Formalization of Multiway-Join Algorithms

Thibault Dardinier

December 14, 2021

Abstract

Worst-case optimal multiway-join algorithms are recent seminal achievement of the database community. These algorithms compute the natural join of multiple relational databases and improve in the worst case over traditional query plan optimizations of nested binary joins. In 2014, Ngo, Ré, and Rudra [1] gave a unified presentation of different multi-way join algorithms. We formalized and proved correct their "Generic Join" algorithm and extended it to support negative joins.

Contents

1 The algorithm

theory *Generic_Join*
imports *MFOTL_Monitor.Table*
begin

type_synonym *'a atable* = *nat set* × *'a table*
type_synonym *'a query* = *'a atable set*
type_synonym *vertices* = *nat set*

1.1 Generic algorithm

locale *getIJ* =
fixes *getIJ* :: *'a query* ⇒ *'a query* ⇒ *vertices* ⇒ *vertices* × *vertices*
assumes *coreProperties*: *card V* ≥ *2* ⟹ *getIJ Q_pos Q_neg V* = (*I*, *J*) ⟹
card I ≥ *1* ∧ *card J* ≥ *1* ∧ *V* = *I* ∪ *J* ∧ *I* ∩ *J* = {}
begin

lemma *getIJProperties*:
assumes *card V* ≥ *2*
assumes (*I*, *J*) = *getIJ Q_pos Q_neg V*
shows *card I* ≥ *1* **and** *card J* ≥ *1* **and** *card I* < *card V* **and** *card J* < *card V*

```
    and V = I ∪ J and I ∩ J = {}
proof −
  show 1 ≤ card I using coreProperties[of V Q_pos Q_neg I J] assms by auto
  show 1 ≤ card J using coreProperties[of V Q_pos Q_neg I J] assms by auto
  show card I < card V by (metis (no_types, lifting) Int_ac(3) One_nat_def Suc_le_lessD assms(1)
       assms(2) card_gt_0_iff card_seteq dual_order.trans getIJ.coreProperties getIJ_axioms leI
       le_iff_inf one_le_numeral sup_ge1 sup_ge2)
  show card J < card V by (metis One_nat_def Suc_1 assms(1) assms(2) card_gt_0_iff card_seteq
       getIJ.coreProperties getIJ_axioms leI le_0_eq le_iff_inf nat.simps(3) sup_ge1 sup_ge2)
  show V = I ∪ J by (metis assms(1) assms(2) getIJ.coreProperties getIJ_axioms)
  show I ∩ J = {} by (metis assms(1) assms(2) getIJ_axioms getIJ_def)
qed

fun projectTable :: vertices ⇒ 'a atable ⇒ 'a atable where
  projectTable V (s, t) = (s ∩ V, Set.image (restrict V) t)

fun filterQuery :: vertices ⇒ 'a query ⇒ 'a query where
  filterQuery V Q = Set.filter (λ(s, _). ¬ Set.is_empty (s ∩ V)) Q

fun filterQueryNeg :: vertices ⇒ 'a query ⇒ 'a query where
  filterQueryNeg V Q = Set.filter (λ(A, _). A ⊆ V) Q

fun projectQuery :: vertices ⇒ 'a query ⇒ 'a query where
  projectQuery V s = Set.image (projectTable V) s

fun isSameIntersection :: 'a tuple ⇒ nat set ⇒ 'a tuple ⇒ bool where
  isSameIntersection t1 s t2 = (∀ i∈s. t1!i = t2!i)

fun semiJoin :: 'a atable ⇒ (nat set × 'a tuple) ⇒ 'a atable where
  semiJoin (s, tab) (st, tup) = (s, Set.filter (isSameIntersection tup (s ∩ st)) tab)

fun newQuery :: vertices ⇒ 'a query ⇒ (nat set × 'a tuple) ⇒ 'a query where
  newQuery V Q (st, t) = Set.image (λtab. projectTable V (semiJoin tab (st, t))) Q

fun merge_option :: 'a option × 'a option ⇒ 'a option where
  merge_option (None, None) = None
| merge_option (Some x, None) = Some x
| merge_option (None, Some x) = Some x
| merge_option (Some a, Some b) = Some a


fun merge :: 'a tuple ⇒ 'a tuple ⇒ 'a tuple where
  merge t1 t2 = map merge_option (zip t1 t2)

function (sequential) genericJoin :: vertices ⇒ 'a query ⇒ 'a query ⇒ 'a table where
  genericJoin V Q_pos Q_neg =
    (if card V ≤ 1 then
      (⋂(_, x) ∈ Q_pos. x) − (⋃(_, x) ∈ Q_neg. x)
    else
      let (I, J) = getIJ Q_pos Q_neg V in
      let Q_I_pos = projectQuery I (filterQuery I Q_pos) in
      let Q_I_neg = filterQueryNeg I Q_neg in
      let R_I = genericJoin I Q_I_pos Q_I_neg in
      let Q_J_neg = Q_neg − Q_I_neg in
      let Q_J_pos = filterQuery J Q_pos in
       let X = {(t, genericJoin J (newQuery J Q_J_pos (I, t)) (newQuery J Q_J_neg (I, t))) | t . t ∈
R_I} in
      (⋃(t, x) ∈ X. {merge xx t | xx . xx ∈ x}))
```

```
by pat_completeness auto
termination
  by (relation measure (λ(V, Q_pos, Q_neg). card V)) (auto simp add: getIJProperties)

fun wrapperGenericJoin :: 'a query ⇒ 'a query ⇒ 'a table where
  wrapperGenericJoin Q_pos Q_neg =
    (if ((∃(A, X)∈Q_pos. Set.is_empty X) ∨ (∃(A, X)∈Q_neg. Set.is_empty A ∧ ¬ Set.is_empty X))
then
      {}
    else
      let Q = Set.filter (λ(A, _). ¬ Set.is_empty A) Q_pos in
      if Set.is_empty Q then
        (⋂(A, X)∈Q_pos. X) −  (⋃(A, X)∈Q_neg. X)
      else
        let V = (⋃(A, X)∈Q. A) in
        let Qn = Set.filter (λ(A, _). A ⊆ V ∧ card A ≥ 1) Q_neg in
        genericJoin V Q Qn)

end
```

1.2 An instantation

```
fun score :: 'a query ⇒ nat ⇒ nat where
  score Q i = (let relevant = Set.image (λ(_, x). card x) (Set.filter (λ(sign, _). i ∈ sign) Q) in
    let l = sorted_list_of_set relevant in
    foldl (+) 0 l
)

fun arg_max_list :: ('a ⇒ nat) ⇒ 'a list ⇒ 'a where
  arg_max_list f l = (let m = Max (set (map f l)) in arg_min_list (λx. m − f x) l)

lemma arg_max_list_element:
  assumes length l ≥ 1 shows arg_max_list f l ∈ set l
 by (metis One_nat_def arg_max_list.simps arg_min_list_in assms le_imp_less_Suc less_irrefl list.size(3))

fun max_getIJ :: 'a query ⇒ 'a query ⇒ vertices ⇒ vertices × vertices where
  max_getIJ Q_pos Q_neg V = (
  let l = sorted_list_of_set V in
  if Set.is_empty Q_neg then
    let x = arg_max_list (score Q_pos) l in
    ({x}, V − {x})
  else
    let x = arg_max_list (score Q_neg) l in
    (V − {x}, {x}))

lemma max_getIJ_coreProperties:
  assumes card V ≥ 2
  assumes (I, J) = max_getIJ Q_pos Q_neg V
  shows card I ≥ 1 ∧ card J ≥ 1 ∧ V = I ∪ J ∧ I ∩ J = {}
proof −
  have finite V using assms(1) card.infinite by force
  define l where l = sorted_list_of_set V
  then have length l ≥ 1 by (metis Suc_1 Suc_le_lessD ‹finite V› assms(1) distinct_card
        distinct_sorted_list_of_set less_imp_le set_sorted_list_of_set)
  show ?thesis
  proof (cases Set.is_empty Q_neg)
    case True
    define x where x = arg_max_list (score Q_pos) l
```

then have $x \in (set\ l)$ **using** ‹$1 \leq length\ l$› $arg\_max\_list\_element$ **by** $blast$
then have $x \in V$ **by** ($simp\ add$: ‹$finite\ V$› l_def)
moreover have $(I, J) = (\{x\}, V - \{x\})$
proof −
have $(I, J) =$ ($let\ l = sorted\_list\_of\_set\ V\ in$
$let\ x = arg\_max\_list\ (score\ Q\_pos)\ l\ in$
$(\{x\}, V - \{x\})$) **by** ($simp\ add$: $True\ assms(2)$)
then show $?thesis$ **by** ($metis\ l\_def\ x\_def$)
qed
then show $?thesis$ **using** $Pair\_inject$ ‹$finite\ V$› $assms(1)$ $calculation$ **by** $auto$
next
case $False$
define x **where** $x = arg\_max\_list\ (score\ Q\_neg)\ l$
then have $x \in (set\ l)$ **using** ‹$1 \leq length\ l$› $arg\_max\_list\_element$ **by** $blast$
then have $x \in V$ **by** ($simp\ add$: ‹$finite\ V$› l_def)
moreover have $(I, J) = (V - \{x\}, \{x\})$
proof −
have $(I, J) = (let\ l = sorted\_list\_of\_set\ V\ in$
$let\ x = arg\_max\_list\ (score\ Q\_neg)\ l\ in\ (V - \{x\}, \{x\}))$ **by** ($simp\ add$: $False\ assms(2)$)
then show $?thesis$ **by** ($metis\ l\_def\ x\_def$)
qed
then show $?thesis$ **using** $Pair\_inject$ ‹$finite\ V$› $assms(1)$ $calculation$ **by** $auto$
qed
qed

global_interpretation New_max: $getIJ\ max\_getIJ$
defines $New\_max\_getIJ\_genericJoin = New\_max.genericJoin$
and $New\_max\_getIJ\_wrapperGenericJoin = New\_max.wrapperGenericJoin$
by $standard$ ($metis\ max\_getIJ\_coreProperties$)

end

2 Correctness

2.1 Well-formed queries

theory $Generic\_Join\_Correctness$
imports $Generic\_Join$
begin

definition $wf\_set :: nat \Rightarrow vertices \Rightarrow bool$ **where**
$wf\_set\ n\ V \longleftrightarrow (\forall x \in V.\ x < n)$

definition $wf\_atable :: nat \Rightarrow 'a\ atable \Rightarrow bool$ **where**
$wf\_atable\ n\ X \longleftrightarrow table\ n\ (fst\ X)\ (snd\ X) \wedge finite\ (fst\ X)$

definition $wf\_query :: nat \Rightarrow vertices \Rightarrow 'a\ query \Rightarrow 'a\ query \Rightarrow bool$ **where**
$wf\_query\ n\ V\ Q\_pos\ Q\_neg \longleftrightarrow (\forall X \in (Q\_pos \cup Q\_neg).\ wf\_atable\ n\ X) \wedge (wf\_set\ n\ V) \wedge (card\ Q\_pos \geq 1)$

definition $included :: vertices \Rightarrow 'a\ query \Rightarrow bool$ **where**
$included\ V\ Q \longleftrightarrow (\forall (S, X) \in Q.\ S \subseteq V)$

definition $covering :: vertices \Rightarrow 'a\ query \Rightarrow bool$ **where**
$covering\ V\ Q \longleftrightarrow V \subseteq (\bigcup (S, X) \in Q.\ S)$

definition $non\_empty\_query :: 'a\ query \Rightarrow bool$ **where**
$non\_empty\_query\ Q = (\forall X \in Q.\ card\ (fst\ X) \geq 1)$

```
definition rwf_query :: nat ⇒ vertices ⇒ 'a query ⇒ 'a query ⇒ bool where
  rwf_query n V Qp Qn ⟷ wf_query n V Qp Qn ∧ covering V Qp ∧ included V Qp ∧ included V Qn
                       ∧ non_empty_query Qp ∧ non_empty_query Qn

lemma wf_tuple_empty: wf_tuple n {} v ⟷ v = replicate n None
  by (auto intro!: replicate_eqI simp add: wf_tuple_def in_set_conv_nth)

lemma table_empty: table n {} X ⟷ (X = empty_table ∨ X = unit_table n)
  by (auto simp add: wf_tuple_empty unit_table_def table_def)

context getIJ begin

lemma isSame_equi_dev:
  assumes wf_set n V
  assumes wf_tuple n A t1
  assumes wf_tuple n B t2
  assumes s ⊆ A
  assumes s ⊆ B
  assumes A ⊆ V
  assumes B ⊆ V
  shows isSameIntersection t1 s t2 = (restrict s t1 = restrict s t2)
proof −
  have (∀ i∈s. t1!i = t2!i) ⟷ (restrict s t1 = restrict s t2) (is ?A ⟷ ?B)
  proof −
    have ?B ⟹ ?A
    proof −
      assume ?B
      have ⋀i. i∈s ⟹ t1!i = t2!i
      proof −
        fix i assume i ∈ s
        then have i ∈ A using assms(4) by blast
        then have i < n using assms(1) assms(6) wf_set_def by auto
        then show t1!i = t2!i by (metis (no_types, lifting) ‹i ∈ s› ‹restrict s t1 = restrict s t2›
            assms(2) length_restrict nth_restrict wf_tuple_length)
      qed
      then show ?A by blast
    qed
    moreover have ?A ⟹ ?B
    proof −
      assume ?A
      obtain length (restrict s t1) = n length (restrict s t2) = n
        using assms(2) assms(3) length_restrict wf_tuple_length by blast
      then have ⋀i. i < n ⟹ (restrict s t1)!i = (restrict s t2)!i
      proof −
        fix i assume i < n
        then show (restrict s t1)!i = (restrict s t2)!i
        proof (cases i ∈ s)
          case True
          then show ?thesis by (metis ‹∀ i∈s. t1 ! i = t2 ! i› ‹i < n› ‹length (restrict s t1) = n›
              ‹length (restrict s t2) = n› length_restrict nth_restrict)
        next
          case False
          then show ?thesis
         by (metis (no_types, opaque_lifting) ‹i < n› assms(2) assms(3) assms(4) assms(5) wf_tuple_def
              wf_tuple_restrict_simple)
        qed
      qed
```

```
    then show ?B
      by (metis ‹length (restrict s t1) = n› ‹length (restrict s t2) = n› nth_equalityI)
   qed
   then show ?thesis using calculation by linarith
  qed
  then show ?thesis by simp
qed

lemma wf_getIJ:
  assumes card V ≥ 2
  assumes wf_set n V
  assumes (I, J) = getIJ Q_pos Q_neg V
  shows wf_set n I and wf_set n J
  using assms unfolding wf_set_def by (metis Un_iff coreProperties)+

lemma wf_projectTable:
  assumes wf_atable n X
  shows wf_atable n (projectTable I X) ∧ (fst (projectTable I X) = (fst X ∩ I))
proof −
  obtain Y where Y = projectTable I X by simp
  obtain sX tX where (sX, tX) = X by (metis surj_pair)
  moreover obtain S where S = I ∩ sX by simp
  moreover obtain sY tY where (sY, tY) = Y by (metis surj_pair)
  then have sY = S
    using calculation(1) calculation(2) ‹Y = projectTable I X› by auto
  then have ⋀t. t ∈ tY ⟹ wf_tuple n S t
  proof −
    fix t assume t ∈ tY
    obtain x where x ∈ tX t = restrict I x using ‹(sY, tY) = Y› ‹t ∈ tY› ‹Y = projectTable I X›
calculation(1) by auto
    then have wf_tuple n sX x
    proof −
      have table n sX tX using assms(1) calculation(1) wf_atable_def by fastforce
      then show ?thesis using ‹x ∈ tX› table_def by blast
    qed
    then show wf_tuple n S t using ‹t = restrict I x› calculation(2) wf_tuple_restrict by blast
  qed
  then have ∀ t ∈ tY. wf_tuple n S t by blast
  then have table n S tY using table_def by blast
  then show ?thesis
    by (metis ‹(sY, tY) = Y› ‹Y = projectTable I X› ‹sY = S› assms calculation(1) calculation(2)
finite_Int fst_conv inf_commute snd_conv wf_atable_def)
qed

lemma set_filterQuery:
  assumes QQ = filterQuery I Q
  assumes non_empty_query Q
  shows ∀ X∈Q. (card (fst X ∩ I) ≥ 1 ⟷ X ∈ QQ)
proof −
  have ⋀X. X ∈ Q ⟹ (card (fst X ∩ I) ≥ 1 ⟷ X ∈ QQ)
  proof −
    fix X assume X ∈ Q
    have card (fst X ∩ I) ≥ 1 ⟹ X ∈ QQ
    proof −
      assume card (fst X ∩ I) ≥ 1
      then have (λ(s, _). s ∩ I ≠ {}) X by force
      then show ?thesis by (simp add: ‹case X of (s, uu_) ⇒ s ∩ I ≠ {}› Set.is_empty_def ‹X ∈ Q›
assms(1))
```

```
    qed
    moreover have X ∈ QQ ⟹ card (fst X ∩ I) ≥ 1
    proof −
      assume X ∈ QQ
      have (λ(s, _). s ∩ I ≠ {}) X using Set.is_empty_def ‹X ∈ QQ› assms(1) by auto
      then have fst X ∩ I ≠ {} by (simp add: case_prod_beta')
      then show ?thesis
          by (metis One_nat_def Suc_leI ‹X ∈ Q› assms(2) card.infinite card_gt_0_iff finite_Int
non_empty_query_def)
    qed
    then show (card (fst X ∩ I) ≥ 1 ⟷ X ∈ QQ)
      using calculation by blast
  qed
  then show ?thesis by blast
qed

lemma wf_filterQuery:
  assumes I ⊆ V
  assumes card I ≥ 1
  assumes rwf_query n V Qp Qn
  assumes QQp = filterQuery I Qp
  assumes QQn = filterQueryNeg I Qn
  shows wf_query n I QQp QQn non_empty_query QQp covering I QQp
proof −
  show non_empty_query QQp
   by (metis assms(3) assms(4) filterQuery.simps member_filter non_empty_query_def rwf_query_def)
  show covering I QQp
  proof −
    have ∀ X∈Qp. (card (fst X ∩ I) ≥ 1 ⟷ X ∈ QQp)
      using set_filterQuery assms(3) assms(4) rwf_query_def by fastforce
    have (⋃(S, X)∈Qp. S) ∩ I ⊆ (⋃(S, _)∈QQp. S) (is ?A ∩ I ⊆ ?B)
    proof (rule subsetI)
      fix x assume x ∈ ?A ∩ I
      have x ∈ ?A using ‹x ∈ (⋃(S, X)∈Qp. S) ∩ I› by blast
      then obtain S X where (S, X) ∈ Qp and x ∈ S by blast
      moreover have (S, X) ∈ QQp by (metis Int_iff One_nat_def Suc_le_eq
          ‹∀ X∈Qp. (1 ≤ card (fst X ∩ I)) = (X ∈ QQp)› ‹x ∈ (⋃(S, X)∈Qp. S) ∩ I› assms(2)
          calculation(1) calculation(2) card_gt_0_iff empty_iff finite_Int fst_conv)
      ultimately show x ∈ ?B by auto
    qed
    then show ?thesis
    by (metis (mono_tags, lifting) assms(1) assms(3) covering_def inf.absorb_iff2 le_infI1 rwf_query_def)
  qed
  show wf_query n I QQp QQn
  proof −
    have (∀ X∈QQp. wf_atable n X)
      using assms(3) assms(4) rwf_query_def wf_query_def by fastforce
    moreover have (wf_set n I)
      by (meson assms(1) assms(3) rwf_query_def subsetD wf_query_def wf_set_def)
    moreover have card QQp ≥ 1
    proof −
      have covering I QQp by (simp add: ‹covering I QQp›)
      have ¬ (Set.is_empty QQp)
      proof (rule ccontr)
        assume ¬ (¬ (Set.is_empty QQp))
        have Set.is_empty QQp using ‹¬ ¬ Set.is_empty QQp› by auto
        have (⋃(S, X)∈QQp. S) = {} by (metis SUP_empty Set.is_empty_def ‹Set.is_empty QQp›)
        then show False
```

```
      by (metis ‹covering I QQp› assms(2) card_eq_0_iff covering_def not_one_le_zero subset_empty)
    qed
    moreover have finite QQp
    by (metis assms(3) assms(4) card.infinite filterQuery.simps finite_filter not_one_le_zero rwf_query_def
wf_query_def)
    then show ?thesis
      by (metis One_nat_def Set.is_empty_def Suc_leI calculation card_gt_0_iff)
  qed
  moreover have QQn ⊆ Qn
  proof −
    have QQn = filterQueryNeg I Qn by (simp add: assms(5))
    then show ?thesis by auto
  qed
  moreover have wf_query n I QQp Qn
  by (meson Un_iff assms(3) calculation(1) calculation(2) calculation(3) rwf_query_def wf_query_def)
  then have (∀ X∈Qn. wf_atable n X) by (simp add: wf_query_def)
  then show ?thesis
    by (meson ‹wf_query n I QQp Qn› calculation(4) subset_eq sup_mono wf_query_def)
 qed
qed

lemma wf_set_subset:
 assumes I ⊆ V
 assumes card I ≥ 1
 assumes wf_set n V
 shows wf_set n I
 using assms(1) assms(3) wf_set_def by auto

lemma wf_projectQuery:
 assumes card I ≥ 1
 assumes wf_query n I Q Qn
 assumes non_empty_query Q
 assumes covering I Q
 assumes ∀ X∈Q. card (fst X ∩ I) ≥ 1
 assumes QQ = projectQuery I Q
 assumes included I Qn
 assumes non_empty_query Qn
 shows rwf_query n I QQ Qn
proof −
 have wf_query n I QQ Qn
 proof −
  have ∀ X∈QQ. wf_atable n X using assms(2) assms(6) wf_query_def
   by (simp add: wf_projectTable wf_query_def)
  moreover have wf_set n I using assms(2) wf_query_def by blast
  moreover have card QQ ≥ 1
  proof −
   have card QQ = card (Set.image (projectTable I) Q) by (simp add: assms(6))
   then show ?thesis
       by (metis One_nat_def Suc_le_eq assms(2) card_gt_0_iff finite_imageI image_is_empty
wf_query_def)
  qed
  then show ?thesis by (metis Un_iff assms(2) calculation(1) wf_query_def)
 qed
 moreover have covering I QQ
 proof −
  have I ⊆ (⋃(S, X)∈Q. S) using assms(4) covering_def by auto
  moreover have (⋃(S, X)∈Q. S ∩ I) ⊆ (⋃(S, X)∈QQ. S)
  proof (rule subsetI)
```

```
    fix x assume x ∈ (⋃(S, X)∈Q. S ∩ I)
    obtain S X where (S, X) ∈ Q and x ∈ S ∩ I using ‹x ∈ (⋃(S, X)∈Q. S ∩ I)› by blast
    then have fst (projectTable I (S, X)) = S ∩ I by simp
    have wf_atable n (S, X) using ‹(S, X) ∈ Q› assms(2) wf_query_def by blast
    then have wf_atable n (projectTable I (S, X)) using wf_projectTable by blast
    then show x ∈ (⋃(S, X)∈QQ. S) using ‹(S, X) ∈ Q› ‹x ∈ S ∩ I› assms(6) by fastforce
  qed
  moreover have (⋃(S, X)∈Q. S ∩ I) = (⋃(S, X)∈Q. S) ∩ I by blast
  then show ?thesis using calculation(1) calculation(2) covering_def inf_absorb2 by fastforce
 qed
 moreover have included I QQ
 proof −
  have ⋀S X. (S, X) ∈ QQ ⟹ S ⊆ I
  proof −
   fix S X assume (S, X) ∈ QQ
   have (S, X) ∈ Set.image (projectTable I) Q using ‹(S, X) ∈ QQ› assms(6) by simp
   obtain XX where XX ∈ Q and (S, X) = projectTable I XX using ‹(S, X) ∈ projectTable I ‘ Q›
by blast
   then have S = I ∩ (fst XX)
    by (metis projectTable.simps fst_conv inf_commute prod.collapse)
   then show S ⊆ I by simp
  qed
  then have (∀(S, X)∈QQ. S ⊆ I) by blast
  then show ?thesis by (simp add: included_def)
 qed
 moreover have non_empty_query QQ using assms(5) assms(6) non_empty_query_def by fastforce
 then show ?thesis
  by (simp add: assms(7) assms(8) calculation(1) calculation(2) calculation(3) rwf_query_def)
qed

lemma wf_firstRecursiveCall:
 assumes rwf_query n V Qp Qn
 assumes card V ≥ 2
 assumes (I, J) = getIJ Qp Qn V
 assumes Q_I_pos = projectQuery I (filterQuery I Qp)
 assumes Q_I_neg = filterQueryNeg I Qn
 shows rwf_query n I Q_I_pos Q_I_neg
proof −
 obtain I ⊆ V card I ≥ 1 using assms(2) assms(3) getIJProperties(5) getIJProperties(1) by fastforce
 define tQ where tQ = filterQuery I Qp
 obtain wf_query n I tQ Q_I_neg non_empty_query tQ covering I tQ
  by (metis wf_filterQuery(1) wf_filterQuery(2) wf_filterQuery(3)
      ‹1 ≤ card I› ‹I ⊆ V› assms(1) assms(5) tQ_def)
 moreover obtain card I ≥ 1 and ∀X∈tQ. card (fst X ∩ I) ≥ 1
  using set_filterQuery ‹1 ≤ card I› assms(1) rwf_query_def tQ_def by fastforce
 moreover have included I Q_I_neg by (simp add: assms(5) included_def)
 then show ?thesis
  by (metis wf_projectQuery ‹⋀thesis. (⟦wf_query n I tQ Q_I_neg; non_empty_query tQ; covering I
tQ⟧ ⟹ thesis) ⟹ thesis›
       assms(1) assms(4) assms(5) calculation(4) calculation(5) filterQueryNeg.simps member_filter
non_empty_query_def rwf_query_def tQ_def)
qed

lemma wf_atable_subset:
 assumes table n V X
 assumes Y ⊆ X
 shows table n V Y
 by (meson assms(1) assms(2) subsetD table_def)
```

