Aggregate Join Pushdown

Zhong Yu

March 25, 2018

A SQL aggregate over a join may be transformed such that some aggregation work is pushed down under the join, possibly reducing execution cost.

This article demonstrates and proves a way to do it for any type of aggregate, grouping, join condition, and join type (semi, anti, inner, outer). It is based on the fact that joins can be expressed as unions and products, and aggregations can be pushed down over unions and products.

1 Basic Notations and Formulas

1.1 set and duplicate rows

In this article, we'll use set notations to represent tables and operations on tables. It's understood that every row has an invisible marker that distinguish itself from any other row, even if they have the same values in all columns. No "deduplication" is done unless explicitly specified.

1.2 distinct and non-distinct

In this article, when comparing column values, we never invoke the concept of 'equality'. Instead, we only use 'distinct', 'not-distinct' as defined in SQL Standard, *and* only among values in the same type. When we use the terms '=', 'same', 'match', 'contain', we mean it in the sense of 'not-distinct'.

Row r_1 contains row r_2, if r_1 has all the columns of r_2, and their values are the same in these columns.

Filter δ_r selects rows that contain r.

1.3 special tables

$\emptyset$ is an empty table whose schema depends on the context.

$\varnothing$ is used to represent empty set of other stuff, depending on the context.

$I_r := \{r\}$ is a table with one row, r .

N^C is the table of columns C with exactly one row filled with null values in all columns. This is used in outer joins.

Γ^C is the table of columns C, containing all possible distinct values (including nulls). For example, if C consists of two boolean columns, Γ^C will have $3 * 3$ rows; each row is a distinct combination of nullable values of C. For an empty set of columns, $\Gamma^\varnothing$ consists of 1 row of zero columns.

In places that expects a column set, we may also put in a table or an operator to imply the columns of. For example, N^T implies columns of table T, Γ^A implies columns of result of aggregate A.

1.4 aggregate and group-by

An aggregate A applied on a table T, as AT, produces a table with exactly one row. This corresponds to SQL aggregation without a group-by clause.

Define $\tilde{A}$ as the same as A, except, when applied on an empty table, it produces zero rows:

$$\tilde{A}\emptyset = \emptyset; \quad \tilde{A}T = AT \text{ when } T \neq \emptyset$$

An aggregate A applied on a table T *partitioned* by a set of columns G , as $\overline{AG}T$, is defined as

$$\overline{AG}T := \bigcup_{g \in \Gamma^G} I_g \times \tilde{A}\delta_g T \tag{1}$$

When G is not empty, this definition corresponds to SQL's concept of *group-by*, that is, for every g contained by one or more rows in T, apply A on these rows; and then, g is attached to the result.

When G is empty, $G = \varnothing$, there is only one g in Γ^G, and $\delta_g = \sigma_{true}$, therefore $\overline{A\varnothing}T = \tilde{A}T$. This has no direct correspondence in SQL Standard; in particular, it does not correspond to '`group by ()`', which always produces one group, even if the table is empty; same as the case without a group-by clause.

1.4.1) Later, we will prove formulas in the form of $\overline{AG}T_1 = \overline{A^+G}T_2$, true for any G. We can deduce that $\overline{A\varnothing}T_1 = \overline{A^+\varnothing}T_2 \to \tilde{A}T_1 = \tilde{A}^+T_2 \to AT_1 = A^+T_2$ (see section 1.5). Therefore, we are proving the case of group by any set of columns, including '`group by ()`' . However, that's only on the outside; inside T_2 there may be $\overline{A_3\varnothing}T_3$ which we have to beware.

> Interestingly, in Oracle and MS SQL, the behavior of '`group by ()`' follows our definition of $\overline{A\varnothing}T$, and is different from the case of without group-by. Our proofs also cover all grouping cases in their dialect.

