partkey)
AND NOT EXISTS(
        SELECT *
        FROM lineitem
        WHERE l_orderkey = o_orderkey
        AND EXISTS(SELECT * FROM part WHERE l_partkey IS NULL));
```

Figure 17: Query optimize with UNION ALL

```
SELECT o_orderkey
FROM orders
WHERE NOT EXISTS(
        SELECT *
        FROM ((SELECT * FROM lineitem , part WHERE l_partkey = p_partkey)
        UNION ALL (SELECT * FROM lineitem , part WHERE l_partkey IS NULL)) as lp
        WHERE l_orderkey = o_orderkey);
```