lemma *same_set_semiJoin*:
fst (*semiJoin x other*) = *fst x*
proof −
obtain *sx tx* **where** *x* = (*sx*, *tx*) **by** (*metis surj_pair*)
obtain *so to* **where** *other* = (*so*, *to*) **by** (*metis surj_pair*)
then show *?thesis* **by** (*simp add*: ‹*x* = (*sx*, *tx*)›)
qed

lemma *wf_semiJoin*:
assumes *card J* ≥ *1*
assumes *wf_query n J Q Qn*
assumes *non_empty_query Q*
assumes *covering J Q*
assumes ∀ *X*∈*Q*. *card* (*fst X* ∩ *J*) ≥ *1*
assumes *QQ* = (*Set.image* (λ*tab*. *semiJoin tab* (*st*, *t*)) *Q*)
shows *wf_query n J QQ Qn non_empty_query QQ covering J QQ*
proof −
show *wf_query n J QQ Qn*
proof −
have ∀ *X*∈*QQ*. *wf_atable n X*
proof −
have ⋀*X*. *X*∈*QQ* ⟹ *wf_atable n X*
proof −
fix *X* **assume** *X* ∈ *QQ*
obtain *Y* **where** *Y* ∈ *Q* **and** *X* = *semiJoin Y* (*st*, *t*) **using** ‹*X* ∈ *QQ*› *assms*(*6*) **by** *blast*
then have *wf_atable n Y* **using** *assms*(*2*) *wf_query_def* **by** *blast*
then show *wf_atable n X*
proof −
have *fst X* = *fst Y*
by (*metis* ‹*X* = *semiJoin Y* (*st*, *t*)› *fst_conv prod.collapse semiJoin.simps*)
moreover have *snd X* ⊆ *snd Y*
by (*metis* ‹*X* = *semiJoin Y* (*st*, *t*)› *member_filter prod.collapse semiJoin.simps snd_conv subsetI*)
then have *table n* (*fst X*) (*snd X*) **by** (*metis* ‹*wf_atable n Y*› *calculation wf_atable_def wf_atable_subset*)
moreover have *finite* (*fst X*) **by** (*metis* ‹*wf_atable n Y*› *calculation*(*1*) *wf_atable_def*)
then show *?thesis* **by** (*simp add*: *calculation*(*2*) *wf_atable_def*)
qed
qed
then show *?thesis* **by** *blast*
qed
moreover have *wf_set n J* **using** *assms*(*2*) *wf_query_def* **by** *blast*
moreover have *card QQ* ≥ *1*
by (*metis One_nat_def Suc_leI assms*(*2*) *assms*(*6*) *card.infinite card_gt_0_iff finite_imageI image_is_empty wf_query_def*)
then show *?thesis* **using** *calculation*(*1*) *calculation*(*2*) *wf_query_def Un_iff assms*(*2*) **by** *metis*
qed
show *non_empty_query QQ*
by (*metis* (*no_types*, *lifting*) *assms*(*3*) *assms*(*6*) *image_iff non_empty_query_def same_set_semiJoin*)
show *covering J QQ*
proof −
have (⋃(*S*, *X*)∈*Q*. *S*) = (⋃(*S*, *X*)∈*QQ*. *S*) **using** *assms*(*6*) *same_set_semiJoin* **by** *auto*
then show *?thesis* **by** (*metis assms*(*4*) *covering_def*)
qed
qed

lemma *newQuery_equiv_def*:

```
  newQuery V Q (st, t) = projectQuery V (Set.image (λtab. semiJoin tab (st, t)) Q)
  by (metis image_image newQuery.simps projectQuery.elims)

lemma included_project:
  included V (projectQuery V Q)
proof −
  have ⋀S X. (S, X)∈(projectQuery V Q) ⟹ S ⊆ V
  proof −
    fix S X assume (S, X)∈(projectQuery V Q)
    obtain SS XX where (S, X) = projectTable V (SS, XX)
      using ‹(S, X) ∈ projectQuery V Q› by auto
    then have S = SS ∩ V by auto
    then show S ⊆ V by simp
  qed
  then show ?thesis by (metis case_prodI2 included_def)
qed

lemma non_empty_newQuery:
  assumes Q1 = filterQuery J Q0
  assumes Q2 = newQuery J Q1 (I, t)
  assumes ∀ X∈Q0. wf_atable n X
  shows non_empty_query Q2
proof −
  have ⋀X. X∈Q2 ⟹ card (fst X) ≥ 1
  proof −
    fix X assume X ∈ Q2
    obtain X2 where X = projectTable J X2 and X2 ∈ Set.image (λtab. semiJoin tab (I, t)) Q1
      by (metis (mono_tags, lifting) newQuery.simps ‹X ∈ Q2› assms(2) image_iff)
    then have card (fst X2 ∩ J) ≥ 1
    proof −
      obtain X1 where X1 ∈ Q1 and X2 = semiJoin X1 (I, t)
        using ‹X2 ∈ (λtab. semiJoin tab (I, t)) ‘ Q1› by blast
      then have fst X1 = fst X2 by (simp add: same_set_semiJoin)
      moreover have X1 ∈ filterQuery J Q0 using ‹X1 ∈ Q1› assms(1) by blast
      then have (λ(s, _). s ∩ J ≠ {}) X1 using Set.is_empty_def by auto
      then have ¬ (Set.is_empty (fst X1 ∩ J)) by (simp add: Set.is_empty_def case_prod_beta′)
      then show ?thesis
        by (metis filterQuery.elims One_nat_def Set.is_empty_def Suc_leI ‹X1 ∈ Q1› assms(1)
            assms(3) calculation card_gt_0_iff finite_Int member_filter wf_atable_def)
    qed
    then show card (fst X) ≥ 1
      by (metis projectTable.simps ‹X = projectTable J X2› fst_conv prod.collapse)
  qed
  then show ?thesis by (simp add: non_empty_query_def)
qed

lemma wf_newQuery:
  assumes card J ≥ 1
  assumes wf_query n J Q Qn0
  assumes non_empty_query Q
  assumes covering J Q
  assumes ∀ X∈Q. card (fst X ∩ J) ≥ 1
  assumes QQ = newQuery J Q t
  assumes QQn = newQuery J Qn t
  assumes non_empty_query Qn
  assumes Qn = filterQuery J Qn0
  shows rwf_query n J QQ QQn
proof −
```

obtain *tt st* **where** $(st, tt) = t$ **by** (*metis surj_pair*)
have $QQ = projectQuery\ J\ (Set.image\ (\lambda tab.\ semiJoin\ tab\ (st, tt))\ Q)$
by (*metis* ‹$(st, tt) = t$› *assms*(6) *newQuery_equiv_def*)
define QS **where** $QS = Set.image\ (\lambda tab.\ semiJoin\ tab\ (st, tt))\ Q$
obtain *wf_query n J QS Qn0 non_empty_query QS covering J QS*
by (*metis wf_semiJoin*(1) *wf_semiJoin*(2) *wf_semiJoin*(3) *QS_def*
assms(1) *assms*(2) *assms*(3) *assms*(4) *assms*(5))
moreover have $\forall X{\in}QS.\ card\ (fst\ X \cap J) \geq 1$ **using** *QS_def assms*(5) **by** *auto*
then have $\forall X{\in}(projectQuery\ J\ QS).\ wf\_atable\ n\ X$
by (*metis* (*no_types*, *lifting*) *projectQuery.simps Un_iff calculation*(1) *image_iff*
wf_projectTable wf_query_def)
then have *wf_query n J QQ QQn*
proof −
have $\bigwedge X.\ X{\in}QQn \Longrightarrow wf\_atable\ n\ X$
proof −
fix X **assume** $X \in QQn$
have $QQn = projectQuery\ J\ (Set.image\ (\lambda tab.\ semiJoin\ tab\ (st, tt))\ Qn)$
using *newQuery_equiv_def* ‹$(st, tt) = t$› *assms*(7) **by** *blast*
then obtain XX **where** $X = projectTable\ J\ XX\ XX \in (Set.image\ (\lambda tab.\ semiJoin\ tab\ (st, tt))\ Qn)$
using ‹$X \in QQn$› **by** *auto*
then obtain XXX **where** $XX = semiJoin\ XXX\ (st, tt)\ XXX \in Qn$ **by** *blast*
then have *wf_atable n XXX*
by (*metis filterQuery.elims Un_iff assms*(2) *assms*(9) *member_filter wf_query_def*)
then have *wf_atable n XX*
proof −
have $fst\ XX = fst\ XXX$
by (*simp add*: *same_set_semiJoin* ‹$XX = semiJoin\ XXX\ (st, tt)$›)
moreover have $snd\ XX = Set.filter\ (isSameIntersection\ tt\ (fst\ XX \cap st))\ (snd\ XXX)$
by (*metis semiJoin.simps* ‹$XX = semiJoin\ XXX\ (st, tt)$› *calculation prod.collapse snd_conv*)
moreover have $snd\ XX \subseteq snd\ XXX$ **using** *calculation*(2) **by** *auto*
then show *?thesis*
by (*metis wf_atable_subset* ‹*wf_atable n XXX*› *calculation*(1) *wf_atable_def*)
qed
then show *wf_atable n X* **by** (*simp add*: *wf_projectTable* ‹$X = projectTable\ J\ XX$›)
qed
then have $\forall X{\in}QQn.\ wf\_atable\ n\ X$ **by** *blast*
then have $\forall X{\in}(QQ \cup QQn).\ wf\_atable\ n\ X$
using *QS_def* ‹$QQ = projectQuery\ J\ ((\lambda tab.\ semiJoin\ tab\ (st, tt))\ `\ Q)$› ‹$\forall X{\in}projectQuery\ J\ QS.$
wf_atable n X› **by** *blast*
moreover have $card\ QQ \geq 1$
by (*metis* (*no_types*, *lifting*) *newQuery.simps One_nat_def Suc_leI* ‹$(st, tt) = t$› *assms*(2)
assms(6) *card.infinite card_gt_0_iff finite_imageI image_is_empty wf_query_def*)
then show *?thesis* **using** *assms*(2) *calculation wf_query_def* **by** *blast*
qed
moreover have *included J QQn*
proof −
have $QQn = projectQuery\ J\ (Set.image\ (\lambda tab.\ semiJoin\ tab\ (st, tt))\ Qn)$
using *newQuery_equiv_def* ‹$(st, tt) = t$› *assms*(7) **by** *blast*
then show *?thesis* **using** *included_project* **by** *blast*
qed
moreover have *covering J QQ*
proof −
have $QQ = projectQuery\ J\ ((\lambda tab.\ semiJoin\ tab\ (st, tt))\ `Q)$
using ‹$QQ = projectQuery\ J\ ((\lambda tab.\ semiJoin\ tab\ (st, tt))\ `\ Q)$› **by** *blast*
then have *covering J* $((\lambda tab.\ semiJoin\ tab\ (st, tt))\ `Q)$ **using** *QS_def calculation*(3) **by** *blast*
then have $J \subseteq (\bigcup (S, X){\in}(((\lambda tab.\ semiJoin\ tab\ (st, tt))\ `Q)).\ S)$
by (*simp add*: *covering_def*)
then have $J \subseteq (\bigcup (S, X){\in}(((\lambda tab.\ semiJoin\ tab\ (st, tt))\ `Q)).\ S) \cap J$ **by** *blast*

moreover have $(\bigcup (S,\ X) \in (((\lambda tab.\ semiJoin\ tab\ (st,\ tt))\ `Q)).\ S) \cap J \subseteq (\bigcup (S,\ X) \in (((\lambda tab.\ semiJoin\ tab\ (st,\ tt))\ `Q)).\ S \cap J)$
using *image_cong* **by** *auto*
then have $(\bigcup (S,\ X) \in ((\lambda tab.\ semiJoin\ tab\ (st,\ tt))\ `Q).\ S) \cap J \subseteq (\bigcup (S,\ X) \in (projectQuery\ J\ ((\lambda tab.\ semiJoin\ tab\ (st,\ tt))\ `Q)).\ S)$
by *auto*
then show *?thesis*
by (*metis* ‹$J \subseteq (\bigcup (S,\ X) \in (\lambda tab.\ semiJoin\ tab\ (st,\ tt))\ `\ Q.\ S)$› ‹$QQ = projectQuery\ J\ ((\lambda tab.\ semiJoin\ tab\ (st,\ tt))\ `\ Q)$› *covering_def inf_absorb2*)
qed
moreover have *non_empty_query QQ* **using** *QS_def* ‹$QQ = projectQuery\ J\ ((\lambda tab.\ semiJoin\ tab\ (st,\ tt))\ `\ Q)$›
‹$\forall X \in QS.\ 1 \leq card\ (fst\ X \cap J)$› *non_empty_query_def* **by** *fastforce*
moreover have *non_empty_query QQn*
by (*metis non_empty_newQuery Un_iff* ‹$(st,\ tt) = t$› *assms*(7) *assms*(9) *calculation*(1) *wf_query_def*)
then show *?thesis*
using *included_project* ‹$QQ = projectQuery\ J\ ((\lambda tab.\ semiJoin\ tab\ (st,\ tt))\ `\ Q)$›
calculation(4) *calculation*(5) *calculation*(6) *calculation*(7) *rwf_query_def* **by** *blast*
qed

lemma *subset_Q_neg*:
assumes *rwf_query n V Q Qn*
assumes $QQn \subseteq Qn$
shows *rwf_query n V Q QQn*
proof −
have *wf_query n V Q QQn*
proof −
have $\forall X \in QQn.$ *wf_atable n X* **by** (*meson Un_iff assms*(1) *assms*(2) *rwf_query_def subsetD wf_query_def*)
then show *?thesis*
by (*meson UnE UnI1 assms*(1) *rwf_query_def wf_query_def*)
qed
moreover have *included V QQn* **by** (*meson assms*(1) *assms*(2) *included_def rwf_query_def subsetD*)
then show *?thesis* **by** (*metis* (*full_types*) *assms*(2) *non_empty_query_def subsetD assms*(1) *calculation rwf_query_def*)
qed