1.4.2) While $\overline{A\varnothing}T$ has no direct correspondence in standard SQL, it can be easily simulated; one way is to translate it to group by a constant value, for example, '`group by (x-x)`' , which seems to work consistently across major databases.

1.5 aggregate over union

Every aggregate A over unions can be expressed as

$$A\bigcup_i T_i = A^+\bigcup_i A'T_i \tag{2}$$

This can always be done; in the worst case, A' can retain all the rows.

Note that $A\emptyset = A^+\emptyset$, which is needed to prove (1.4.1).

It's easy to prove that,

$$\tilde{A}\bigcup_i T_i = \tilde{A}^+\bigcup_i \tilde{A}'T_i \tag{3}$$

1.6 aggregate over product

Every aggregate A' over a product can be expressed as

$$A'(L \times R) = a^*(A^L L \times A^R R) \tag{4}$$

where a^* is a generalized projection.

This can always be done; in the worst case, A^L and A^R can retain all the rows.

It's easy to see that

$$\tilde{A}'(L \times R) = a^*(\tilde{A}^L L \times \tilde{A}^R R) \tag{5}$$

1.6.1 on null row

For outer joins, we need to have the property of

$$A^L N^L = N^{A^L}, \quad A^R N^R = N^{A^R} \tag{1.6}$$

that is, A^L, A^R applying on an all-null row produces an all-null row. For many aggregates, this is naturally true, e.g. `sum(x)` . It is not always true though, e.g. `count(*)` .

Fortunately, we have the freedom of choice of a^*, A^L, A^R; and we can always find such operators that satisfies (4) and (1.6). If we already have $A'(L \times R) = b^*(B^L L \times B^R R)$, define a permutation function f_L on Γ^{B^L} such that $f_L B^L N^L = N^{B^L}$; define $A^L L := f_L B^L L$; define $a^*(a^L \times a^R) := b^*(f_L^{-1} a^L \times f_R^{-1} a^R)$. We can verify that a^*, A^L, A^R satisfies (4) and (1.6).

2 Inner Join

Every inner join with condition θ that references columns K can be expanded as

$$L \bowtie_\theta R = \bigcup_{k \in \sigma_\theta \Gamma^K} \delta_k(L \times R) \tag{6}$$

that is, we enumerate every k that satisfies θ, and pick out rows in $L \times R$ that match k.

Every k consists of k^L and k^R, and $\delta_k(L \times R) = \delta_{k^L} L \times \delta_{k^R} R$, thus,

$$L \bowtie_\theta R = \bigcup_k \delta_{k^L} L \times \delta_{k^R} R \tag{7}$$

We want to prove that the following two expressions are equivalent

$$E_1 := \overline{AG}(L \bowtie_\theta R) \tag{8}$$

$$E_2 := \overline{A^+ G} a^*(\overline{A^L P^L} L \bowtie_\theta \overline{A^R P^R} R) \tag{9}$$

where $P^L \supseteq G^L \cup K^L$, i.e. P^L contains L-side columns of G and K .

Let's expand E_2 first. Every g in Γ^G consists of g^L and g^R, and

$$\delta_g(L \times R) = \delta_{g^L} L \times \delta_{g^R} R \tag{10}$$

Apply (7, 1, 10) to E_2 (9)

$$E_2 = \bigcup_g I_g \times \tilde{A}^+ a^* \bigcup_{k, p^L, p^R} (\delta_{g^L} \delta_{k^L} I_{p^L} \times \tilde{A}^L \delta_{p^L} L) \times (...R) \tag{11}$$

For E_1, we need to expand L with P^L to match the form of (11).

$$L = \bigcup_{p^L \in \Gamma^{P^L}} \delta_{p^L} L \tag{12}$$

Apply (1, 7, 12, 10) to E_1 (8)

$$E_1 = \bigcup_g I_g \times \tilde{A} \bigcup_{k, p^L, p^R} (\delta_{g^L} \delta_{k^L} \delta_{p^L} L) \times (...R) \tag{13}$$