lemma *wf_secondRecursiveCalls*:
assumes $card\ V \geq 2$
assumes *rwf_query n V Q Qn*
assumes $(I,\ J) = getIJ\ Q\ Qn\ V$
assumes $Qns \subseteq Qn$
assumes $Q\_J\_neg = filterQuery\ J\ Qns$
assumes $Q\_J\_pos = filterQuery\ J\ Q$
shows *rwf_query n J* (*newQuery J Q_J_pos t*) (*newQuery J Q_J_neg t*)
proof −
have $\forall X \in Q\_J\_pos.\ card\ (fst\ X \cap J) \geq 1$
using *set_filterQuery assms*(2) *assms*(6) *rwf_query_def* **by** *fastforce*
moreover have $card\ J \geq 1$ **by** (*metis assms*(1) *assms*(3) *getIJ.coreProperties getIJ_axioms*)
moreover have *wf_query n J Q_J_pos Qns*
proof −
have *wf_query n J Q Qns*
by (*metis subset_Q_neg wf_set_subset assms*(1) *assms*(2) *assms*(3) *assms*(4)
getIJ.coreProperties getIJ_axioms rwf_query_def sup_ge2 wf_query_def)
moreover have $Q\_J\_pos \subseteq Q$ **using** *assms*(6) **by** *auto*
then have $\forall X \in (Q\_J\_pos \cup Qns).$ *wf_atable n X* **using** *calculation wf_query_def* **by** *fastforce*
moreover have $card\ Q\_J\_pos \geq 1$
by (*metis wf_filterQuery*(1) *assms*(1) *assms*(2) *assms*(3) *assms*(6) *getIJ.coreProperties*

```
      getIJ_axioms sup_ge2 wf_query_def)
  then show ?thesis using calculation(1) calculation(2) wf_query_def by blast
 qed
 moreover have non_empty_query Q_J_pos
  by (metis wf_filterQuery(2) assms(1) assms(2) assms(3) assms(6) getIJ.coreProperties
      getIJ_axioms sup_ge2)
 moreover have  covering J Q_J_pos
  by (metis wf_filterQuery(3) assms(1) assms(2) assms(3) assms(6) getIJ.coreProperties
      getIJ_axioms sup_ge2)
 moreover have non_empty_query Q_J_neg
  by (metis (no_types, lifting) filterQuery.elims assms(2) assms(4) assms(5) member_filter
      non_empty_query_def rwf_query_def subsetD)
 then show ?thesis
  using wf_newQuery assms(5) calculation(1) calculation(2) calculation(3) calculation(4)
   calculation(5) by blast
qed

lemma simple_merge_option:
 merge_option (a, b) = None ⟷ (a = None ∧ b = None)
 using merge_option.elims by blast

lemma wf_merge:
 assumes wf_tuple n I t1
 assumes wf_tuple n J t2
 assumes V = I ∪ J
 assumes t = merge t1 t2
 shows wf_tuple n V t
proof −
 have ⋀i. i < n ⟹ (t ! i = None ⟷ i ∉ V)
 proof −
  fix i
  assume i < n
  show t ! i = None ⟷ i ∉ V
  proof (cases t ! i = None)
   case True
   have t = merge t1 t2 by (simp add: assms(4))
   then have ... = map merge_option (zip t1 t2) by simp
   then have merge_option (t1 ! i, t2 ! i) = None
     by (metis True ‹i < n› assms(1) assms(2) assms(4) length_zip min_less_iff_conj nth_map
nth_zip wf_tuple_def)
   obtain t1 ! i = None and t2 ! i = None
    by (meson ‹merge_option (t1 ! i, t2 ! i) = None› simple_merge_option)
   then show ?thesis
    using True ‹i < n› assms(1) assms(2) assms(3) wf_tuple_def by auto
  next
   case False
   have t = map merge_option (zip t1 t2) by (simp add: assms(4))
   then obtain x where merge_option (t1 ! i, t2 ! i) = Some x
     by (metis False ‹i < n› assms(1) assms(2) length_zip merge_option.elims min_less_iff_conj
nth_map nth_zip wf_tuple_def)
   then show ?thesis
     by (metis False UnI1 UnI2 ‹i < n› assms(1) assms(2) assms(3) option.distinct(1) sim-
ple_merge_option wf_tuple_def)
  qed
 qed
 moreover have length t = n
 proof −
  obtain length t1 = n and length t2 = n
```

```
    using assms(1) assms(2) wf_tuple_def by blast
   then have length (zip t1 t2) = n by simp
   then show ?thesis by (simp add: assms(4))
  qed
  then show ?thesis by (simp add: calculation wf_tuple_def)
qed

lemma wf_inter:
  assumes rwf_query n {i} Q Qn
  assumes (sa, a) ∈ Q
  assumes (sb, b) ∈ Q
  shows table n {i} (a ∩ b)
proof −
  obtain card sa ≥ 1 card sb ≥ 1
    by (metis assms(1) assms(2) assms(3) fst_conv non_empty_query_def rwf_query_def)
  have included {i} Q using assms(1) rwf_query_def by blast
  then have (∀(S, X)∈Q. S ⊆ {i}) by (simp add: included_def)
  then obtain sa ⊆ {i} sb ⊆ {i} using assms(2) assms(3) by blast
  then obtain sa = {i} sb = {i}
    by (metis ‹1 ≤ card sa› ‹1 ≤ card sb› card.empty not_one_le_zero subset_singletonD)
  then show ?thesis
    using assms(1) assms(2) inf_le1 prod.sel(1) prod.sel(2) rwf_query_def wf_atable_def
      wf_atable_subset wf_query_def Un_iff by metis
qed

lemma table_subset:
  assumes table n V T
  assumes S ⊆ T
  shows table n V S
  using wf_atable_subset assms(1) assms(2) by blast

lemma wf_base_case:
  assumes card V = 1
  assumes rwf_query n V Q Qn
  assumes R = genericJoin V Q Qn
  shows table n V R
proof −
  have wf_query n V Q Qn ∧ included V Q ∧ non_empty_query Q ⟹ table n V ((⋂(_, x) ∈ Q. x) −
(⋃(_, x) ∈ Qn. x))
  proof (induction card Q − 1 arbitrary: Q)
    case 0
    have card Q = 1
      by (metis 0.hyps 0.prems One_nat_def le_add_diff_inverse plus_1_eq_Suc wf_query_def)
    obtain s x where Q = {(s, x)}
     by (metis One_nat_def ‹card Q = 1› card_eq_0_iff card_eq_SucD card_mono finite_insert insertE
         nat.simps(3) not_one_le_zero subrelI)
    moreover obtain i where V = {i} using assms(1) card_1_singletonE by auto
    then have card s ≥ 1
    proof −
      have (s, x) ∈ Q by (simp add: calculation)
      moreover obtain X where X = (s, x) by simp
      then show ?thesis
        using 0.prems calculation non_empty_query_def rwf_query_def by fastforce
    qed
    moreover obtain i where V = {i} using ‹⋀thesis. (⋀i. V = {i} ⟹ thesis) ⟹ thesis› by blast
    then have s = {i}
    proof −
      have included {i} Q using 0.prems ‹V = {i}› rwf_query_def by simp
```

```
    then show ?thesis
      by (metis ‹V = {i}› assms(1) calculation(1) calculation(2) card_seteq case_prodD finite.emptyI
finite.insertI included_def singletonI)
  qed
  moreover have table n s x
    using 0.prems calculation(1) rwf_query_def wf_atable_def wf_query_def
    by (simp add: rwf_query_def wf_atable_def wf_query_def)
  then show ?case
    by (simp add: wf_atable_subset ‹V = {i}› calculation(1) calculation(3))
 next
  case (Suc y)
 obtain xx where xx ∈ Q by (metis Suc.hyps(2) all_not_in_conv card.empty nat.simps(3) zero_diff)
  moreover obtain H where H = Q − {xx} by simp
  then have card H − 1 = y
  by (metis Suc.hyps(2) calculation card_Diff_singleton card.infinite diff_Suc_1 less_imp_le not_one_le_zero
zero_less_Suc zero_less_diff)
  moreover have wf_query n V H Qn ∧ included V H ∧ non_empty_query H
  proof −
    have wf_query n V H Qn
      using DiffD1 Suc.hyps(2) Suc.prems ‹H = Q − {xx}› calculation(1) card_Diff_singleton
        card.infinite le_add1 not_one_le_zero plus_1_eq_Suc wf_query_def
      by (metis (no_types, lifting) Un_iff)
    then show ?thesis
      using DiffD1 Suc.prems ‹H = Q − {xx}› included_def non_empty_query_def by fastforce
  qed
  then have wf_query n V H Qn ∧ included V H ∧ non_empty_query H by simp
  then have table n V ((⋂(_, x)∈H. x) − (⋃(_, x)∈Qn. x)) using Suc.hyps(1) calculation(2) by
simp
  moreover obtain sa a where (sa, a) ∈ H
    by (metis One_nat_def Suc.hyps(2) ‹H = Q − {xx}› calculation(1) calculation(2) card.empty
card_eq_0_iff card_le_Suc0_iff_eq diff_is_0_eq′ equals0I insert_Diff le0 nat.simps(3) prod.collapse
singletonD)
  moreover have ¬ (Set.is_empty sa)
    by (metis Set.is_empty_def ‹wf_query n V H Qn ∧ included V H ∧ non_empty_query H› calcu-
lation(4)
        card.empty non_empty_query_def not_one_le_zero prod.sel(1))
  then have table n V ((((⋂(_, x) ∈ H. x) ∩ (snd xx)) − (⋃(_, x)∈Qn. x))
    by (metis Diff_Int2 Diff_Int_distrib2 IntE calculation(3) table_def)
 then show ?case using INF_insert Int_commute ‹H = Q − {xx}› calculation(1) insert_Diff snd_def
by metis
 qed
 then show ?thesis
  using assms(1) assms(2) assms(3) genericJoin.simps le_numeral_extra(4) rwf_query_def by auto
qed

lemma filter_Q_J_neg_same:
 assumes card V ≥ 2
 assumes (I, J) = getIJ Q Qn V
 assumes Q_I_neg = filterQueryNeg I Qn
 assumes rwf_query n V Q Qn
 shows filterQuery J (Qn − Q_I_neg) = Qn − Q_I_neg (is ?A = ?B)
proof−
 have ?A ⊆ ?B by (simp add: subset_iff)
 moreover have ?B ⊆ ?A
 proof (rule subsetI)
  fix x assume x ∈ Qn − Q_I_neg
  obtain A X where (A, X) = x by (metis surj_pair)
  have card (A ∩ J) ≥ 1
```

```
    proof (rule ccontr)
      assume ¬ (card (A ∩ J) ≥ 1)
      have Set.is_empty (A ∩ J)
        by (metis One_nat_def Set.is_empty_def Suc_leI Suc_le_lessD ‹¬ 1 ≤ card (A ∩ J)› assms(1)
            assms(2) card_gt_0_iff finite_Int getIJ.coreProperties getIJ_axioms)
      moreover have A ⊆ I
      proof −
        have (A, X) ∈ Qn using ‹(A, X) = x› ‹x ∈ Qn − Q_I_neg› by auto
        then have included V Qn using assms(4) rwf_query_def by blast
        then have A ⊆ V using ‹(A, X) ∈ Qn› included_def by fastforce
        then show ?thesis
          by (metis Set.is_empty_def UnE assms(1) assms(2) calculation disjoint_iff_not_equal
              getIJProperties(5) subsetD subsetI)
      qed
      then have (A, X) ∈ Q_I_neg using ‹(A, X) = x› ‹x ∈ Qn − Q_I_neg› assms(3) by auto
      then show False using ‹(A, X) = x› ‹x ∈ Qn − Q_I_neg› by blast
    qed
    then show x ∈ ?A using ‹(A, X) = x› ‹x ∈ Qn − Q_I_neg›
      by (metis Diff_subset subset_Q_neg assms(4) fst_conv rwf_query_def set_filterQuery)
  qed
  then show ?thesis by auto
qed

lemma vars_genericJoin:
  assumes card V ≥ 2
  assumes (I, J) = getIJ Q Qn V
  assumes Q_I_pos = projectQuery I (filterQuery I Q)
  assumes Q_I_neg = filterQueryNeg I Qn
  assumes R_I = genericJoin I Q_I_pos Q_I_neg
  assumes Q_J_neg = filterQuery J (Qn − Q_I_neg)
  assumes Q_J_pos = filterQuery J Q
  assumes X = {(t, genericJoin J (newQuery J Q_J_pos (I, t)) (newQuery J Q_J_neg (I, t))) | t . t
∈ R_I}
  assumes R = (⋃(t, x) ∈ X. {merge xx t | xx . xx ∈ x})
  assumes rwf_query n V Q Qn
  shows R = genericJoin V Q Qn
proof −
  have filterQuery J (Qn − Q_I_neg) = Qn − Q_I_neg
    using assms(1) assms(10) assms(2) assms(4) filter_Q_J_neg_same by blast
  then have Q_J_neg = Qn − Q_I_neg by (simp add: assms(6))
  moreover have genericJoin V Q Qn =
    (if card V ≤ 1 then
      (⋂(_, x) ∈ Q. x) − (⋃(_, x) ∈ Qn. x)
    else
      let (I, J) = getIJ Q Qn V in
      let Q_I_pos = projectQuery I (filterQuery I Q) in
      let Q_I_neg = filterQueryNeg I Qn in
      let R_I = genericJoin I Q_I_pos Q_I_neg in
      let Q_J_neg = Qn − Q_I_neg in
      let Q_J_pos = filterQuery J Q in
       let X = {(t, genericJoin J (newQuery J Q_J_pos (I, t)) (newQuery J Q_J_neg (I, t))) | t . t ∈
R_I} in
      (⋃(t, x) ∈ X. {merge xx t | xx . xx ∈ x}))
    by simp
  moreover have ¬ (card V ≤ 1) using assms(1) by linarith
  then have gen: genericJoin V Q Qn = (let (I, J) = getIJ Q Qn V in
      let Q_I_pos = projectQuery I (filterQuery I Q) in
      let Q_I_neg = filterQueryNeg I Qn in
```

```
    let R_I = genericJoin I Q_I_pos Q_I_neg in
    let Q_J_neg = Qn − Q_I_neg in
    let Q_J_pos = filterQuery J Q in
    let X = {(t, genericJoin J (newQuery J Q_J_pos (I, t)) (newQuery J Q_J_neg (I, t))) | t . t ∈
R_I} in
    (⋃(t, x) ∈ X. {merge xx t | xx . xx ∈ x}))
  using assms by simp
 then have ... = (
    let Q_I_pos = projectQuery I (filterQuery I Q) in
    let Q_I_neg = filterQueryNeg I Qn in
    let R_I = genericJoin I Q_I_pos Q_I_neg in
    let Q_J_neg = Qn − Q_I_neg in
    let Q_J_pos = filterQuery J Q in
    let X = {(t, genericJoin J (newQuery J Q_J_pos (I, t)) (newQuery J Q_J_neg (I, t))) | t . t ∈
R_I} in
    (⋃(t, x) ∈ X. {merge xx t | xx . xx ∈ x}))
  using assms(2) by (metis (no_types, lifting) case_prod_conv)
 then show ?thesis using assms by (metis calculation(1) gen)
qed

lemma base_genericJoin:
 assumes card V ≤ 1
 shows genericJoin V Q Qn = (⋂(_, x) ∈ Q. x) − (⋃(_, x) ∈ Qn. x)
proof −
 have genericJoin V Q Qn =
  (if card V ≤ 1 then
    (⋂(_, x) ∈ Q. x) − (⋃(_, x) ∈ Qn. x)
  else
    let (I, J) = getIJ Q Qn V in
    let Q_I_pos = projectQuery I (filterQuery I Q) in
    let Q_I_neg = filterQueryNeg I Qn in
    let R_I = genericJoin I Q_I_pos Q_I_neg in

    let Q_J_neg = Qn − Q_I_neg in
    let Q_J_pos = filterQuery J Q in
    let X = {(t, genericJoin J (newQuery J Q_J_pos (I, t)) (newQuery J Q_J_neg (I, t))) | t . t ∈
R_I} in
    (⋃(t, x) ∈ X. {merge xx t | xx . xx ∈ x}))
  by simp
 then show ?thesis using assms by auto
qed

lemma wf_genericJoin:
 ⟦rwf_query n V Q Qn; card V ≥ 1⟧ ⟹ table n V (genericJoin V Q Qn)
proof (induction V Q Qn rule: genericJoin.induct)
 case (1 V Q Qn)
 then show ?case
 proof (cases card V ≤ 1)
  case True
  then show ?thesis using 1.prems(1) 1.prems(2) le_antisym wf_base_case by blast
 next
  case False
  obtain I J where (I, J) = getIJ Q Qn V by (metis surj_pair)
  define Q_I_pos where Q_I_pos = projectQuery I (filterQuery I Q)
  define Q_I_neg where Q_I_neg = filterQueryNeg I Qn
  define R_I where R_I = genericJoin I Q_I_pos Q_I_neg
  define Q_J_neg where Q_J_neg = filterQuery J (Qn − Q_I_neg)
  define Q_J_pos where Q_J_pos = filterQuery J Q
```

define X **where** $X = \{(t,\ genericJoin\ J\ (newQuery\ J\ Q\_J\_pos\ (I,\ t))\ (newQuery\ J\ Q\_J\_neg\ (I,\ t)))\ |\ t\ .\ t \in R\_I\}$
define R **where** $R = (\bigcup (t,\ x) \in X.\ \{merge\ xx\ t\ |\ xx\ .\ xx \in x\})$
moreover have *card* $V \geq 2$ **using** *False* **by** *auto*
then have $R = genericJoin\ V\ Q\ Qn$
using *vars_genericJoin*[**where** *?V=V* **and** *?I=I* **and** *?J=J* **and** *?Q_I_pos=Q_I_pos* **and** *?Q=Q* **and** *?Qn=Qn* **and**
?Q_I_neg=Q_I_neg **and** *?R_I=R_I* **and** *?Q_J_neg=Q_J_neg* **and** *?Q_J_pos=Q_J_pos*]
using *1.prems(1) Q_I_neg_def Q_I_pos_def Q_J_neg_def Q_J_pos_def R_I_def X_def* ‹$(I,\ J) = getIJ\ Q\ Qn\ V$› *calculation* **by** *blast*
obtain *card* $I \geq 1$ *card* $J \geq 1$
using ‹$(I,\ J) = getIJ\ Q\ Qn\ V$› ‹$2 \leq card\ V$› *getIJ.getIJProperties(1) getIJProperties(2) getIJ_axioms* **by** *blast*
moreover have *rwf_query n I Q_I_pos Q_I_neg*
using *1.prems(1) Q_I_neg_def Q_I_pos_def* ‹$(I,\ J) = getIJ\ Q\ Qn\ V$› ‹$2 \leq card\ V$› *getIJ.wf_firstRecursiveCall getIJ_axioms* **by** *blast*
moreover have $\bigwedge t.\ t \in R\_I \Longrightarrow table\ n\ J\ (genericJoin\ J\ (newQuery\ J\ Q\_J\_pos\ (I,\ t))\ (newQuery\ J\ Q\_J\_neg\ (I,\ t)))$
proof −
fix t **assume** $t \in R\_I$
have *rwf_query n J* $(newQuery\ J\ Q\_J\_pos\ (I,\ t))\ (newQuery\ J\ Q\_J\_neg\ (I,\ t))$
using *1.prems(1) Q_J_neg_def Q_J_pos_def* ‹$(I,\ J) = getIJ\ Q\ Qn\ V$› ‹$2 \leq card\ V$› *getIJ.wf_secondRecursiveCalls getIJ_axioms* **by** *fastforce*
then show $table\ n\ J\ (genericJoin\ J\ (newQuery\ J\ Q\_J\_pos\ (I,\ t))\ (newQuery\ J\ Q\_J\_neg\ (I,\ t)))$
by (*metis 1.IH(2) 1.prems(1) False Q_I_neg_def Q_J_neg_def Q_J_pos_def* ‹$(I,\ J) = getIJ\ Q\ Qn\ V$›
‹$2 \leq card\ V$› *calculation(3) filter_Q_J_neg_same*)
qed
then have $\bigwedge t\ xx.\ t \in R\_I \wedge xx \in (genericJoin\ J\ (newQuery\ J\ Q\_J\_pos\ (I,\ t))\ (newQuery\ J\ Q\_J\_neg\ (I,\ t)))$
$\Longrightarrow wf\_tuple\ n\ V\ (merge\ xx\ t)$
proof −
fix $t\ xx$ **assume** $t \in R\_I \wedge xx \in genericJoin\ J\ (newQuery\ J\ Q\_J\_pos\ (I,\ t))\ (newQuery\ J\ Q\_J\_neg\ (I,\ t))$
have $V = I \cup J$
using ‹$(I,\ J) = getIJ\ Q\ Qn\ V$› ‹$2 \leq card\ V$› *getIJ.coreProperties getIJ_axioms* **by** *metis*
moreover have *wf_tuple n J xx*
using ‹$\bigwedge t.\ t \in R\_I \Longrightarrow table\ n\ J\ (genericJoin\ J\ (newQuery\ J\ Q\_J\_pos\ (I,\ t))\ (newQuery\ J\ Q\_J\_neg\ (I,\ t)))$›
‹$t \in R\_I \wedge xx \in genericJoin\ J\ (newQuery\ J\ Q\_J\_pos\ (I,\ t))\ (newQuery\ J\ Q\_J\_neg\ (I,\ t))$› *table_def* **by** *blast*
moreover have *wf_tuple n I t*
by (*metis 1.IH(1) False Q_I_neg_def Q_I_pos_def*
R_I_def ‹$(I,\ J) = getIJ\ Q\ Qn\ V$› ‹$\bigwedge thesis.\ (\llbracket 1 \leq card\ I;\ 1 \leq card\ J \rrbracket \Longrightarrow thesis) \Longrightarrow thesis$›
‹*rwf_query n I Q_I_pos Q_I_neg*› ‹$t \in R\_I \wedge xx \in genericJoin\ J\ (newQuery\ J\ Q\_J\_pos\ (I,\ t))$
$(newQuery\ J\ Q\_J\_neg\ (I,\ t))$› *table_def*)
then show *wf_tuple n V* (*merge xx t*)
by (*metis calculation(1) calculation(2) sup_commute wf_merge*)
qed
then have $\forall t \in R\_I.\ \forall xx \in (genericJoin\ J\ (newQuery\ J\ Q\_J\_pos\ (I,\ t))\ (newQuery\ J\ Q\_J\_neg\ (I,\ t))).$
wf_tuple n V (*merge xx t*) **by** *blast*
then have $\forall x \in R.\ wf\_tuple\ n\ V\ x$ **using** *R_def X_def* **by** *blast*
then show *?thesis* **using** ‹$R = genericJoin\ V\ Q\ Qn$› *table_def* **by** *blast*
qed
qed

2.2 Correctness

lemma *base_correctness*:
assumes *card V = 1*
assumes *rwf_query n V Q Qn*
assumes *R = genericJoin V Q Qn*
shows *z ∈ genericJoin V Q Qn ⟷ wf_tuple n V z ∧ (∀(A, X)∈Q. restrict A z ∈ X) ∧ (∀(A, X)∈Qn. restrict A z ∉ X)*
proof −
have *z ∈ genericJoin V Q Qn ⟹ wf_tuple n V z ∧ (∀(A, X)∈Q. restrict A z ∈ X) ∧ (∀(A, X)∈Qn. restrict A z ∉ X)*
proof −
fix *z* **assume** *z ∈ genericJoin V Q Qn*
have *wf_tuple n V z* **by** (*meson ‹z ∈ genericJoin V Q Qn› assms(1) assms(2) table_def wf_base_case*)
moreover have *⋀A X. (A, X) ∈ Q ⟹ restrict A z ∈ X*
proof −
fix *A X* **assume** *(A, X) ∈ Q*
have *A = V*
proof −
have *card A ≥ 1*
using *‹(A, X) ∈ Q› assms(2) non_empty_query_def rwf_query_def* **by** *fastforce*
moreover have *A ⊆ V*
using *‹(A, X) ∈ Q› assms(2) included_def rwf_query_def* **by** *fastforce*
then show *?thesis*
by (*metis One_nat_def assms(1) calculation card.infinite card_seteq nat.simps(3)*)
qed
then have *restrict A z = z* **using** *calculation restrict_idle* **by** *blast*
moreover have *z ∈ (⋂(_, x) ∈ Q. x)*
using *‹z ∈ genericJoin V Q Qn› assms(1)* **by** *auto*
then have *z ∈ X* **using** *INT_D ‹(A, X) ∈ Q› case_prod_conv* **by** *auto*
then show *restrict A z ∈ X* **using** *calculation* **by** *auto*
qed
moreover have *⋀A X. (A, X) ∈ Qn ⟹ restrict A z ∉ X*
proof −
fix *A X* **assume** *(A, X) ∈ Qn*
have *card A ≥ 1* **using** *‹(A, X) ∈ Qn› assms(2) non_empty_query_def rwf_query_def* **by** *fastforce*
moreover have *A ⊆ V* **using** *‹(A, X) ∈ Qn› assms(2) included_def rwf_query_def* **by** *blast*
then have *A = V* **by** (*metis assms(1) calculation card_gt_0_iff card_seteq zero_less_one*)
then have *restrict A z = z* **using** *‹wf_tuple n V z› restrict_idle* **by** *blast*
moreover have *z ∉ (⋃(_, x) ∈ Qn. x)*
proof −
have *z ∈ (⋂(_, x) ∈ Q. x) − (⋃(_, x) ∈ Qn. x)*
using *‹z ∈ genericJoin V Q Qn› assms(1)* **by** *auto*
then show *?thesis* **by** (*metis DiffD2*)
qed
then show *restrict A z ∉ X* **using** *UN_iff ‹(A, X) ∈ Qn› calculation(2) prod.sel(2) snd_def* **by** *auto*
qed
then show *wf_tuple n V z ∧ (∀(A, X)∈Q. restrict A z ∈ X) ∧ (∀(A, X)∈Qn. restrict A z ∉ X)*
using *calculation(1) calculation(2)* **by** *blast*
qed
moreover have *wf_tuple n V z ∧ (∀(A, X)∈Q. restrict A z ∈ X) ∧ (∀(A, X)∈Qn. restrict A z ∉ X) ⟹ z ∈ genericJoin V Q Qn*
proof −
assume *wf_tuple n V z ∧ (∀(A, X)∈Q. restrict A z ∈ X) ∧ (∀(A, X)∈Qn. restrict A z ∉ X)*
have *genericJoin V Q Qn = (⋂(_, x) ∈ Q. x) − (⋃(_, x) ∈ Qn. x)* **by** (*simp add*: *assms(1)*)
moreover have *∀(A, X)∈Q. restrict A z = z*
by (*metis* (*mono_tags*, *lifting*) *One_nat_def ‹wf_tuple n V z ∧ (∀(A, X)∈Q. restrict A z ∈ X) ∧ (∀(A, X)∈Qn. restrict A z ∉ X)›*

assms(1) assms(2) card.infinite card_seteq case_prod_beta′ included_def nat.simps(3) non_empty_query_def restrict_idle rwf_query_def)
moreover have *card Q ≥ 1* **using** *assms(2) rwf_query_def wf_query_def* **by** *blast*
moreover have $z \notin (\bigcup (\_, x) \in Qn.\ x)$
proof −
have $\forall (\_, x) \in Qn.\ z \notin x$
by (*metis* (*mono_tags*, *lifting*) *One_nat_def* ‹*wf_tuple n V z* ∧ (∀ (*A*, *X*)∈*Q*. *restrict A z* ∈ *X*) ∧ (∀ (*A*, *X*)∈*Qn*. *restrict A z* ∉ *X*)›
assms(1) assms(2) card.infinite card_seteq case_prod_beta′ included_def nat.simps(3) non_empty_query_def restrict_idle rwf_query_def)
then show *?thesis* **using** *UN_iff case_prod_beta′* **by** *auto*
qed
moreover have $z \in (\bigcap (\_, x) \in Q.\ x)$
proof −
have $\forall (\_, x) \in Q.\ z \in x$
using ‹*wf_tuple n V z* ∧ (∀ (*A*, *X*)∈*Q*. *restrict A z* ∈ *X*) ∧ (∀ (*A*, *X*)∈*Qn*. *restrict A z* ∉ *X*)› *calculation(2)* **by** *fastforce*
then show *?thesis* **using** *INT_I case_prod_beta′* **by** *auto*
qed
ultimately show *?thesis*
proof −
have *genericJoin V Q Qn* ⊆ *R*
using *assms(3)* **by** *blast*
then have $(\bigcap (N, Z) \in Q.\ Z) - (\bigcup (N, Z) \in Qn.\ Z) \subseteq R$
by (*metis* ‹*genericJoin V Q Qn* = $(\bigcap (\_, x) \in Q.\ x) - (\bigcup (\_, x) \in Qn.\ x)$›)
then have ∃ *Z Za*. *Z* − *Za* ⊆ *R* ∧ *z* ∉ *Za* ∧ *z* ∈ *Z*
by (*metis* ‹*z* ∈ $(\bigcap (\_, x) \in Q.\ x)$› ‹*z* ∉ $(\bigcup (\_, x) \in Qn.\ x)$›)
then show *?thesis*
using *assms(3)* **by** *blast*
qed
qed
then show *?thesis* **using** *calculation* **by** *linarith*
qed

lemma *simple_list_index_equality*:
assumes *length a* = *n*
assumes *length b* = *n*
assumes ∀ *i* < *n*. *a*!*i* = *b*!*i*
shows *a* = *b*
using *assms(1) assms(2) assms(3) nth_equalityI* **by** *force*

lemma *simple_restrict_none*:
assumes *i* < *length X*
assumes *i* ∉ *A*
shows (*restrict A X*)!*i* = *None*
by (*simp add*: *assms(1) assms(2) restrict_def*)

lemma *simple_restrict_some*:
assumes *i* < *length X*
assumes *i* ∈ *A*
shows (*restrict A X*)!*i* = *X*!*i*
by (*simp add*: *assms(1) assms(2) restrict_def*)

lemma *merge_restrict*:
assumes *A* ∩ *J* = {}
assumes *A* ⊆ *I*
assumes *length xx* = *n*
assumes *length t* = *n*

```
  assumes restrict J xx = xx
  shows restrict A (merge xx t) = restrict A t
proof −
  have ⋀i. i < n ⟹ (restrict A (merge xx t))!i = (restrict A t)!i
  proof −
    fix i assume i < n
    show (restrict A (merge xx t))!i = (restrict A t)!i
    proof (cases i ∈ A)
      case True
      have (restrict A t)!i = t!i by (simp add: True ‹i < n› assms(4) nth_restrict)
      moreover have (restrict A (merge xx t))!i = t!i
      proof −
        have xx!i = None
        by (metis True ‹i < n› assms(1) assms(3) assms(5) disjoint_iff_not_equal simple_restrict_none)
        obtain length xx = length t by (simp add: assms(3) assms(4))
        moreover have (merge xx t)!i = merge_option (xx!i, t!i)
          using ‹i < n› ‹length xx = length t› assms(3) by auto
        moreover have merge_option (None, t!i) = t!i
          by (metis merge_option.simps(1) merge_option.simps(3) option.exhaust)
        then have (merge xx t)!i = t!i using ‹xx ! i = None› calculation(2) by auto
        moreover have (restrict A (merge xx t))!i = (merge xx t)!i
        proof −
          have length (zip xx t) = n using assms(3) calculation(1) by auto
          then have length (merge xx t) = n by simp
          then show ?thesis by (simp add: True ‹i < n› nth_restrict)
        qed
        then show ?thesis using calculation(3) by auto
      qed
      then show ?thesis by (simp add: calculation)
    next
      case False
      have (restrict A t)!i = None by (simp add: False ‹i < n› assms(4) restrict_def)
      obtain length xx = n and length t = n
        by (simp add: assms(3) assms(4))
      then have length (merge xx t) = n by simp
      moreover have (restrict A (merge xx t))!i = None
        using False ‹i < n› calculation simple_restrict_none by blast
      then show ?thesis by (simp add: ‹restrict A t ! i = None›)
    qed
  qed
  then have ∀ i < n. (restrict A (merge xx t))!i = (restrict A t)!i by blast
 then show ?thesis using simple_list_index_equality[where ?a=restrict A (merge xx t) and ?b=restrict
A t and ?n=n]
      assms(3) assms(4) by simp
qed

lemma restrict_idle_include:
  assumes wf_tuple n A v
  assumes A ⊆ I
  shows restrict I v = v
proof −
  have ⋀i. i < length v ⟹ (restrict I v)!i = v!i
  proof −
    fix i assume i < length v
    show (restrict I v)!i = v!i
    proof (cases i ∈ A)
      case True
      then show ?thesis using ‹i < length v› assms(2) nth_restrict by blast
```

```
    next
      case False
    then show ?thesis by (metis ‹i < length v› assms(1) nth_restrict simple_restrict_none wf_tuple_def)
    qed
  qed
  then show ?thesis by (simp add: list_eq_iff_nth_eq)
qed

lemma merge_index:
  assumes I ∩ J = {}
  assumes wf_tuple n I tI
  assumes wf_tuple n J tJ
  assumes t = merge tI tJ
  assumes i < n
  shows (i ∈ I ∧ t!i = tI!i) ∨ (i ∈ J ∧ t!i = tJ!i) ∨ (i ∉ I ∧ i ∉ J ∧ t!i = None)
proof −
  have t!i = merge_option ((zip tI tJ)!i)
    by (metis (full_types) assms(2) assms(3) assms(4) assms(5) length_zip merge.simps
        min_less_iff_conj nth_map wf_tuple_def)
 then have t!i = merge_option (tI!i, tJ!i) by (metis assms(2) assms(3) assms(5) nth_zip wf_tuple_def)
  then show ?thesis
  proof (cases i ∈ I)
    case True
    have t!i = tI!i
    proof −
    have tJ!i = None by (meson True assms(1) assms(3) assms(5) disjoint_iff_not_equal wf_tuple_def)
      moreover have merge_option (tI!i, None) = tI!i
        by (metis True assms(2) assms(5) merge_option.simps(2) option.exhaust wf_tuple_def)
      then show ?thesis by (simp add: ‹t ! i = merge_option (tI ! i, tJ ! i)› calculation)
    qed
    then show ?thesis using True by blast
  next
    case False
    have i ∉ I by (simp add: False)
    then show ?thesis
    proof (cases i ∈ J)
      case True
      have t!i = tJ!i
      proof −
        have tI!i = None using False assms(2) assms(5) wf_tuple_def by blast
        moreover have merge_option (None, tJ!i) = tJ!i
          by (metis True assms(3) assms(5) merge_option.simps(3) option.exhaust wf_tuple_def)
        then show ?thesis by (simp add: ‹t ! i = merge_option (tI ! i, tJ ! i)› calculation)
      qed
      then show ?thesis using True by blast
    next
      case False
       obtain tI!i = None and tJ!i = None  by (meson False ‹i ∉ I› assms(2) assms(3) assms(5)
wf_tuple_def)
      have t!i = None
        by (simp add: ‹t ! i = merge_option (tI ! i, tJ ! i)› ‹tI ! i = None› ‹tJ ! i = None›)
      then show ?thesis using False ‹i ∉ I› by blast
    qed
  qed
qed

lemma restrict_index_in:
  assumes i < length X
```

assumes $i \in I$
shows $(restrict\ I\ X)!i = X!i$
by (*simp add*: *assms*(*1*) *assms*(*2*) *nth_restrict*)

lemma *restrict_index_out*:
assumes $i < length\ X$
assumes $i \notin I$
shows $(restrict\ I\ X)!i = None$
by (*simp add*: *assms*(*1*) *assms*(*2*) *simple_restrict_none*)

lemma *merge_length*:
assumes $length\ a = n$
assumes $length\ b = n$
shows $length\ (merge\ a\ b) = n$
by (*simp add*: *assms*(*1*) *assms*(*2*))

lemma *real_restrict_merge*:
assumes $I \cap J = \{\}$
assumes *wf_tuple n I tI*
assumes *wf_tuple n J tJ*
shows $restrict\ I\ (merge\ tI\ tJ) = restrict\ I\ tI \wedge restrict\ J\ (merge\ tI\ tJ) = restrict\ J\ tJ$
proof −
have $length\ (merge\ tI\ tJ) = n$
using *assms*(*2*) *assms*(*3*) *merge_length wf_tuple_def* **by** *blast*
have $\bigwedge i.\ i < n \Longrightarrow (restrict\ I\ (merge\ tI\ tJ))!i = (restrict\ I\ tI)!i$
$\wedge\ (restrict\ J\ (merge\ tI\ tJ))!i = (restrict\ J\ tJ)!i$
proof −
fix i **assume** $i < n$
show $(restrict\ I\ (merge\ tI\ tJ))!i = (restrict\ I\ tI)!i \wedge (restrict\ J\ (merge\ tI\ tJ))!i = (restrict\ J\ tJ)!i$
proof (*cases* $i \in I$)
case *True*
have $(merge\ tI\ tJ)!i = tI!i$
by (*meson True* ‹$i < n$› *assms*(*1*) *assms*(*2*) *assms*(*3*) *disjoint_iff_not_equal merge_index*)
then have $(restrict\ I\ (merge\ tI\ tJ))!i = tI!i$
by (*metis True* ‹$i < n$› ‹$length\ (merge\ tI\ tJ) = n$› *simple_restrict_some*)
then show *?thesis*
by (*metis True* ‹$i < n$› ‹$length\ (merge\ tI\ tJ) = n$› *assms*(*1*) *assms*(*2*) *assms*(*3*) *disjoint_iff_not_equal restrict_idle simple_restrict_none wf_tuple_def*)
next
case *False*
have $i \notin I$ **by** (*simp add*: *False*)
then show *?thesis*
proof (*cases* $i \in J$)
case *True*
have $(merge\ tI\ tJ)!i = tJ!i$
using *True* ‹$i < n$› *assms*(*1*) *assms*(*2*) *assms*(*3*) *merge_index* **by** *blast*
then show *?thesis*
by (*metis* (*no_types*, *lifting*) *False* ‹$i < n$› ‹$length\ (merge\ tI\ tJ) = n$› *assms*(*2*) *assms*(*3*) *simple_restrict_none simple_restrict_some wf_tuple_def*)
next
case *False*
have $(merge\ tI\ tJ)!i = None$ **using** *False* ‹$i < n$› ‹$i \notin I$› *assms*(*1*) *assms*(*2*) *assms*(*3*) *merge_index* **by** *blast*
then show *?thesis*
by (*metis False* ‹$i < n$› ‹$i \notin I$› ‹$length\ (merge\ tI\ tJ) = n$› *assms*(*2*) *assms*(*3*) *eq_iff equalityD1 restrict_idle_include simple_restrict_none wf_tuple_def wf_tuple_restrict_simple*)
qed
qed

```
  qed
  then obtain ∀ i < n. (restrict I (merge tI tJ))!i = (restrict I tI)!i
        and ∀ i < n. (restrict J (merge tI tJ))!i = (restrict J tJ)!i by blast
  moreover have length (merge tI tJ) = n by (meson assms(2) assms(3) wf_merge wf_tuple_def)
  moreover obtain length (restrict I tI) = n and length (restrict J tJ) = n
    using assms(2) assms(3) wf_tuple_def by auto
  then show ?thesis
    by (metis ‹⋀i. i < n ⟹ restrict I (merge tI tJ) ! i = restrict I tI ! i ∧ restrict J (merge tI tJ) ! i
= restrict J tJ ! i› calculation(3) length_restrict simple_list_index_equality)
qed

lemma simple_set_image_id:
  assumes ∀ x∈X. f x = x
  shows Set.image f X = X
proof −
  have Set.image f X = {f x |x. x ∈ X} by (simp add: Setcompr_eq_image)
  then have ... = {x |x. x ∈ X} by (simp add: assms)
  moreover have ... = X by simp
  then show ?thesis by (simp add: ‹f ‘ X = {f x |x. x ∈ X}› calculation)
qed

lemma nested_include_restrict:
  assumes restrict I z = t
  assumes A ⊆ I
  shows restrict A z = restrict A t
proof −
  have length (restrict A z) = length (restrict A t) using assms(1) by auto
  moreover have ⋀i. i < length (restrict A z) ⟹ (restrict A z) ! i = (restrict A t) ! i
  proof −
    fix i assume i < length (restrict A z)
    then show (restrict A z) ! i = (restrict A t) ! i
    proof (cases i ∈ A)
      case True
      then show ?thesis
       by (metis restrict_index_in ‹i < length (restrict A z)› assms(1) assms(2) length_restrict subsetD)
    next
      case False
      then show ?thesis
        by (metis simple_restrict_none ‹i < length (restrict A z)› calculation length_restrict)
    qed
  qed
  ultimately show ?thesis by (simp add: list_eq_iff_nth_eq)
qed

lemma restrict_nested:
  restrict A (restrict B x) = restrict (A ∩ B) x (is ?lhs = ?rhs)
proof −
  have ⋀i. i < length x ⟹ ?lhs!i = ?rhs!i
    by (metis Int_iff length_restrict restrict_index_in simple_restrict_none)
  then show ?thesis by (simp add: simple_list_index_equality)
qed

lemma newQuery_equi_dev:
  newQuery V Q (I, t) = Set.image (projectTable V) (Set.image (λtab. semiJoin tab (I, t)) Q)
  by (metis newQuery_equiv_def projectQuery.elims)

lemma projectTable_idle:
  assumes table n A X
```

```
  assumes A ⊆ I
  shows projectTable I (A, X) = (A, X)
proof −
  have projectTable I (A, X) = (A ∩ I, Set.image (restrict I) X)
        using projectTable.simps by blast
  then have A ∩ I = A using assms(2) by blast
  have ⋀x. x ∈ X ⟹ (restrict I) x = x
  proof −
    fix x assume x ∈ X
    have wf_tuple n A x using ‹x ∈ X› assms(1) table_def by blast
    then show (restrict I) x = x using assms(2) restrict_idle_include by blast
  qed
  then have ∀ x ∈ X. (restrict I) x = x by blast
  moreover have Set.image (restrict I) X = X
    by (simp add: ‹⋀x. x ∈ X ⟹ restrict I x = x›)
  then show ?thesis by (simp add: ‹A ∩ I = A›)
qed

lemma restrict_partition_merge:
  assumes I ∪ J = V
  assumes wf_tuple n V z
  assumes xx = restrict J z
  assumes t = restrict I z
  assumes Set.is_empty (I ∩ J)
  shows z = merge xx t
proof −
  have ⋀i. i < n ⟹ z!i = (merge xx t)!i
  proof −
    fix i assume i < n
    show z!i = (merge xx t)!i
    proof (cases i ∈ I)
      case True
      have z!i = t!i
        by (metis True ‹i < n› assms(2) assms(4) nth_restrict wf_tuple_def)
      moreover have (merge xx t)!i = t!i
      proof −
        have xx ! i = None
          by (metis simple_restrict_none Set.is_empty_def True ‹i < n› assms(2) assms(3) assms(5)
disjoint_iff_not_equal wf_tuple_length)
        moreover have (merge xx t) ! i = merge_option (xx ! i, t ! i) using ‹i < n› assms(2) assms(3)
assms(4) wf_tuple_length by fastforce
        ultimately show ?thesis
        proof (cases t ! i)
          case None
          then show ?thesis using ‹merge xx t ! i = merge_option (xx ! i, t ! i)› ‹xx ! i = None› by auto
        next
          case (Some a)
          then show ?thesis using ‹merge xx t ! i = merge_option (xx ! i, t ! i)› ‹xx ! i = None› by auto
        qed
      qed
      then show ?thesis by (simp add: calculation)
    next
      case False
      have i ∉ I by (simp add: False)
      then show ?thesis
      proof (cases i ∈ J)
        case True
        have z!i = xx!i
```

```
        by (metis True ‹i < n› assms(2) assms(3) nth_restrict wf_tuple_def)
       moreover have (merge xx t)!i = xx!i
       proof (cases xx ! i)
        case None
        then show ?thesis  by (metis True UnI1 ‹i < n› assms(1) assms(2) calculation sup_commute
wf_tuple_def)
       next
        case (Some a)
      have t ! i = None by (metis False simple_restrict_none ‹i < n› assms(2) assms(4) wf_tuple_length)
       then show ?thesis using Some ‹i < n› assms(2) assms(3) assms(4) wf_tuple_length by fastforce
       qed
       then show ?thesis by (simp add: calculation)
     next
       case False
       have z!i = None by (metis False UnE ‹i < n› ‹i ∉ I› assms(1) assms(2) wf_tuple_def)
       moreover have (merge xx t)!i = None
       proof −
        have xx ! i = None
         by (metis False New_max.simple_restrict_none ‹i < n› assms(2) assms(3) wf_tuple_length)
        moreover have t ! i = None
         by (metis New_max.simple_restrict_none ‹i < n› ‹i ∉ I› assms(2) assms(4) wf_tuple_length)
          ultimately show ?thesis using ‹i < n› assms(2) assms(3) assms(4) wf_tuple_length by
fastforce
       qed
       then show ?thesis by (simp add: calculation)
     qed
   qed
  qed
  moreover have length z = n using assms(2) wf_tuple_def by blast
  then show ?thesis
   by (simp add: assms(3) assms(4) calculation simple_list_index_equality)
qed

lemma restrict_merge:
  assumes zI = restrict I z
  assumes zJ = restrict J z
  assumes restrict (A ∩ I) zI ∈ Set.image (restrict I) X
  assumes restrict (A ∩ J) zJ ∈ Set.image (restrict J) (Set.filter (isSameIntersection zI (A ∩ I)) X)
  assumes z = merge zJ zI
  assumes table n A X
  assumes A ⊆ I ∪ J
  assumes card (A ∩ I) ≥ 1
  assumes wf_set n (I ∪ J)
  assumes wf_tuple n (I ∪ J) z
  shows restrict A z ∈ X
proof −
  define zAJ where zAJ = restrict (A ∩ J) zJ
  obtain zz where zAJ = restrict J zz isSameIntersection zI (A ∩ I) zz zz ∈ X
   using assms(4) zAJ_def by auto
  then have restrict (A ∩ I) zz = restrict A zI
  proof −
   have restrict (A ∩ I) zI = restrict (A ∩ I) zz
   proof −
    have wf_set n A using assms(7) assms(9) wf_set_def by auto
    moreover have wf_tuple n I zI using assms(1) assms(10) wf_tuple_restrict_simple by auto
    moreover have wf_tuple n A zz using ‹zz ∈ X› assms(6) table_def by blast
    moreover obtain A ∩ I ⊆ A A ∩ I ⊆ I by simp
    then show ?thesis using isSame_equi_dev[of n _ I zI A zz A ∩ I]
```

```
      using ‹isSameIntersection zI (A ∩ I) zz› assms(7) assms(9) calculation(2) calculation(3) by blast
    qed
    then show ?thesis
      by (simp add: restrict_nested assms(1))
  qed
  then have zz = restrict A z
  proof −
    have length zz = n using ‹zz ∈ X› assms(6) table_def wf_tuple_def by blast
    moreover have length (restrict A z) = n
      by (metis ‹restrict (A ∩ I) zz = restrict A zI› assms(1) calculation length_restrict)
    moreover have ⋀i. i < n ⟹ zz!i = (restrict A z)!i
    proof −
      fix i assume i < n
      show zz!i = (restrict A z)!i
      proof (cases i ∈ A)
        case True
        have i ∈ A using True by simp
        then show ?thesis
        proof (cases i ∈ I)
          case True
          have zz!i = (restrict (A ∩ I) zz)!i
            by (simp add: True ‹i < n› ‹i ∈ A› calculation(1) restrict_index_in)
          then have ... = (restrict A zI)!i by (simp add: ‹restrict (A ∩ I) zz = restrict A zI›)
          then show ?thesis
            by (metis True ‹i < n› ‹i ∈ A› ‹zz ! i = restrict (A ∩ I) zz ! i› assms(1) calculation(2)
                length_restrict restrict_index_in)
        next
          case False
          have zz!i = (restrict (A ∩ J) zJ)!i
            by (metis False True UnE ‹i < n› ‹zAJ = restrict J zz› assms(7) calculation(1)
                restrict_index_in subsetD zAJ_def)
          then have ... = (restrict A zJ)!i by (simp add: assms(2) restrict_nested)
          then show ?thesis
            by (metis False True UnE ‹i < n› ‹zz ! i = restrict (A ∩ J) zJ ! i› assms(2) assms(7)
                calculation(2) length_restrict restrict_index_in subsetD)
        qed
      next
        case False
        then show ?thesis
        by (metis ‹i < n› ‹zz ∈ X› assms(6) calculation(2) length_restrict simple_restrict_none table_def
wf_tuple_def)
      qed
    qed
    then show ?thesis using calculation(1) calculation(2) simple_list_index_equality by blast
  qed
  then show ?thesis
    using ‹zz ∈ X› by auto
qed

lemma partial_correctness:
  assumes V = I ∪ J
  assumes Set.is_empty (I ∩ J)
  assumes card I ≥ 1
  assumes card J ≥ 1
  assumes Q_I_pos = projectQuery I (filterQuery I Q)
  assumes Q_J_pos = filterQuery J Q
  assumes Q_I_neg = filterQueryNeg I Qn
  assumes Q_J_neg = filterQuery J (Qn − Q_I_neg)
```