Apply (3, 5)

$$E_1 = \bigcup_g I_g \times \tilde{A}^+ a^* \bigcup_{k, p^L, p^R} (\tilde{A}^L \delta_{g^L} \delta_{k^L} \delta_{p^L} L) \times (...R) \tag{14}$$

Compare (14) to (11), we are missing a term I_{p^L} . Since a^* only operates on columns of A^L, A^R, it is harmless to introduce more columns to a^*'s operands with any value, e.g. p^L .

$$E_1 = \bigcup_g I_g \times \tilde{A}^+ a^* \bigcup_{k, p^L, p^R} (I_{p^L} \times \tilde{A}^L \delta_{g^L} \delta_{k^L} \delta_{p^L} L) \times (...R) \tag{15}$$

Compare (15) to (11), we need to prove that

$$\delta_{g^L} \delta_{k^L} I_{p^L} \times \tilde{A}^L \delta_{p^L} L = I_{p^L} \times \tilde{A}^L \delta_{g^L} \delta_{k^L} \delta_{p^L} L \tag{16}$$

If p^L contains both g^L and k^L, both sides become $I_{p^L} \times \tilde{A}^L \delta_{p^L} L$; otherwise, both sides become $\emptyset$.

Therefore, we proved that $E_1 = E_2$

$$\overline{AG}(L \bowtie_\theta R) = \overline{A^+G} a^* (\overline{A^L P^L} L \bowtie_\theta \overline{A^R P^R} R) \tag{17}$$

Refer to (1.4.1), this also implies that (with $G^L = G^R = \emptyset$)

$$A(L \bowtie_\theta R) = A^+ a^* (\overline{A^L P^L} L \bowtie_\theta \overline{A^R P^R} R) \tag{18}$$

2.1 in SQL

Translate the finding to SQL,

`select` A `from` L `inner join` R `on` $\theta(K)$ `group by` G

is equivalent to

`select` $A^+ a^*$ `from` L' `inner join` R' `on` $\theta(K)$ `group by` G

with

L' `:= select` A^L`,` P^L `from` L `group by` P^L

as long as P^L and P^R are not empty.

If P^L is empty, K^L is empty too; this is a pathological join, rare in practice. In this case, we cannot translate $\overline{A^L P^L}$ to SQL's `'group by ()'` ; but it could be implemented in SQL as mentioned in (1.4.2)

3 Semi Join and Anti Join

Every semi join with condition θ can be expressed as

$$L \ltimes_\theta R = \bigcup_{k \in \Upsilon(\theta, R)} \delta_k L \tag{19}$$

where $\Upsilon(\theta, R) := \Gamma^{K^L} \ltimes_\theta R$. Note that it depends on θ and values in R, but not on values in L. We enumerate every k of K^L that satisfies θ with a row in R, and pick out rows in L that match k.

Anti join $\triangleright_\theta$ is very similar to semi join; the only difference is the set of k; the rest of the deductions are exactly the same, therefore same conclusions apply.

We want to prove that the following two expressions are equivalent

$$E_1 := \overline{AG}(L \ltimes_\theta R) \tag{20}$$

$$E_2 := \overline{A^+G}(\overline{A'P}L \ltimes_\theta R) \tag{21}$$

where $P \supseteq G \cup K^L$. Note that aggregation and grouping is done on L side only.

The proof is similar to, and simpler than, the inner join case.