assumes *NQ_pos* = *newQuery J Q_J_pos* (*I*, *t*)
assumes *NQ_neg* = *newQuery J Q_J_neg* (*I*, *t*)
assumes *R_NQ* = *genericJoin J NQ_pos NQ_neg*
assumes ∀*x*. (*x* ∈ *R_I* ⟷ *wf_tuple n I x* ∧ (∀(*A*, *X*)∈*Q_I_pos*. *restrict A x* ∈ *X*) ∧ (∀(*A*, *X*)∈*Q_I_neg*. *restrict A x* ∉ *X*))
assumes ∀*y*. (*y* ∈ *R_NQ* ⟷ *wf_tuple n J y* ∧ (∀(*A*, *X*)∈*NQ_pos*. *restrict A y* ∈ *X*) ∧ (∀(*A*, *X*)∈*NQ_neg*. *restrict A y* ∉ *X*))
assumes *z* = *merge xx t*
assumes *t* ∈ *R_I*
assumes *xx* ∈ *R_NQ*
assumes *rwf_query n V Q Qn*
shows *wf_tuple n V z* ∧ (∀(*A*, *X*)∈*Q*. *restrict A z* ∈ *X*) ∧ (∀(*A*, *X*)∈*Qn*. *restrict A z* ∉ *X*)
proof −
obtain *wf_tuple n I t wf_tuple n J xx*
using *assms*(*12*) *assms*(*13*) *assms*(*15*) *assms*(*16*) **by** *blast*
then have *wf_tuple n V z*
by (*metis wf_merge assms*(*1*) *assms*(*14*) *sup_commute*)
moreover have ⋀*A X*. (*A*, *X*) ∈ *Qn* ⟹ *restrict A z* ∉ *X*
proof −
fix *A X* **assume** (*A*, *X*) ∈ *Qn*
have *restrict I* (*merge xx t*) = *restrict I t*
by (*metis* (*no_types*, *lifting*) *Set.is_empty_def* ‹⋀*thesis*. (⟦*wf_tuple n I t*; *wf_tuple n J xx*⟧ ⟹ *thesis*) ⟹ *thesis*›
assms(*2*) *merge_restrict restrict_idle sup.cobounded1 wf_tuple_def*)
moreover have *restrict J* (*merge xx t*) = *restrict J xx*
by (*metis Set.is_empty_def* ‹⋀*thesis*. (⟦*wf_tuple n I t*; *wf_tuple n J xx*⟧ ⟹ *thesis*) ⟹ *thesis*›
assms(*2*) *inf_commute real_restrict_merge*)
moreover have *restrict J xx* = *xx* **using** ‹*wf_tuple n J xx*› *restrict_idle* **by** *auto*
moreover have *restrict I t* = *t* **using** ‹*wf_tuple n I t*› *restrict_idle* **by** *auto*
then obtain *restrict I z* = *t restrict J z* = *xx*
using *assms*(*14*) *calculation*(*1*) *calculation*(*2*) *calculation*(*3*) **by** *auto*
moreover have ∀(*A*, *X*)∈*Q_I_pos*. *restrict A t* ∈ *X* **using** *assms*(*12*) *assms*(*15*) **by** *blast*
moreover have ∀(*A*, *X*)∈*NQ_pos*. *restrict A xx* ∈ *X* **using** *assms*(*13*) *assms*(*16*) **by** *blast*
moreover have *card A* ≥ *1*
using ‹(*A*, *X*) ∈ *Qn*› *assms*(*17*) *non_empty_query_def rwf_query_def* **by** *fastforce*
then show *restrict A z* ∉ *X*
proof (*cases A* ⊆ *I*)
case *True*
have (*A*, *X*) ∈ *Q_I_neg* **by** (*simp add*: *True* ‹(*A*, *X*) ∈ *Qn*› *assms*(*7*))
have *table n A X*
proof −
have *wf_query n V Q Qn* **using** *assms*(*17*) *rwf_query_def* **by** *blast*
moreover have (*A*, *X*) ∈ (*Q* ∪ *Qn*) **by** (*simp add*: ‹(*A*, *X*) ∈ *Qn*›)
then show *?thesis* **by** (*metis calculation fst_conv snd_conv wf_atable_def wf_query_def*)
qed
then have *restrict A t* ∉ *X* **using** ‹(*A*, *X*) ∈ *Q_I_neg*› *assms*(*12*) *assms*(*15*) **by** *blast*
moreover have *restrict A z* = *restrict A t* **using** *True* ‹*restrict I z* = *t*› *nested_include_restrict* **by** *blast*
then show *?thesis* **by** (*simp add*: *calculation*)
next
case *False*
have (*A*, *X*) ∈ *Q_J_neg*
proof −
have (*A*, *X*) ∈ *Qn* − *Q_I_neg* **using** *False* ‹(*A*, *X*) ∈ *Qn*› *assms*(*7*) **by** *auto*
moreover have *card* (*A* ∩ *J*) ≥ *1*
by (*metis* (*no_types*, *lifting*) *False Int_greatest One_nat_def Set.is_empty_def Suc_leI Suc_le_lessD* ‹(*A*, *X*) ∈ *Qn*› *assms*(*1*) *assms*(*17*) *assms*(*2*) *assms*(*4*) *card_gt_0_iff case_prodD finite_Int included_def rwf_query_def sup_ge2 sup_inf_absorb sup_inf_distrib1*)

```
      then show ?thesis using assms(8) calculation
        by (metis Diff_subset subset_Q_neg assms(17) fst_conv rwf_query_def set_filterQuery)
    qed
    define AI where AI = A ∩ I
    define AJ where AJ = A ∩ J
    then have NQ_neg = projectQuery J (Set.image (λtab. semiJoin tab (I, t)) Q_J_neg)
     by (metis newQuery_equi_dev projectQuery.simps assms(10))
    then obtain XX where (A, XX) = (λtab. semiJoin tab (I, t)) (A, X) by simp
    then obtain XXX where (AJ, XXX) ∈ NQ_neg and (AJ, XXX) = projectTable J (A, XX)
     by (metis AJ_def newQuery.simps projectTable.simps ‹(A, X) ∈ Q_J_neg› assms(10) image_eqI)
    then have restrict AJ xx ∉ XXX
    proof −
      have xx ∈ R_NQ by (simp add: assms(16))
     then have wf_tuple n J xx ∧ (∀(A, X)∈NQ_pos. restrict A xx ∈ X) ∧ (∀(A, X)∈NQ_neg. restrict
A xx ∉ X)
       by (simp add: assms(13))
      then show ?thesis using ‹(AJ, XXX) ∈ NQ_neg› by blast
    qed

    define zA where zA = restrict A z
    have zA ∉ X
    proof (rule ccontr)
      assume ¬ (zA ∉ X)
      then have zA ∈ X by simp
      moreover have restrict (A ∩ I) zA = restrict (A ∩ I) t
        by (metis nested_include_restrict ‹restrict I z = t› inf_le1 inf_le2 zA_def)
      then have isSameIntersection t (A ∩ I) zA
      proof −
        have wf_set n V using assms(17) rwf_query_def wf_query_def by blast
        moreover obtain A ∩ I ⊆ A A ∩ I ⊆ I I ⊆ V using assms(1) by blast
          moreover have A ⊆ V using ‹(A, X) ∈ Qn› assms(17) included_def rwf_query_def by
fastforce
        moreover have wf_tuple n A zA
          using ‹wf_tuple n V z› calculation(5) wf_tuple_restrict_simple zA_def by blast
        then show ?thesis using isSame_equi_dev[of n V A zA I t A ∩ I]
            by (simp add: ‹restrict (A ∩ I) zA = restrict (A ∩ I) t› ‹wf_tuple n I t› calculation(1)
calculation(4) calculation(5))
      qed
      then have zA ∈ XX using ‹(A, XX) = semiJoin (A, X) (I, t)› calculation by auto
      then have restrict J zA ∈ XXX using ‹(AJ, XXX) = projectTable J (A, XX)› by auto
      moreover have restrict AJ xx = restrict J zA
        by (metis AJ_def restrict_nested ‹restrict J z = xx› inf.right_idem inf_commute zA_def)
      then show False using ‹restrict AJ xx ∉ XXX› calculation(2) by auto
    qed
    then show ?thesis using zA_def by auto
  qed
 qed
 moreover have ⋀A X. (A, X) ∈ Q ⟹ restrict A z ∈ X
 proof −
   fix A X assume (A, X) ∈ Q
   have restrict I (merge xx t) = restrict I t
     by (metis (no_types, lifting) Set.is_empty_def ‹⋀thesis. (⟦wf_tuple n I t; wf_tuple n J xx⟧ ⟹
thesis) ⟹ thesis›
       assms(2) merge_restrict restrict_idle sup.cobounded1 wf_tuple_def)
   moreover have restrict J (merge xx t) = restrict J xx
    by (metis Set.is_empty_def ‹⋀thesis. (⟦wf_tuple n I t; wf_tuple n J xx⟧ ⟹ thesis) ⟹ thesis›
       assms(2) inf_commute real_restrict_merge)
   moreover have restrict J xx = xx using ‹wf_tuple n J xx› restrict_idle by auto
```

moreover have *restrict I t = t* **using** ‹*wf_tuple n I t*› *restrict_idle* **by** *auto*
then obtain *restrict I z = t restrict J z = xx*
using *assms(14) calculation(1) calculation(2) calculation(3)* **by** *auto*
moreover have ∀(*A*, *X*)∈*Q_I_pos*. *restrict A t* ∈ *X* **using** *assms(12) assms(15)* **by** *blast*
moreover have ∀(*A*, *X*)∈*NQ_pos*. *restrict A xx* ∈ *X* **using** *assms(13) assms(16)* **by** *blast*
moreover have *card A* ≥ *1*
using ‹(*A*, *X*) ∈ *Q*› *assms(17) non_empty_query_def rwf_query_def* **by** *fastforce*
then show *restrict A z* ∈ *X*
proof (*cases A* ⊆ *I*)
case *True*
have (*A*, *X*) ∈ *filterQuery I Q*
proof −
have *A* ∩ *I* = *A* **using** *True* **by** *auto*
then have *A* ∩ *I* ≠ {} **using** ‹*1* ≤ *card A*› **by** *auto*
then have (λ(*s*, _). *s* ∩ *V* ≠ {}) (*A*, *X*) **using** *assms(1)* **by** *blast*
then show *?thesis*
by (*metis* ‹(*A*, *X*) ∈ *Q*› ‹*1* ≤ *card A*› ‹*A* ∩ *I* = *A*› *assms(17) fst_conv rwf_query_def set_filterQuery*)
qed
have *table n A X*
proof −
have *wf_query n V Q Qn* **using** *assms(17) rwf_query_def* **by** *blast*
moreover have (*A*, *X*) ∈ (*Q* ∪ *Qn*) **by** (*simp add*: ‹(*A*, *X*) ∈ *Q*›)
then show *?thesis* **by** (*metis calculation fst_conv snd_conv wf_atable_def wf_query_def*)
qed
moreover have *projectTable I* (*A*, *X*) = (*A*, *X*) **using** *True calculation projectTable_idle* **by** *blast*
then have (*A*, *X*) ∈ *Q_I_pos* **by** (*metis* ‹(*A*, *X*) ∈ *filterQuery I Q*› *assms(5) image_eqI projectQuery.elims*)
then have *restrict A t* ∈ *X* **using** ‹∀(*A*, *X*)∈*Q_I_pos*. *restrict A t* ∈ *X*› **by** *blast*
moreover have *restrict A z = restrict A t* **using** *True* ‹*restrict I z = t*› *nested_include_restrict* **by** *blast*
then show *?thesis* **by** (*simp add*: *calculation(2)*)
next
case *False*
have *A* ⊆ *V*
proof −
have *included V Q* **using** *assms(17) rwf_query_def* **by** *blast*
then show *?thesis* **using** ‹(*A*, *X*) ∈ *Q*› *included_def* **by** *fastforce*
qed
then have *card* (*A* ∩ *J*) ≥ *1* **by** (*metis False One_nat_def Suc_leI Suc_le_lessD UnE assms(1) assms(4) card_gt_0_iff disjoint_iff_not_equal finite_Int subsetD subsetI*)
then show *?thesis*
proof (*cases card* (*A* ∩ *I*) ≥ *1*)
case *True*
define *zI* **where** *zI* = *restrict I z*
define *zJ* **where** *zJ* = *restrict J z*
obtain *zI* = *t zJ* = *xx*
by (*simp add*: *calculation(4) calculation(5) zI_def zJ_def*)
then have *wf_tuple n I zI* ∧ (∀(*A*, *X*)∈*Q_I_pos*. *restrict A zI* ∈ *X*)
using ‹*wf_tuple n I t*› *calculation(6)* **by** *blast*
moreover have *wf_tuple n J zJ* ∧ (∀(*A*, *X*)∈*NQ_pos*. *restrict A zJ* ∈ *X*)
using ‹∀(*A*, *X*)∈*NQ_pos*. *restrict A xx* ∈ *X*› ‹*wf_tuple n J xx*› ‹*zJ = xx*› **by** *blast*
obtain (*A*, *X*) ∈ (*filterQuery I Q*) (*A*, *X*) ∈ *Q_J_pos*
using *True* ‹(*A*, *X*) ∈ *Q*› ‹*1* ≤ *card* (*A* ∩ *J*)› *assms(6) assms(17) rwf_query_def set_filterQuery* **by** *fastforce*
define *AI* **where** *AI* = *A*∩*I*
define *XI* **where** *XI* = *Set.image* (*restrict I*) *X*
then have (*AI*, *XI*) = *projectTable I* (*A*, *X*) **using** *AI_def XI_def* **by** *simp*

```
      then have (AI, XI) ∈ Q_I_pos by (metis ‹(A, X) ∈ filterQuery I Q› assms(5) image_eqI
projectQuery.elims)
    then have restrict AI zI ∈ XI using ‹wf_tuple n I zI ∧ (∀(A, X)∈Q_I_pos. restrict A zI ∈ X)›
by blast
     obtain AJ XJ where (AJ, XJ) = projectTable J (semiJoin (A, X) (I, zI)) by simp
     then have AJ = A ∩ J by auto
     then have (AJ, XJ) ∈ NQ_pos
       using ‹(A, X) ∈ Q_J_pos› ‹(AJ, XJ) = projectTable J (semiJoin (A, X) (I, zI))› ‹zI = t›
image_iff
       using assms(9) by fastforce
     then have restrict AJ zJ ∈ XJ
       using ‹wf_tuple n J zJ ∧ (∀(A, X)∈NQ_pos. restrict A zJ ∈ X)› by blast
     have XJ = Set.image (restrict J) (Set.filter (isSameIntersection zI (A ∩ I)) X)
       using ‹(AJ, XJ) = projectTable J (semiJoin (A, X) (I, zI))› by auto
     then have restrict AJ zJ ∈ Set.image (restrict J) (Set.filter (isSameIntersection zI (A ∩ I)) X)
       using ‹restrict AJ zJ ∈ XJ› by blast
     moreover have table n A X
       using ‹(A, X) ∈ Q› assms(17) rwf_query_def wf_atable_def wf_query_def by fastforce
     moreover have A ⊆ I ∪ J using ‹A ⊆ V› assms(1) by auto
    then show ?thesis using restrict_merge[of zI I z zJ J A X n] AI_def True XI_def ‹AJ = A ∩ J›
        ‹restrict AI zI ∈ XI› ‹restrict I z = t› ‹restrict J z = xx› ‹wf_tuple n V z› assms(1)
       assms(14) assms(17) calculation(2) calculation(3) rwf_query_def wf_query_def zI_def zJ_def
by blast
   next
     case False
     have (A, X) ∈ Q_J_pos using ‹(A, X) ∈ Q› ‹1 ≤ card (A ∩ J)› assms(6) assms(17)
        rwf_query_def set_filterQuery by fastforce
     moreover have A ⊆ J
       by (metis False One_nat_def Set.is_empty_def Suc_leI Suc_le_lessD ‹1 ≤ card A› ‹A ⊆ V›
           assms(1) assms(2) card_gt_0_iff finite_Int inf.absorb_iff2 inf_commute sup_commute
sup_inf_absorb sup_inf_distrib1)
     then have restrict A z = restrict A xx using ‹restrict J z = xx› nested_include_restrict by blast
     define zI where zI = restrict I z
     define zJ where zJ = restrict J z
     have zJ = xx by (simp add: ‹restrict J z = xx› zJ_def)
     have zI = t by (simp add: ‹restrict I z = t› zI_def)
     have z = merge zJ zI by (simp add: ‹zI = t› ‹zJ = xx› assms(14))
     obtain AA XX where (AA, XX) = projectTable J (semiJoin (A, X) (I, t)) by simp
     have AA = A ∩ J
       using ‹(AA, XX) = projectTable J (semiJoin (A, X) (I, t))› by auto
     have (AA, XX) ∈ NQ_pos
       using ‹(AA, XX) = projectTable J (semiJoin (A, X) (I, t))› calculation image_iff assms(9)
        by fastforce
     then have restrict AA zJ ∈ XX
       using ‹(AA, XX) ∈ NQ_pos› ‹∀(A, X)∈NQ_pos. restrict A xx ∈ X› ‹zJ = xx› by blast
     then have restrict A z = restrict A zJ by (simp add: ‹restrict A z = restrict A xx› ‹zJ = xx›)
    moreover have restrict AA zJ = restrict A zJ by (simp add: ‹A ⊆ J› ‹AA = A ∩ J› inf.absorb1)
     then have restrict A z ∈ XX using ‹restrict AA zJ ∈ XX› calculation(2) by auto
     moreover have XX ⊆ Set.image (restrict J) X
     proof −
       obtain AAA XXX where (AAA, XXX) = semiJoin (A, X) (I, t) by simp
       then have XXX ⊆ X by auto
       then have XX = Set.image (restrict J) XXX
         using ‹(AA, XX) = projectTable J (semiJoin (A, X) (I, t))› ‹(AAA, XXX) = semiJoin (A,
X) (I, t)› by auto
       then show ?thesis by (simp add: ‹XXX ⊆ X› image_mono)
     qed
     then have restrict A z ∈ Set.image (restrict J) X using calculation(3) by blast
```

```
      obtain zz where restrict A z = restrict J zz zz ∈ X
        using ‹restrict A z ∈ restrict J ‘ X› by blast
      then have restrict A z = restrict A zz
        by (metis Int_absorb2 ‹A ⊆ J› restrict_nested subset_refl)
      moreover have restrict A zz = zz
      proof −
        have (A, X) ∈ Q by (simp add: ‹(A, X) ∈ Q›)
          then have table n A X using assms(17) rwf_query_def wf_atable_def wf_query_def by
fastforce
        then have wf_tuple n A zz using ‹zz ∈ X› table_def by blast
        then show ?thesis using restrict_idle by blast
      qed
      then have restrict A zz = zz using ‹restrict A z = restrict J zz› calculation(4) by auto
      then show ?thesis by (simp add: ‹zz ∈ X› calculation(4))
    qed
  qed
  then show ?thesis using calculation by blast
qed

lemma simple_set_inter:
  assumes I ⊆ (⋃X∈A. f X)
  shows I ⊆ (⋃X∈A. (f X) ∩ I)
proof −
  have ⋀x. x ∈ I ⟹ x ∈ (⋃X∈A. (f X) ∩ I)
  proof −
    fix x assume x ∈ I
    obtain X where X ∈ A x ∈ f X using ‹x ∈ I› assms by auto
    then show x ∈ (⋃X∈A. (f X) ∩ I) using ‹x ∈ I› by blast
  qed
  then show ?thesis by (simp add: subsetI)
qed

lemma union_restrict:
  assumes restrict I z1 = restrict I z2
  assumes restrict J z1 = restrict J z2
  shows restrict (I ∪ J) z1 = restrict (I ∪ J) z2
proof −
  define zz1 where zz1 = restrict (I ∪ J) z1
  define zz2 where zz2 = restrict (I ∪ J) z2
  have length z1 = length z2 by (metis assms(2) length_restrict)
  have ⋀i. i < length z1 ⟹ zz1!i = zz2!i
  proof −
    fix i assume i < length z1
    then show zz1!i = zz2!i
    proof (cases i ∈ I)
      case True
      then show ?thesis
        by (metis simple_restrict_none ‹i < length z1› ‹length z1 = length z2› assms(1)
            nth_restrict zz1_def zz2_def)
    next
      case False
      then show ?thesis
        by (metis simple_restrict_none UnE ‹i < length z1› ‹length z1 = length z2› assms(2)
            nth_restrict zz1_def zz2_def)
    qed
  qed
  then have ∀ i < length z1. (restrict (I ∪ J) z1)!i = (restrict (I ∪ J) z2)!i
```

using *zz1_def zz2_def* **by** *blast*
then show *?thesis*
by (*simp add*: *simple_list_index_equality* ‹*length z1 = length z2*›)
qed

lemma *partial_correctness_direct*:
assumes $V = I \cup J$
assumes *Set.is_empty* $(I \cap J)$
assumes *card* $I \geq 1$
assumes *card* $J \geq 1$
assumes *Q_I_pos* = *projectQuery I* (*filterQuery I Q*)
assumes *Q_J_pos* = *filterQuery J Q*
assumes *Q_I_neg* = *filterQueryNeg I Qn*
assumes *Q_J_neg* = *filterQuery J* (*Qn* − *Q_I_neg*)
assumes *R_I* = *genericJoin I Q_I_pos Q_I_neg*
assumes $X = \{(t,$ *genericJoin J* (*newQuery J Q_J_pos* $(I, t))$ (*newQuery J Q_J_neg* $(I, t))) \mid t\ .\ t \in$ *R_I*$\}$
assumes $R = (\bigcup (t, x) \in X.\ \{$*merge xx t* $\mid$ *xx . xx* $\in x\})$
assumes *R_NQ* = *genericJoin J NQ_pos NQ_neg*
assumes $\forall x.\ (x \in$ *R_I* $\longleftrightarrow$ *wf_tuple n I x* $\wedge\ (\forall (A,\ X) \in$ *Q_I_pos. restrict A x* $\in X) \wedge\ (\forall (A, X) \in$ *Q_I_neg. restrict A x* $\notin X))$
assumes $\forall t \in$ *R_I*. $(\forall y.\ (y \in$ *genericJoin J* (*newQuery J Q_J_pos* $(I, t))$ (*newQuery J Q_J_neg* $(I, t)) \longleftrightarrow$ *wf_tuple n J y* $\wedge$
$(\forall (A,\ X) \in ($*newQuery J Q_J_pos* $(I,\ t)).$ *restrict A y* $\in X) \wedge\ (\forall (A,\ X) \in ($*newQuery J Q_J_neg* $(I, t)).$ *restrict A y* $\notin X)))$
assumes *wf_tuple n V z* $\wedge\ (\forall (A,\ X) \in$ *Q. restrict A z* $\in X) \wedge\ (\forall (A,\ X) \in$ *Qn. restrict A z* $\notin X)$
assumes *rwf_query n V Q Qn*
shows $z \in R$
proof −
define *CI* **where** *CI* = *filterQuery I Q*
define *zI* **where** *zI* = *restrict I z*
have *wf_tuple n I zI*
using *assms*(*1*) *assms*(*15*) *wf_tuple_restrict_simple zI_def* **by** *auto*
have $\bigwedge A\ X.\ ((A,\ X) \in$ *Q_I_pos* $\Longrightarrow$ *restrict A zI* $\in X)$
proof −
fix *A X* **assume** $(A,\ X) \in$ *Q_I_pos*
have $(A,\ X) \in$ *projectQuery I Q* **using** ‹$(A,\ X) \in$ *Q_I_pos*› *assms*(*5*) **by** *auto*
then obtain *AA XX* **where** *X* = *Set.image* (*restrict I*) *XX* $(AA,\ XX) \in Q\ A = AA \cap I$ **by** *auto*
moreover have (*restrict AA z*) $\in XX$ **using** *assms*(*15*) *calculation*(*2*) **by** *blast*
then have *restrict I* (*restrict AA z*) $\in X$ **by** (*simp add*: *calculation*(*1*))
then show *restrict A zI* $\in X$
by (*metis calculation*(*3*) *inf.right_idem inf_commute restrict_nested zI_def*)
qed
moreover have $\bigwedge A\ X.\ ((A,\ X) \in$ *Q_I_neg* $\Longrightarrow$ *restrict A zI* $\notin X)$
proof −
fix *A X* **assume** $(A,\ X) \in$ *Q_I_neg*
then have $(A,\ X) \in$ *Qn* **by** (*simp add*: *assms*(*7*))
then have *restrict A z* $\notin X$ **using** *assms*(*15*) **by** *blast*
moreover have $A \subseteq I$ **using** ‹$(A,\ X) \in$ *Q_I_neg*› *assms*(*7*) **by** *auto*
then have *restrict A z* = *restrict A zI*
using *nested_include_restrict zI_def* **by** *metis*
then show *restrict A zI* $\notin X$ **using** *calculation* **by** *auto*
qed
then have *zI* $\in$ *R_I* **using** ‹*wf_tuple n I zI*› *assms*(*13*) *calculation* **by** *auto*
define *zJ* **where** *zJ* = *restrict J z*
have *wf_tuple n J zJ* **using** *assms*(*1*) *assms*(*15*) *wf_tuple_restrict_simple zJ_def* **by** *auto*
have $\bigwedge A\ X.\ ((A,\ X) \in$ *Q_J_pos* $\Longrightarrow$ *restrict A z* $\in X)$ **using** *assms*(*15*) *assms*(*6*) **by** *auto*
define *NQ* **where** *NQ* = *newQuery J Q_J_pos* $(I,$ *zI*$)$

have $\bigwedge A\ X.\ ((A,\ X) \in Q\_J\_pos \Longrightarrow (isSameIntersection\ zI\ (A \cap I)\ (restrict\ A\ z)))$
proof −
fix $A\ X$ **assume** $(A,\ X) \in Q\_J\_pos$
obtain $wf\_set\ n\ V\ wf\_tuple\ n\ I\ zI$ **using** ‹$wf\_tuple\ n\ I\ zI$› $assms(16)\ rwf\_query\_def\ wf\_query\_def$
by *blast*
moreover have $A \subseteq V$
proof −
have *included V Q_J_pos*
by (*metis filterQuery.elims assms(16) assms(6) included_def member_filter rwf_query_def*)
then show *?thesis* **using** ‹$(A,\ X) \in Q\_J\_pos$› *included_def* **by** *fastforce*
qed
moreover have $wf\_tuple\ n\ A\ (restrict\ A\ z)$ **by** $(meson\ assms(15)\ calculation(3)\ wf\_tuple\_restrict\_simple)$
then show $isSameIntersection\ zI\ (A \cap I)\ (restrict\ A\ z)$
using $isSame\_equi\_dev[of\ n\ V\ I\ zI\ A\ restrict\ A\ z\ A \cap I]$
by (*metis nested_include_restrict assms(1) calculation(1) calculation(2) calculation(3) inf_le1 inf_le2 sup_ge1 zI_def*)
qed
then have $\bigwedge A\ X.\ ((A,\ X) \in NQ \Longrightarrow restrict\ A\ zJ \in X)$
proof −
fix $A\ X$ **assume** $(A,\ X) \in NQ$
obtain $AA\ XX$ **where** $(A,\ X) = projectTable\ J\ (semiJoin\ (AA,\ XX)\ (I,\ zI))\ (AA,\ XX) \in Q\_J\_pos$
using NQ_def ‹$(A,\ X) \in NQ$› **by** *auto*
then have $restrict\ AA\ z \in XX$ **using** ‹$\bigwedge X\ A.\ (A,\ X) \in Q\_J\_pos \Longrightarrow restrict\ A\ z \in X$› **by** *blast*
then have $restrict\ AA\ z \in snd\ (semiJoin\ (AA,\ XX)\ (I,\ zI))$
using ‹$(AA,\ XX) \in Q\_J\_pos$› ‹$\bigwedge X\ A.\ (A,\ X) \in Q\_J\_pos \Longrightarrow isSameIntersection\ zI\ (A \cap I)\ (restrict\ A\ z)$› **by** *auto*
then have $restrict\ J\ (restrict\ AA\ z) \in X$
using ‹$(A,\ X) = projectTable\ J\ (semiJoin\ (AA,\ XX)\ (I,\ zI))$› **by** *auto*
then show $restrict\ A\ zJ \in X$
by (*metis* ‹$(A,\ X) = projectTable\ J\ (semiJoin\ (AA,\ XX)\ (I,\ zI))$› *fst_conv inf.idem inf_commute projectTable.simps restrict_nested semiJoin.simps zJ_def*)
qed
moreover have $\forall y.\ (y \in genericJoin\ J\ (newQuery\ J\ Q\_J\_pos\ (I,\ zI))\ (newQuery\ J\ Q\_J\_neg\ (I,\ zI)) \longleftrightarrow wf\_tuple\ n\ J\ y \wedge$
$(\forall (A,\ X) \in newQuery\ J\ Q\_J\_pos\ (I,\ zI).\ restrict\ A\ y \in X) \wedge (\forall (A,\ X) \in newQuery\ J\ Q\_J\_neg\ (I,\ zI).\ restrict\ A\ y \notin X))$
using ‹$zI \in R\_I$› $assms(14)$ **by** *auto*
then have $zJ \in genericJoin\ J\ (newQuery\ J\ Q\_J\_pos\ (I,\ zI))\ (newQuery\ J\ Q\_J\_neg\ (I,\ zI))$
$\longleftrightarrow wf\_tuple\ n\ J\ zJ \wedge (\forall (A,\ X) \in newQuery\ J\ Q\_J\_pos\ (I,\ zI).\ restrict\ A\ zJ \in X) \wedge$
$(\forall (A,\ X) \in newQuery\ J\ Q\_J\_neg\ (I,\ zI).\ restrict\ A\ zJ \notin X)$ **by** *blast*
moreover have $\forall (A,\ X) \in newQuery\ J\ Q\_J\_pos\ (I,\ zI).\ restrict\ A\ zJ \in X$
using $NQ\_def\ calculation(2)$ **by** *blast*
moreover have $\bigwedge A\ X.\ (A,\ X) \in newQuery\ J\ Q\_J\_neg\ (I,\ zI) \Longrightarrow restrict\ A\ zJ \notin X$
proof −
fix $A\ X$ **assume** $(A,\ X) \in newQuery\ J\ Q\_J\_neg\ (I,\ zI)$
then have $(A,\ X) \in (Set.image\ (\lambda tab.\ projectTable\ J\ (semiJoin\ tab\ (I,\ zI)))\ Q\_J\_neg)$
using *newQuery.simps* **by** *blast*
then obtain $AA\ XX$ **where** $(A,\ X) = projectTable\ J\ (semiJoin\ (AA,\ XX)\ (I,\ zI))$ **and** $(AA,\ XX) \in Q\_J\_neg$
by *auto*
then have $A = AA \cap J$ **by** *auto*
then have $(AA,\ XX) \in Qn$ **using** ‹$(AA,\ XX) \in Q\_J\_neg$› $assms(8)$ **by** *auto*
then have $restrict\ AA\ z \notin XX$ **using** $assms(15)$ **by** *blast*
show $restrict\ A\ zJ \notin X$
proof (*rule ccontr*)
assume $\neg\ (restrict\ A\ zJ \notin X)$
then have $restrict\ A\ zJ \in X$ **by** *simp*
then have $restrict\ A\ zJ \in Set.image\ (restrict\ J)\ (Set.filter\ (isSameIntersection\ zI\ (I \cap AA))\ XX)$

```
      by (metis projectTable.simps semiJoin.simps ‹(A, X) = projectTable J (semiJoin (AA, XX) (I,
zI))›
          inf_commute snd_conv)
    then obtain zz where restrict A zJ = restrict J zz and zz ∈ (Set.filter (isSameIntersection zI (I
∩ AA)) XX)
      by blast
    moreover have restrict A zJ = restrict AA zJ
      by (simp add: restrict_nested ‹A = AA ∩ J› zJ_def)
    then have restrict AA z = zz
    proof −
      have restrict J (restrict AA zz) = restrict J (restrict AA z)
        by (metis (no_types, lifting) restrict_nested ‹restrict A zJ = restrict AA zJ›
            calculation(1) inf_commute inf_left_idem zJ_def)
      moreover have isSameIntersection zI (I ∩ AA) zz
        using ‹zz ∈ Set.filter (isSameIntersection zI (I ∩ AA)) XX› by auto
      moreover have wf_tuple n AA zz
      proof −
        have rwf_query n V Q Qn by (simp add: assms(16))
        moreover have (AA, XX) ∈ Q ∪ Qn by (simp add: ‹(AA, XX) ∈ Qn›)
        then have wf_atable n (AA, XX) using calculation rwf_query_def wf_query_def by blast
        then show ?thesis
            using ‹zz ∈ Set.filter (isSameIntersection zI (I ∩ AA)) XX› table_def wf_atable_def by
fastforce
      qed
      moreover have restrict AA zz = zz using calculation(3) restrict_idle by blast
      moreover have AA ⊆ V
      proof −
        have included V Qn using assms(16) rwf_query_def by blast
        then show ?thesis using ‹(AA, XX) ∈ Qn› included_def by fastforce
      qed
      moreover have wf_set n V using assms(16) rwf_query_def wf_query_def by blast
      moreover have restrict (I ∩ AA) zz = restrict (I ∩ AA) zI
        using isSame_equi_dev[of n V AA zz V z I ∩ AA]
        by (metis (mono_tags, lifting) isSame_equi_dev ‹wf_tuple n I zI› assms(1)
            calculation(2) calculation(3) calculation(5) calculation(6) inf_le1 inf_le2 sup_ge1)
      then have restrict I (restrict AA zz) = restrict I (restrict AA z)
        by (metis (mono_tags, lifting) restrict_nested inf_le1 nested_include_restrict zI_def)
      then have restrict (I ∪ J) (restrict AA z) = restrict (I ∪ J) (restrict AA zz)
        using union_restrict calculation(1) by fastforce
      moreover have AA ⊆ I ∪ J
        by (metis ‹(AA, XX) ∈ Qn› assms(1) assms(16) case_prodD included_def rwf_query_def)
      then show ?thesis
        by (metis restrict_nested calculation(4) calculation(7) inf.absorb_iff2)
    qed
    then show False using ‹restrict AA z ∉ XX› calculation(2) by auto
  qed
 qed
 then have zJ ∈ genericJoin J (newQuery J Q_J_pos (I, zI)) (newQuery J Q_J_neg (I, zI))
   using ‹wf_tuple n J zJ› calculation(3) calculation(4) by blast
 have z = merge zJ zI
   using restrict_partition_merge assms(1) assms(15) assms(2) zI_def zJ_def by fastforce
 moreover have (zI, genericJoin J (newQuery J Q_J_pos (I, zI)) (newQuery J Q_J_neg (I, zI))) ∈
X
   using ‹zI ∈ R_I› assms(10) by blast
 then show ?thesis
  using ‹zJ ∈ genericJoin J (newQuery J Q_J_pos (I, zI)) (newQuery J Q_J_neg (I, zI))› assms(11)
    calculation(5) by blast
qed
```

lemma *obvious_forall*:
assumes $\forall\, x{\in}X.\ P\ x$
assumes $x{\in}X$
shows $P\ x$
by (*simp add*: *assms*(*1*) *assms*(*2*))

lemma *correctness*:
$[\![$*rwf_query n V Q Qn*; *card V* $\geq$ *1*$]\!] \Longrightarrow$ (*z* $\in$ *genericJoin V Q Qn* $\longleftrightarrow$ *wf_tuple n V z* $\wedge$
$(\forall\,(A,\ X){\in}Q.$ *restrict A z* $\in X) \wedge (\forall\,(A,\ X){\in}Qn.$ *restrict A z* $\notin X))$
proof (*induction V Q Qn arbitrary*: *z rule*: *genericJoin.induct*)
case (*1 V Q Qn*)
then show *?case*
proof (*cases card V* $\leq$ *1*)
case *True*
have *card V* = *1* **using** *1.prems*(*2*) *True le_antisym* **by** *blast*
then show *?thesis* **using** *base_correctness*[*of V n Q Qn genericJoin V Q Qn z*] **using** *1.prems*(*1*)
by *blast*
next
case *False*
obtain *I J* **where** (*I*, *J*) = *getIJ Q Qn V* **by** (*metis surj_pair*)
define *Q_I_pos* **where** *Q_I_pos* = *projectQuery I* (*filterQuery I Q*)
define *Q_I_neg* **where** *Q_I_neg* = *filterQueryNeg I Qn*
define *R_I* **where** *R_I* = *genericJoin I Q_I_pos Q_I_neg*
define *Q_J_neg* **where** *Q_J_neg* = *filterQuery J* (*Qn* $-$ *Q_I_neg*)
define *Q_J_pos* **where** *Q_J_pos* = *filterQuery J Q*
define *X* **where** *X* = {(*t*, *genericJoin J* (*newQuery J Q_J_pos* (*I*, *t*)) (*newQuery J Q_J_neg* (*I*,
t))) | *t* . *t* $\in$ *R_I*}
define *R* **where** *R* = ($\bigcup$(*t*, *x*) $\in$ *X*. {*merge xx t* | *xx* . *xx* $\in$ *x*})
then have *R* = *genericJoin V Q Qn*
using *vars_genericJoin*[*of V I J Q Qn Q_I_pos Q_I_neg R_I Q_J_neg Q_J_pos X R*]
by (*metis 1.prems*(*1*) *False Q_I_neg_def Q_I_pos_def Q_J_neg_def Q_J_pos_def R_I_def
Suc_1 X_def*
‹(*I*, *J*) = *getIJ Q Qn V*› *not_less_eq_eq*)
obtain *rwf_query n I Q_I_pos Q_I_neg* **and** *card I* $\geq$ *1*
by (*metis 1.prems*(*1*) *False Q_I_neg_def Q_I_pos_def Suc_1* ‹(*I*, *J*) = *getIJ Q Qn V*›
getIJ.getIJProperties(*1*)
getIJ.wf_firstRecursiveCall getIJ_axioms not_less_eq_eq)
then have $\forall\, x.$ (*x* $\in$ *R_I* $\longleftrightarrow$
wf_tuple n I x $\wedge$ $(\forall\,(A,\ X){\in}Q\_I\_pos.$ *restrict A x* $\in X)$ $\wedge$ $(\forall\,(A,\ X){\in}Q\_I\_neg.$ *restrict A x* $\notin X))$
using *1.IH*(*1*) *False Q_I_neg_def Q_I_pos_def R_I_def* ‹(*I*, *J*) = *getIJ Q Qn V*› **by** *auto*
moreover have $\forall\, t{\in}R\_I.\ (\forall\, y.$ (*y* $\in$ *genericJoin J* (*newQuery J Q_J_pos* (*I*, *t*)) (*newQuery J
Q_J_neg* (*I*, *t*)) $\longleftrightarrow$ *wf_tuple n J y* $\wedge$
$(\forall\,(A,\ X){\in}(newQuery\ J\ Q\_J\_pos\ (I,\ t)).$ *restrict A y* $\in X) \wedge (\forall\,(A,\ X){\in}(newQuery\ J\ Q\_J\_neg\ (I,$
t)). *restrict A y* $\notin X)))$
proof
fix *t* **assume** *t* $\in$ *R_I*
have *card J* $\geq$ *1*
by (*metis False Suc_1* ‹(*I*, *J*) = *getIJ Q Qn V*› *getIJProperties*(*2*) *le_SucE nat_le_linear*)
moreover have *rwf_query n J* (*newQuery J Q_J_pos* (*I*, *t*)) (*newQuery J Q_J_neg* (*I*, *t*))
by (*metis 1.prems*(*1*) *Diff_subset False Q_J_neg_def Q_J_pos_def Suc_1* ‹(*I*, *J*) = *getIJ Q
Qn V*›
getIJ.wf_secondRecursiveCalls getIJ_axioms not_less_eq_eq)
define *NQ_pos* **where** *NQ_pos* = *newQuery J Q_J_pos* (*I*, *t*)
define *NQ_neg* **where** *NQ_neg* = *newQuery J Q_J_neg* (*I*, *t*)
have $\bigwedge y.$ *y* $\in$ *genericJoin J NQ_pos NQ_neg* $\longleftrightarrow$
wf_tuple n J y $\wedge$ $(\forall\,(A,\ X){\in}NQ\_pos.$ *restrict A y* $\in X) \wedge (\forall\,(A,\ X){\in}NQ\_neg.$ *restrict A y* $\notin X)$
proof $-$