Apply (19, 12, 1) to E_2 (21)

$$E_2 = \bigcup_g I_g \times \tilde{A}^+ \bigcup_{k,p} \delta_g \delta_k I_p \times \tilde{A}' \delta_p L \tag{22}$$

Apply (19, 12, 1) to E_1 (20),

$$E_1 = \bigcup_g I_g \times \tilde{A} \bigcup_{k,p} \delta_g \delta_k \delta_p L \tag{23}$$

Apply (3),

$$E_1 = \bigcup_g I_g \times \tilde{A}^+ \bigcup_{k,p} \tilde{A}' \delta_g \delta_k \delta_p L \tag{24}$$

Compare (24) to (22), we are missing a term I_p . Since A^+ only operates on columns of A', it is harmless to introduce more columns to A^+'s operand with any value, e.g. p .

$$E_1 = \bigcup_g I_g \times \tilde{A}^+ \bigcup_{k,p} I_p \times \tilde{A}' \delta_g \delta_k \delta_p L \tag{25}$$

Compare (25) to (22), similar to the inner join case, it's true that

$$\delta_g \delta_k I_p \times \tilde{A}' \delta_p L = I_p \times \tilde{A}' \delta_g \delta_k \delta_p L \tag{26}$$

Therefore $E_1 = E_2$,

$$\overline{AG}(L \ltimes_\theta R) = \overline{A^+G}(\overline{A'P}L \ltimes_\theta R) \tag{27}$$

4 Outer Join

We can think of outer join as combination of inner join and anti join. For left-outer join,

$$L \ltimes\!\!\!\bowtie_\theta R = \quad L \bowtie_\theta R \quad \cup \quad (L \rhd_\theta R) \times N^R \tag{28}$$

We want to prove that the following two expressions are equivalent

$$E_1 := \overline{AG}(L \ltimes\!\!\!\bowtie_\theta R) \tag{29}$$

$$E_2 := \overline{A^+G}a^*(\overline{A^L P^L}L \ltimes\!\!\!\bowtie_\theta \overline{A^R P^R}R) \tag{30}$$

Using the same tactics as before, the join is expanded to unions of products, and aggregation is pushed down under unions and products, through which we can prove that $E_1 = E_2$. There might be better methods, but here is the ugly expansion:

$$E_1 = \bigcup_g I_g \times \tilde{A}^+ a^* \left(\bigcup_{j_1, p^L} (I_{p^L} \times \tilde{A}^L \delta_{g^L} \delta_{j_1} \delta_{p^L} L) \times (N^{P^R} \times \tilde{A}^R \delta_{g^R} N^R) \quad \cup \bigcup_{k, p^L, p^R} (I_{p^L} \times \tilde{A}^L \delta_{g^L} \delta_{k^L} \delta_{p^L} L) \times (...R) \right)$$

$$E_2 = \bigcup_g I_g \times \tilde{A}^+ a^* \left(\bigcup_{j_2, p^L} (\delta_{g^L} \delta_{j_2} I_{p^L} \times \tilde{A}^L \delta_{R^L} L) \times (\delta_{g^R} N^{\overline{A^R P^R}}) \quad \cup \bigcup_{k, p^L, p^R} (\delta_{g^L} \delta_{k^L} I_{p^L} \times \tilde{A}^L \delta_{p^L} L) \times (...R) \right)$$

where

$$j_1 \in \Gamma^{k^L} \rhd_\theta R, \quad j_2 \in \Gamma^{k^L} \rhd_\theta \overline{A^R P^R} R$$

It's intuitive that j_1, j_2 range over the same set.

Based on (1.6), the two terms involving N also match. At this point, we reached $E_1 = E_2$.

The case of full-outer join can be proven in the same way.

Actually, all types of joins – inner, semi/anti, left/right/full-outer – can be handled in a unified way. However, proving it in a unified framework is even more complex, therefore we break it down to individual types for "easier" grasp. An attempt was made to prove it for binary operators of a more abstract structure, but it is also full of deltas, partitions, crosses, and unions.

5 Conclusion

Given any aggregate A, we can find A', a^*, A^L, A^R that satisfies (2), (4) and (1.6) . When A is applied on any join with any condition with any grouping, it can be transformed to an equivalent expression where A', A^L, A^R are pushed down under the join.

Whether such transformation is profitable is left to practical concerns.