fix y
have *rwf_query n J NQ_pos NQ_neg*
using *NQ_neg_def NQ_pos_def* ‹*rwf_query n J* (*newQuery J Q_J_pos* (*I*, *t*)) (*newQuery J Q_J_neg* (*I*, *t*))› **by** *blast*
then show $y \in$ *genericJoin J NQ_pos NQ_neg* $\longleftrightarrow$
wf_tuple n J y $\wedge$ ($\forall$(*A*, *X*)$\in$*NQ_pos. restrict A y* $\in$ *X*) $\wedge$ ($\forall$(*A*, *X*)$\in$*NQ_neg. restrict A y* $\notin$ *X*)
using *1.IH*(*2*)[*of* (*I*, *J*) *I J Q_I_pos Q_I_neg R_I Q_J_neg Q_J_pos t y*]
by (*metis 1.prems*(*1*) *False NQ_neg_def NQ_pos_def Q_I_neg_def Q_I_pos_def Q_J_neg_def Q_J_pos_def R_I_def Suc_1* ‹(*I*, *J*) = *getIJ Q Qn V*› *calculation filter_Q_J_neg_same not_less_eq_eq*)
qed
then show $\forall y.$ ($y \in$ *genericJoin J* (*newQuery J Q_J_pos* (*I*, *t*)) (*newQuery J Q_J_neg* (*I*, *t*)) $\longleftrightarrow$ *wf_tuple n J y* $\wedge$
($\forall$(*A*, *X*)$\in$(*newQuery J Q_J_pos* (*I*, *t*)). *restrict A y* $\in$ *X*) $\wedge$ ($\forall$(*A*, *X*)$\in$(*newQuery J Q_J_neg* (*I*, *t*)). *restrict A y* $\notin$ *X*))
using *NQ_neg_def NQ_pos_def* **by** *blast*
qed
moreover obtain *V* = *I* $\cup$ *J Set.is_empty* (*I* $\cap$ *J*) *card I* $\geq$ *1 card J* $\geq$ *1*
by (*metis False Set.is_empty_def Suc_1* ‹(*I*, *J*) = *getIJ Q Qn V*› *coreProperties not_less_eq_eq*)
moreover have *rwf_query n V Q Qn* **by** (*simp add*: *1.prems*(*1*))
then show *?thesis*
proof −
have *z* $\in$ *genericJoin V Q Qn* $\Longrightarrow$ *wf_tuple n V z* $\wedge$ ($\forall$(*A*, *X*)$\in$*Q. restrict A z* $\in$ *X*) $\wedge$ ($\forall$(*A*, *X*)$\in$*Qn. restrict A z* $\notin$ *X*)
proof −
fix *z* **assume** *z* $\in$ *genericJoin V Q Qn*
have *z* $\in$ ($\bigcup$(*t*, *x*) $\in$ *X*. {*merge xx t* | *xx* . *xx* $\in$ *x*})
using *R_def* ‹*R* = *genericJoin V Q Qn*› ‹*z* $\in$ *genericJoin V Q Qn*› **by** *blast*
obtain *t R_NQ* **where** *z* $\in$ {*merge xx t* | *xx* . *xx* $\in$ *R_NQ*} (*t*, *R_NQ*) $\in$ *X*
using ‹*z* $\in$ ($\bigcup$(*t*, *x*)$\in$*X*. {*merge xx t* |*xx*. *xx* $\in$ *x*})› **by** *blast*
then have *t* $\in$ *R_I* **using** *X_def* **by** *blast*
define *NQ* **where** *NQ* = *newQuery J Q_J_pos* (*I*, *t*)
define *NQ_neg* **where** *NQ_neg* = *newQuery J Q_J_neg* (*I*, *t*)
have *R_NQ* = *genericJoin J NQ NQ_neg* **using** *NQ_def NQ_neg_def X_def* ‹(*t*, *R_NQ*) $\in$ *X*› **by** *blast*
obtain *xx* **where** *z* = *merge xx t xx* $\in$ *R_NQ*
using ‹*z* $\in$ {*merge xx t* |*xx*. *xx* $\in$ *R_NQ*}› **by** *blast*
have $\forall y.$ ($y \in$ *R_NQ* $\longleftrightarrow$ *wf_tuple n J y* $\wedge$ ($\forall$(*A*, *X*)$\in$*NQ. restrict A y* $\in$ *X*) $\wedge$ ($\forall$(*A*, *X*)$\in$*NQ_neg. restrict A y* $\notin$ *X*))
proof −
have $\forall$ *tt*$\in$*R_I*. ($\forall x.$ (*x* $\in$ *genericJoin J NQ NQ_neg*
$\longleftrightarrow$ *wf_tuple n J x* $\wedge$ ($\forall$(*A*, *X*)$\in$*NQ. restrict A x* $\in$ *X*) $\wedge$ ($\forall$(*A*, *X*)$\in$*NQ_neg. restrict A x* $\notin$ *X*)))
using *NQ_def NQ_neg_def* ‹*t* $\in$ *R_I*› *calculation*(*2*) **by** *auto*
moreover have *t*$\in$*R_I* **by** (*simp add*: ‹*t* $\in$ *R_I*›)
then have ($\forall x.$ (*x* $\in$ *genericJoin J NQ NQ_neg*
$\longleftrightarrow$ *wf_tuple n J x* $\wedge$ ($\forall$(*A*, *X*)$\in$*NQ. restrict A x* $\in$ *X*) $\wedge$ ($\forall$(*A*, *X*)$\in$*NQ_neg. restrict A x* $\notin$ *X*)))
using *obvious_forall*[**where** *?x*=*t* **and** *?X*=*R_I*] *calculation* **by** *fastforce*
then show *?thesis* **using** ‹*R_NQ* = *genericJoin J NQ NQ_neg*› **by** *blast*
qed
show *wf_tuple n V z* $\wedge$ ($\forall$(*A*, *X*)$\in$*Q. restrict A z* $\in$ *X*) $\wedge$ ($\forall$(*A*, *X*)$\in$*Qn. restrict A z* $\notin$ *X*)
using *partial_correctness*[*of V I J Q_I_pos Q Q_J_pos Q_I_neg Qn Q_J_neg NQ t NQ_neg R_NQ R_I n z xx*]
using *1.prems*(*1*) *NQ_def NQ_neg_def Q_I_neg_def Q_I_pos_def Q_J_neg_def Q_J_pos_def*
‹*R_NQ* = *genericJoin J NQ NQ_neg*› ‹$\forall y.$ (*y* $\in$ *R_NQ*) = (*wf_tuple n J y* $\wedge$ ($\forall$(*A*, *X*)$\in$*NQ. restrict A y* $\in$ *X*) $\wedge$ ($\forall$(*A*, *X*)$\in$*NQ_neg. restrict A y* $\notin$ *X*))›
‹*t* $\in$ *R_I*› ‹*xx* $\in$ *R_NQ*› ‹*z* = *merge xx t*› *calculation*(*1*) *calculation*(*3*) *calculation*(*4*)

calculation(5) *calculation*(6) **by** *blast*
qed
moreover have *wf_tuple n V z* ∧ (∀(*A*, *X*)∈*Q*. *restrict A z* ∈ *X*) ∧ (∀(*A*, *X*)∈*Qn*. *restrict A z* ∉ *X*) ⟹ *z* ∈ *genericJoin V Q Qn*
proof −
fix *z* **assume** *wf_tuple n V z* ∧ (∀(*A*, *X*)∈*Q*. *restrict A z* ∈ *X*) ∧ (∀(*A*, *X*)∈*Qn*. *restrict A z* ∉ *X*)
have *z* ∈ *R*
using *partial_correctness_direct*[*of V I J Q_I_pos Q Q_J_pos Q_I_neg Qn Q_J_neg R_I X R*
_ _ _ *n z*]
1.prems(1) *Q_I_neg_def Q_I_pos_def Q_J_neg_def Q_J_pos_def R_I_def R_def X_def*
‹1 ≤ *card I*› ‹1 ≤ *card J*› ‹*Set.is_empty* (*I* ∩ *J*)› ‹*V* = *I* ∪ *J*›
‹∀*t*∈*R_I*. ∀*y*. (*y* ∈ *genericJoin J* (*newQuery J Q_J_pos* (*I*, *t*)) (*newQuery J Q_J_neg* (*I*, *t*))) = (*wf_tuple n J y* ∧ (∀(*A*, *X*)∈*newQuery J Q_J_pos* (*I*, *t*). *restrict A y* ∈ *X*) ∧ (∀(*A*, *X*)∈*newQuery J Q_J_neg* (*I*, *t*). *restrict A y* ∉ *X*))›
‹∀*x*. (*x* ∈ *R_I*) = (*wf_tuple n I x* ∧ (∀(*A*, *X*)∈*Q_I_pos*. *restrict A x* ∈ *X*) ∧ (∀(*A*, *X*)∈*Q_I_neg*. *restrict A x* ∉ *X*))›
‹*wf_tuple n V z* ∧ (∀(*A*, *X*)∈*Q*. *restrict A z* ∈ *X*) ∧ (∀(*A*, *X*)∈*Qn*. *restrict A z* ∉ *X*)› **by** *blast*
then show *z* ∈ *genericJoin V Q Qn* **using** ‹*R* = *genericJoin V Q Qn*› **by** *blast*
qed
then show *?thesis* **using** *calculation* **by** *linarith*
qed
qed
qed

lemma *wf_set_finite*:
assumes *wf_set n A*
shows *finite A*
using *assms finite_nat_set_iff_bounded wf_set_def* **by** *auto*

lemma *vars_wrapperGenericJoin*:
fixes *Q* :: *'a query* **and** *Q_pos* :: *'a query* **and** *Q_neg* :: *'a query*
and *V* :: *nat set* **and** *Qn* :: *'a query*
assumes *Q* = *Set.filter* (λ(*A*, _). ¬ *Set.is_empty A*) *Q_pos*
and *V* = (⋃(*A*, *X*)∈*Q*. *A*)
and *Qn* = *Set.filter* (λ(*A*, _). *A* ⊆ *V* ∧ *card A* ≥ 1) *Q_neg*
and ¬ *Set.is_empty Q*
and ¬((∃(*A*, *X*)∈*Q_pos*. *Set.is_empty X*) ∨ (∃(*A*, *X*)∈*Q_neg*. *Set.is_empty A* ∧ ¬ *Set.is_empty X*))
shows *wrapperGenericJoin Q_pos Q_neg* = *genericJoin V Q Qn*
using *assms wrapperGenericJoin.simps*
proof −
let *?r* = *wrapperGenericJoin Q_pos Q_neg*
have *?r* = (*if* ((∃(*A*, *X*)∈*Q_pos*. *Set.is_empty X*) ∨ (∃(*A*, *X*)∈*Q_neg*. *Set.is_empty A* ∧ ¬ *Set.is_empty X*)) *then*
{}
else
let Q = *Set.filter* (λ(*A*, _). ¬ *Set.is_empty A*) *Q_pos in*
if Set.is_empty Q then
(⋂(*A*, *X*)∈*Q_pos*. *X*) − (⋃(*A*, *X*)∈*Q_neg*. *X*)
else
let V = (⋃(*A*, *X*)∈*Q*. *A*) *in*
let Qn = *Set.filter* (λ(*A*, _). *A* ⊆ *V* ∧ *card A* ≥ 1) *Q_neg in*
genericJoin V Q Qn) **by** *simp*
also have ... = (*let Q* = *Set.filter* (λ(*A*, _). ¬ *Set.is_empty A*) *Q_pos in*
if Set.is_empty Q then

```
      (⋂(A, X)∈Q_pos. X) − (⋃(A, X)∈Q_neg. X)
    else
      let V = (⋃(A, X)∈Q. A) in
      let Qn = Set.filter (λ(A, _). A ⊆ V ∧ card A ≥ 1) Q_neg in
      genericJoin V Q Qn) using assms(5) by simp
 moreover have ¬ (let Q = Set.filter (λ(A, _). ¬ Set.is_empty A) Q_pos in Set.is_empty Q)
  using assms(1) assms(4) by auto
 ultimately have (let Q = Set.filter (λ(A, _). ¬ Set.is_empty A) Q_pos in
    if Set.is_empty Q then
      (⋂(A, X)∈Q_pos. X) − (⋃(A, X)∈Q_neg. X)
    else
      let V = (⋃(A, X)∈Q. A) in
      let Qn = Set.filter (λ(A, _). A ⊆ V ∧ card A ≥ 1) Q_neg in
      genericJoin V Q Qn) = (let Q = Set.filter (λ(A, _). ¬ Set.is_empty A) Q_pos in
      let V = (⋃(A, X)∈Q. A) in
      let Qn = Set.filter (λ(A, _). A ⊆ V ∧ card A ≥ 1) Q_neg in
      genericJoin V Q Qn) by presburger
 also have ... = (genericJoin V Q Qn) using assms(1) assms(2) assms(3) by metis
 then show ?thesis using wrapperGenericJoin.simps assms(5) calculation by simp
qed

lemma wrapper_correctness:
 assumes card Q_pos ≥1
 assumes ∀(A, X)∈(Q_pos ∪ Q_neg). table n A X ∧ wf_set n A
  shows z ∈ wrapperGenericJoin Q_pos Q_neg ⟷ wf_tuple n (⋃(A, X)∈Q_pos. A) z ∧ (∀(A,
X)∈Q_pos. restrict A z ∈ X) ∧ (∀(A, X)∈Q_neg. restrict A z ∉ X)
proof (cases (∃(A, X)∈Q_pos. Set.is_empty X) ∨ (∃(A, X)∈Q_neg. Set.is_empty A ∧ ¬ Set.is_empty
X))
 let ?r = wrapperGenericJoin Q_pos Q_neg
 case True
 then have ?r = {} using wrapperGenericJoin.simps by simp
 have ¬ (wf_tuple n (⋃(A, X)∈Q_pos. A) z ∧ (∀(A, X)∈Q_pos. restrict A z ∈ X) ∧ (∀(A, X)∈Q_neg.
restrict A z ∉ X))
 proof (rule notI)
  assume wf_tuple n (⋃(A, X)∈Q_pos. A) z ∧ (∀(A, X)∈Q_pos. restrict A z ∈ X) ∧ (∀(A, X)∈Q_neg.
restrict A z ∉ X)
   then show False
   proof (cases ∃(A, X)∈Q_pos. Set.is_empty X)
     case True
     then show ?thesis
        using ‹wf_tuple n (⋃(A, X)∈Q_pos. A) z ∧ (∀(A, X)∈Q_pos. restrict A z ∈ X) ∧ (∀(A,
X)∈Q_neg. restrict A z ∉ X)›
       by (metis (no_types, lifting) Set.is_empty_def case_prod_beta′ empty_iff)
   next
     let ?v = replicate n None
     case False
     then have ∃(A, X)∈Q_neg. Set.is_empty A ∧ ¬ Set.is_empty X using True by blast
     then obtain A X where (A, X) ∈ Q_neg Set.is_empty A ¬ Set.is_empty X by auto
     then have table n A X using assms(2) by auto
     then have X ⊆ {?v} using ‹Set.is_empty A› table_empty unit_table_def
      by (metis Set.is_empty_def ‹¬ Set.is_empty X› empty_table_def set_eq_subset)
     then show ?thesis using ‹(A, X) ∈ Q_neg› ‹Set.is_empty A› ‹¬ Set.is_empty X›
       ‹wf_tuple n (⋃(A, X)∈Q_pos. A) z ∧ (∀(A, X)∈Q_pos. restrict A z ∈ X) ∧ (∀(A, X)∈Q_neg.
restrict A z ∉ X)›
       by (metis (no_types, lifting) Set.is_empty_def ‹table n A X› case_prod_beta′ empty_table_def
        in_unit_table inf_le1 inf_le2 prod.sel(1) snd_conv subset_empty table_empty wf_tuple_restrict_simple)
   qed
 qed
```

then show *?thesis* **using** ‹*?r* = {}› **by** *simp*
next
case *False*
then have *forall*: (∀(*A*, *X*)∈*Q_pos*. ¬ *Set.is_empty X*) ∧ (∀(*A*, *X*)∈*Q_neg*. ¬ *Set.is_empty A* ∨ *Set.is_empty X*) **by** *auto*
define *Q* **where** *Q* = *Set.filter* (λ(*A*, _). ¬ *Set.is_empty A*) *Q_pos*
define *V* **where** *V* = (⋃(*A*, *X*)∈*Q*. *A*)
let *?r* = *wrapperGenericJoin Q_pos Q_neg*
show *?thesis*
proof (*cases Q* = {})
case *True*
then have *r_def*: *?r* = (⋂(*A*, *X*)∈*Q_pos*. *X*) − (⋃(*A*, *X*)∈*Q_neg*. *X*) **using** *Q_def Set.is_empty_def False* **by** *auto*
moreover have *empty_u*: (⋃(*A*, *X*)∈*Q_pos*. *A*) = {}
by (*metis* (*no_types*, *lifting*) *Q_def SUP_bot_conv*(2) *Set.is_empty_def True case_prod_beta*′ *empty_iff member_filter*)
then have *V* = {} **using** *True V_def* **by** *blast*
moreover have ⋀*A X*. (*A*, *X*) ∈ *Q_pos* ⟹ *X* ⊆ {*replicate n None*}
proof −
fix *A X* **assume** (*A*, *X*) ∈ *Q_pos*
then have *table n A X* **using** *assms*(2) **by** *auto*
then have *A* = {}
proof −
have (*A*, *X*) ∉ *Q* **by** (*simp add*: *True*)
then show *?thesis* **by** (*simp add*: *Q_def Set.is_empty_def* ‹(*A*, *X*) ∈ *Q_pos*›)
qed
then show *X* ⊆ {*replicate n None*} **using** ‹*A* = {}› ‹*table n A X*› *table_empty unit_table_def* **by** *fastforce*
qed
have *?r* ⊆ {*replicate n None*}
proof (*rule subsetI*)
fix *x* **assume** *x* ∈ *?r*
obtain *A X* **where** (*A*, *X*) ∈ *Q_pos* **using** ‹*card Q_pos* ≥ *1*›
using *True card.empty not_one_le_zero* **by** (*metis bot.extremum_uniqueI subrelI*)
then have *x* ∈ *X* **using** ‹*x* ∈ *?r*› *r_def* **by** *auto*
then show *x* ∈ {*replicate n None*} **using** ‹(*A*, *X*) ∈ *Q_pos*› ‹⋀*X A*. (*A*, *X*) ∈ *Q_pos* ⟹ *X* ⊆ {*replicate n None*}› **by** *blast*
qed
let *?v* = *replicate n None*
show *?thesis*
proof (*cases ?r* = {})
case *True*
have *disj*: ∃ *A X*. ((*A*, *X*) ∈ *Q_pos* ∧ *X* = {}) ∨ ((*A*, *X*) ∈ *Q_neg* ∧ {*?v*} ⊆ *X*)
proof (*rule ccontr*)
assume ∄ *A X*. (*A*, *X*) ∈ *Q_pos* ∧ *X* = {} ∨ (*A*, *X*) ∈ *Q_neg* ∧ {*?v*} ⊆ *X*
then have *x_pos*: ∀(*A*, *X*)∈*Q_pos*. *X* = {*?v*} **using** ‹⋀*X A*. (*A*, *X*) ∈ *Q_pos* ⟹ *X* ⊆ {*replicate n None*}› **by** *blast*
moreover have *x_neg*: ∀(*A*, *X*)∈*Q_neg*. *?v* ∉ *X* **using** ‹∄ *A X*. (*A*, *X*) ∈ *Q_pos* ∧ *X* = {} ∨ (*A*, *X*) ∈ *Q_neg* ∧ {*replicate n None*} ⊆ *X*› **by** *blast*
ultimately have *?v* ∈ *?r* **using** *r_def*
proof −
have *?v* ∈ (⋂(*A*, *X*)∈*Q_pos*. *X*) **using** *x_pos* **by** *auto*
moreover have *?v* ∉ (⋃(*A*, *X*)∈*Q_neg*. *X*) **using** *x_neg* **by** *auto*
ultimately show *?thesis* **using** *r_def* **by** *auto*
qed
then show *False* **using** *True* **by** *blast*
qed
have ¬ (*wf_tuple n* (⋃(*A*, *X*)∈*Q_pos*. *A*) *z* ∧ (∀(*A*, *X*)∈*Q_pos*. *restrict A z* ∈ *X*) ∧ (∀(*A*,

```
X)∈Q_neg. restrict A z ∉ X))
    proof (rule notI)
      assume wf_tuple n (⋃(A, X)∈Q_pos. A) z ∧ (∀(A, X)∈Q_pos. restrict A z ∈ X) ∧ (∀(A,
X)∈Q_neg. restrict A z ∉ X)
      have z = ?v
        using ‹wf_tuple n (⋃(A, X)∈Q_pos. A) z ∧ (∀(A, X)∈Q_pos. restrict A z ∈ X) ∧ (∀(A,
X)∈Q_neg. restrict A z ∉ X)› empty_u wf_tuple_empty by auto
      then have ⋀A. restrict A z = z
       by (metis getIJ.restrict_index_out getIJ_axioms length_replicate length_restrict nth_replicate
nth_restrict simple_list_index_equality)
     then have (∃(A, X)∈Q_pos. z ∉ X) ∨ (∃(A, X)∈Q_neg. z ∈ X) using disj using ‹z = replicate
n None› by auto
      then show False
       using ‹⋀A. restrict A z = z› ‹wf_tuple n (⋃(A, X)∈Q_pos. A) z ∧ (∀(A, X)∈Q_pos. restrict
A z ∈ X) ∧ (∀(A, X)∈Q_neg. restrict A z ∉ X)› by auto
    qed
    then show ?thesis using True by blast
  next
    case False
    then have ?r = {?v} using ‹wrapperGenericJoin Q_pos Q_neg ⊆ {replicate n None}› by blast
    then have ⋀A X. (A, X) ∈ Q_pos ⟹ X = {?v}
      using Set.is_empty_def ‹⋀X A. (A, X) ∈ Q_pos ⟹ X ⊆ {replicate n None}› forall by fastforce
    then have ∀(A, X)∈Q_pos. X = {?v} by blast
    moreover have ∀(A, X)∈Q_neg. ?v ∉ X using ‹wrapperGenericJoin Q_pos Q_neg = {replicate
n None}› r_def by auto
    ultimately show ?thesis (is ?a ⟷ ?b)
    proof −
      have ?a ⟹ ?b
      proof −
        assume ?a
        then have z = ?v using ‹wrapperGenericJoin Q_pos Q_neg = {replicate n None}› by blast
        then have ⋀A. restrict A z = z
         by (metis getIJ.restrict_index_out getIJ_axioms length_replicate length_restrict nth_replicate
nth_restrict simple_list_index_equality)
       then show ?b using ‹∀(A, X)∈Q_neg. replicate n None ∉ X› ‹∀(A, X)∈Q_pos. X = {replicate
n None}›
          ‹z = replicate n None› empty_u wf_tuple_empty by fastforce
      qed
      moreover have ?b ⟹ ?a
        using ‹wrapperGenericJoin Q_pos Q_neg = {replicate n None}› empty_u wf_tuple_empty by
auto
      ultimately show ?thesis by blast
    qed
  qed
 next
  case False
  then have False_prev: Q ≠ {} by simp
  have covering V Q using V_def covering_def by blast
  moreover have included V Q using included_def V_def by fastforce
  define Qn where Qn = Set.filter (λ(A, _). A ⊆ V ∧ card A ≥ 1) Q_neg
  then have Qn ⊆ Q_neg by auto
  moreover have wf_query n V Q Qn
  proof −
    have wf_set n V
    proof −
      have ⋀x. x ∈ V ⟹ x < n
      proof −
        fix x assume x ∈ V
```

obtain A X **where** $x \in A$ $(A, X) \in Q$ **using** *V_def* ‹$x \in V$› **by** *blast*
then have $(A, X) \in (Q\_pos \cup Q\_neg)$ **by** (*simp add*: *Q_def*)
then have *wf_set n A* **using** *assms*(*2*) **by** *auto*
then show $x < n$ **using** ‹$x \in A$› *wf_set_def* **by** *blast*
qed
then show *?thesis* **using** *wf_set_def* **by** *blast*
qed
moreover have *card Q* $\geq$ *1*
proof −
have *finite Q_pos* **using** *assms*(*1*) *not_one_le_zero* **by** *fastforce*
then have *finite Q* **by** (*simp add*: *Q_def*)
then show *?thesis* **using** *False* **by** (*simp add*: *Suc_leI card_gt_0_iff*)
qed
moreover have $\bigwedge Y.\ Y \in (Q \cup Q\_neg) \Longrightarrow$ *wf_atable n Y*
proof −
fix *Y* **assume** $Y \in (Q \cup Q\_neg)$
then obtain *A X* **where** $Y = (A, X)$ **by** (*meson case_prodE case_prodI2*)
then have *table n A X*
by (*metis* (*no_types*, *lifting*) *Q_def UnE Un_iff* ‹$Y \in Q \cup Q\_neg$› *assms*(*2*) *case_prodD member_filter sup_commute*)
moreover have *finite A*
proof −
have *wf_set n A*
by (*metis* (*no_types*, *lifting*) *Q_def UnE Un_iff* ‹$Y = (A, X)$› ‹$Y \in Q \cup Q\_neg$› *assms*(*2*) *case_prod_conv member_filter sup.commute*)
then show *?thesis* **using** *wf_set_finite* **by** *blast*
qed
ultimately show *wf_atable n Y* **by** (*simp add*: ‹$Y = (A, X)$› *wf_atable_def*)
qed
ultimately have *wf_query n V Q Q_neg* **using** *wf_query_def* **by** *blast*
then show *?thesis* **using** *Un_iff* ‹$Qn \subseteq Q\_neg$› *subsetD wf_query_def*
proof −
obtain *pp* :: (*nat set* × *'a option list set*) *set* ⇒ (*nat set* × *'a option list set*) *set* ⇒ *nat* ⇒ *nat set* × *'a option list set* **where**
f1: ∀ *n N P Pa*. (*wf_query n N P Pa* ∨ ¬ *1* ≤ *card P* ∨ ¬ *wf_set n N* ∨ ¬ *wf_atable n* (*pp Pa P n*) ∧ *pp Pa P n* ∈ *P* ∪ *Pa*) ∧ (*1* ≤ *card P* ∧ *wf_set n N* ∧ (∀ *p*. *wf_atable n p* ∨ *p* ∉ *P* ∪ *Pa*) ∨ ¬ *wf_query n N P Pa*)
by (*metis* (*full_types*) *wf_query_def*)
have *pp Qn Q n* ∈ *Qn* ⟶ *pp Qn Q n* ∈ *Q_neg*
using ‹$Qn \subseteq Q\_neg$› **by** *blast*
then have *pp Qn Q n* ∈ *Q* ∪ *Q_neg* ∨ *wf_query n V Q Qn* **using** ‹*1* ≤ *card Q*› ‹*wf_set n V*› *f1* **by** *auto*
then show *?thesis* **using** ‹*1* ≤ *card Q*› ‹$\bigwedge Y.\ Y \in Q \cup Q\_neg \Longrightarrow$ *wf_atable n Y*› ‹*wf_set n V*› *f1* **by** *blast*
qed
qed
moreover have *non_empty_query Q*
proof −
have $\bigwedge A\ X.\ (A, X) \in Q \Longrightarrow$ *card A* $\geq$ *1*
proof −
fix *A X* **assume** *asm*: $(A, X) \in Q$
then have *wf_set n A*
by (*metis* ‹*included V Q*› *calculation*(*3*) *case_prodD included_def subsetD wf_query_def wf_set_def*)
then have *finite A* **using** *wf_set_finite* **by** *blast*
then show *card A* $\geq$ *1*
by (*metis* (*no_types*, *lifting*) *One_nat_def Q_def Set.is_empty_def Suc_leI asm card_gt_0_iff case_prod_beta' member_filter prod.sel*(*1*))

qed
then show *?thesis* **by** (*metis Q_def case_prodE fst_conv member_filter non_empty_query_def*)
qed
moreover have *included V Qn* **by** (*simp add*: *Qn_def case_prod_beta′ included_def*)
moreover have *non_empty_query Qn* **by** (*simp add*: *Qn_def case_prod_beta′ non_empty_query_def*)
then have *rwf_query n V Q Qn*
by (*simp add*: ‹*included V Q*› *calculation*(*1*) *calculation*(*3*) *calculation*(*4*) *calculation*(*5*) *rwf_query_def*)
moreover have *card V* ≥ *1*
proof −
obtain *A X* **where** (*A*, *X*) ∈ *Q_pos* ¬ *Set.is_empty A* **using** *False Q_def* **by** *force*
then have *A* ⊆ *V* **by** (*metis Q_def* ‹*included V Q*› *included_def member_filter prod.simps*(*2*))
moreover have *finite V* **using** *wf_set_finite* ‹*wf_query n V Q Qn*› *wf_query_def* **by** *blast*
ultimately show *?thesis*
by (*metis One_nat_def Set.is_empty_def Suc_leI* ‹¬ *Set.is_empty A*› *card_gt_0_iff subset_empty*)
qed
then have *z* ∈ *genericJoin V Q Qn* ⟷ *wf_tuple n V z* ∧ (∀(*A*, *X*)∈*Q*. *restrict A z* ∈ *X*) ∧ (∀(*A*, *X*)∈*Qn*. *restrict A z* ∉ *X*)
using *correctness*[**where** *?n*=*n* **and** *?V*=*V* **and** *?Q*=*Q* **and** *?z*=*z*] **by** (*simp add*: *calculation*(*6*))
moreover have *?r* = *genericJoin V Q Qn*
proof −
have *Qn* = *Set.filter* (λ(*A*, _). *A* ⊆ *V* ∧ *1* ≤ *card A*) *Q_neg* **using** *Qn_def* **by** *blast*
moreover have ¬ *Set.is_empty Q* **by** (*simp add*: *False_prev Set.is_empty_def*)
moreover have ¬ ((∃(*A*, *X*)∈*Q_pos*. *Set.is_empty X*) ∨ (∃(*A*, *X*)∈*Q_neg*. *Set.is_empty A* ∧ ¬ *Set.is_empty X*))
using *forall* **by** *blast*
ultimately show *?thesis* **using** *vars_wrapperGenericJoin*[*of Q Q_pos V Qn Q_neg*] *Q_def V_def* **by** *simp*
qed
moreover have *z* ∈ *genericJoin V Q Qn* ⟹ (∀(*A*, *X*)∈*Q_pos* − *Q*. *restrict A z* ∈ *X*) ∧ (∀(*A*, *X*)∈*Q_neg* − *Qn*. *restrict A z* ∉ *X*)
proof −
assume *z* ∈ *genericJoin V Q Qn*
have (⋀*A X*. (*A*, *X*)∈*Q_pos* − *Q* ⟹ *restrict A z* ∈ *X*)
proof −
fix *A X* **assume** (*A*, *X*) ∈ *Q_pos* − *Q*
then have *table n A X* **using** *assms*(*2*) **by** *auto*
moreover have *Set.is_empty A*
by (*metis* (*no_types*, *lifting*) *DiffD1 DiffD2 Q_def* ‹(*A*, *X*) ∈ *Q_pos* − *Q*› *case_prod_beta′ member_filter prod.sel*(*1*))
moreover have ¬ *Set.is_empty X* **using** *forall* **using** ‹(*A*, *X*) ∈ *Q_pos* − *Q*› **by** *blast*
ultimately have *X* = {*replicate n None*} **by** (*simp add*: *Set.is_empty_def empty_table_def table_empty unit_table_def*)
moreover have *wf_tuple n V z*
using ‹(*z* ∈ *genericJoin V Q Qn*) = (*wf_tuple n V z* ∧ (∀(*A*, *X*)∈*Q*. *restrict A z* ∈ *X*) ∧ (∀(*A*, *X*)∈*Qn*. *restrict A z* ∉ *X*))› ‹*z* ∈ *genericJoin V Q Qn*› **by** *linarith*
then have *restrict A z* = *replicate n None*
using ‹*Set.is_empty A*› *wf_tuple_empty wf_tuple_restrict_simple*
by (*metis Diff_Int2 Diff_Int_distrib2 Diff_eq_empty_iff Set.is_empty_def inf_le2*)
then show *restrict A z* ∈ *X* **by** (*simp add*: *calculation*)
qed
moreover have ⋀*A X*. (*A*, *X*)∈*Q_neg* − *Qn* ⟹ *restrict A z* ∉ *X*
proof −
fix *A X* **assume** (*A*, *X*) ∈ *Q_neg* − *Qn*
then have *notc*: ¬ (*card A* ≥ *1* ∧ *A* ⊆ *V*) **using** *Qn_def* **by** *auto*
then show *restrict A z* ∉ *X*
proof (*cases A* ⊆ *V*)
case *True*

then have *card A = 0* **using** *Qn_def* **using** *notc* **by** *linarith*
then have *Set.is_empty X*
by (*metis DiffD1 Set.is_empty_def True* ‹(*A*, *X*) ∈ *Q_neg* − *Qn*› ‹*1* ≤ *card V*› *card_eq_0_iff forall notc prod.simps*(*2*) *rev_finite_subset*)
then show *?thesis* **by** (*simp add*: *Set.is_empty_def*)
next
case *False*
then obtain *i* **where** *i* ∈ *A i* ∉ *V* **by** *blast*
then have *i* < *n*
proof −
have (*A*, *X*) ∈ *Q_neg* **using** ‹(*A*, *X*) ∈ *Q_neg* − *Qn*› **by** *auto*
then have *wf_set n A* **using** *assms*(*2*) **by** *auto*
then show *?thesis* **by** (*simp add*: ‹*i* ∈ *A*› *wf_set_def*)
qed
moreover have *table n A X*
proof −
have (*A*, *X*) ∈ *Q_neg* **using** ‹(*A*, *X*) ∈ *Q_neg* − *Qn*› **by** *auto*
then show *?thesis* **using** *assms*(*2*) **by** *auto*
qed
have *wf_tuple n V z*
using ‹(*z* ∈ *genericJoin V Q Qn*) = (*wf_tuple n V z* ∧ (∀(*A*, *X*)∈*Q*. *restrict A z* ∈ *X*) ∧ (∀(*A*, *X*)∈*Qn*. *restrict A z* ∉ *X*))› ‹*z* ∈ *genericJoin V Q Qn*› **by** *blast*
show *?thesis*
proof (*rule ccontr*)
let *?zz* = *restrict A z*
assume ¬ *?zz* ∉ *X*
then have *?zz* ∈ *X* **by** *blast*
then have *wf_tuple n A ?zz* **using** ‹*table n A X*› *table_def* **by** *blast*
then have *?zz* ! *i* ≠ *None*
by (*simp add*: ‹*i* ∈ *A*› *calculation wf_tuple_def*)
moreover have *z* ! *i* = *None* **using** ‹*wf_tuple n V z*› ‹*i* ∉ *V*› *wf_tuple_def* **using** ‹*i* < *n*› **by** *blast*
ultimately show *False*
using ‹*i* < *n*› ‹*i* ∈ *A*› ‹*wf_tuple n A* (*restrict A z*)› *nth_restrict wf_tuple_length* **by** *fastforce*
qed
qed
qed
ultimately show *?thesis* **by** *blast*
qed
ultimately have *z* ∈ *genericJoin V Q Qn* ⟷ (∀(*A*, *X*)∈*Q_pos* − *Q*. *restrict A z* ∈ *X*) ∧ (∀(*A*, *X*)∈*Q_neg* − *Qn*. *restrict A z* ∉ *X*)
∧ *wf_tuple n V z* ∧ (∀(*A*, *X*)∈*Q*. *restrict A z* ∈ *X*) ∧ (∀(*A*, *X*)∈*Qn*. *restrict A z* ∉ *X*) **by** *blast*
moreover have *V* = (⋃(*A*, *X*)∈*Q_pos*. *A*)
proof −
have (⋃(*A*, *X*)∈*Q_pos* − *Q*. *A*) = {}
proof −
have ⋀*A X*. (*A*, *X*) ∈ (*Q_pos* − *Q*) ⟹ *A* = {} **by** (*simp add*: *Q_def Set.is_empty_def*)
then show *?thesis* **by** *blast*
qed
moreover have *V* = (⋃(*A*, *X*)∈*Q*. *A*) **using** *V_def* **by** *simp*
moreover have (⋃(*A*, *X*)∈*Q_pos*. *A*) = (⋃(*A*, *X*)∈*Q*. *A*) ∪ (⋃(*A*, *X*)∈*Q_pos* − *Q*. *A*) **using** *Q_def* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed
ultimately show *?thesis* (**is** *?a* = *?b*)
proof −
have *?a* ⟹ *?b* **using** *Diff_iff* ‹(*z* ∈ *genericJoin V Q Qn*) = ((∀(*A*, *X*)∈*Q_pos* − *Q*. *restrict A z* ∈ *X*) ∧ (∀(*A*, *X*)∈*Q_neg* − *Qn*. *restrict A z* ∉ *X*) ∧ *wf_tuple n V z* ∧ (∀(*A*, *X*)∈*Q*. *restrict A z* ∈ *X*)

∧ (∀(A, X)∈Qn. restrict A z ∉ X))› ‹V = (⋃(A, X)∈Q_pos. A)› ‹wrapperGenericJoin Q_pos Q_neg = genericJoin V Q Qn› **by** *blast*
moreover have *?b* ⟹ *?a* **using** *Q_def Qn_def* ‹(z ∈ genericJoin V Q Qn) = (wf_tuple n V z ∧ (∀(A, X)∈Q. restrict A z ∈ X) ∧ (∀(A, X)∈Qn. restrict A z ∉ X))› ‹V = (⋃(A, X)∈Q_pos. A)› ‹wrapperGenericJoin Q_pos Q_neg = genericJoin V Q Qn› **by** *auto*
ultimately show *?thesis* **by** *blast*
qed
qed
qed

end

end

3 Example instantiations and queries

theory *Examples_Join*
imports *Generic_Join*
begin

3.1 Instantiations

global_interpretation *Max_getIJ*: *getIJ* λ_ _ *V*. (*V* − {*Max V*}, {*Max V*})
defines *Max_getIJ_genericJoin* = *Max_getIJ.genericJoin*
and *Max_getIJ_wrapperGenericJoin* = *Max_getIJ.wrapperGenericJoin*
by *standard* (*metis Diff_disjoint Max_in One_nat_def Pair_inject Suc_1 Suc_le_mono card.insert_remove card.empty card.infinite card_insert_disjoint finite.emptyI inf_commute insert_Diff insert_absorb insert_is_Un insert_not_empty le_numeral_extra*(*4*) *not_numeral_le_zero sup_commute*)

global_interpretation *Min_getIJ*: *getIJ* λ_ _ *V*. ({*Min V*}, *V* − {*Min V*})
defines *Min_getIJ_genericJoin* = *Min_getIJ.genericJoin*
and *Min_getIJ_wrapperGenericJoin* = *Min_getIJ.wrapperGenericJoin*
by *standard* (*metis Diff_disjoint Min_in One_nat_def Pair_inject Suc_1 card.insert_remove card.empty card.infinite card_insert_disjoint finite.emptyI insert_Diff insert_absorb insert_is_Un insert_not_empty le_numeral_extra*(*4*) *not_less_eq_eq not_numeral_le_zero*)

3.2 Queries

value *Max_getIJ.genericJoin* {*0*, *1*} {({*0* , *1*}, {[*Some* (*0* :: *nat*), *Some 0*], [*Some 1*, *Some 1*]}), ({*0* , *1*}, {[*Some 0*, *Some 0*], [*Some 0*, *Some 1*]})} {} :: *nat table*
value *Min_getIJ.genericJoin* {*0*, *1*} {({*0* , *1*}, {[*Some* (*0* :: *nat*), *Some 0*], [*Some 1*, *Some 1*]}), ({*0* , *1*}, {[*Some 0*, *Some 0*], [*Some 0*, *Some 1*]})} {} :: *nat table*

fun *protoTableTriangle* :: *nat* ⇒ *nat table* **where**
protoTableTriangle 0 = {[*Some 0*, *Some 0*]}
| *protoTableTriangle* (*Suc n*) = (*protoTableTriangle n*) ∪ {[*Some* (*Suc n*), *Some 0*], [*Some 0*, *Some* (*Suc n*)]}

fun *auxInsertNoneTriangle* :: *nat tuple* ⇒ *nat* ⇒ *nat tuple* **where**
auxInsertNoneTriangle l 0 = *None* # *l*
| *auxInsertNoneTriangle* (*x* # *q*) (*Suc n*) = *x* # (*auxInsertNoneTriangle q n*)
| *auxInsertNoneTriangle* [] (*Suc v*) = *undefined*

fun *insertNoneTriangle* :: *nat table* ⇒ *nat* ⇒ *nat table* **where**
insertNoneTriangle t n = {*auxInsertNoneTriangle x n* | *x* . *x* ∈ *t*}

value *set* [*0* ..< *5*]

fun *getTableTriangle* :: *nat* ⇒ *nat* ⇒ *nat atable* **where**
getTableTriangle n i = ({*0*, *1*, *2*} − {*i*}, *insertNoneTriangle* (*protoTableTriangle n*) *i*)

fun *getQueryTriangle* :: *nat* ⇒ *nat query* **where**
getQueryTriangle n = {*getTableTriangle n 0*, *getTableTriangle n 1*, *getTableTriangle n 2*}

definition *verticesTriangle* :: *vertices* **where** *verticesTriangle* = {*0*, *1*, *2*}

value *getQueryTriangle 2*

value *Max_getIJ.genericJoin verticesTriangle* (*getQueryTriangle 2*) {({*0*, *2*}, {[*Some 0*, *None*, *Some 0*]})}

value *let n = 2 in let* ((_, *A*), (_, *B*), (_, *C*)) = (*getTableTriangle n 0*, *getTableTriangle n 1*, *getTableTriangle n 2*) *in*
let AB = join A True B in join AB True C
value *Min_getIJ.wrapperGenericJoin* (*getQueryTriangle 2*) {}
value *Max_getIJ.wrapperGenericJoin* (*getQueryTriangle 2*) {}
value *New_max.wrapperGenericJoin* (*getQueryTriangle 2*) {}

end

References

[1] H. Q. Ngo, C. Ré, and A. Rudra. Skew strikes back: New developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, Feb. 2014.