Fakultät für Mathematik und Informatik
Lehrgebiet Datenbanksysteme für neue Anwendungen

Prof. Dr. Ralf Hartmut Güting

Diplomarbeit zum Thema

Optimierung geschachtelter Abfragen im Datenbanksystem SECONDO

zur Erlangung des akademischen Grades eines
Diplom-Informatikers

Burkart Poneleit
Oberlindelbach 28
91338 Igensdorf
Matrikel-Nr. 5672031

Igensdorf, den 14. Juni 2009

Inhaltsverzeichnis

1. Einleitung

1.1. Beschreibung SECONDO

- Erweiterbares Datenbanksystem
- Second-Order-Signatures
- Kernel/Algebren
- Optimierer
- GUI

1.2. SQL

- Datenbanksprache

  Datendefinition (DDL)

  Datenmanipulation (DML)

  Rechtevergabe (DCL)

- Abfrageblock bestehend aus

  SELECT-Klausel

  FROM-Klausel

  WHERE-Klausel

  GROUP BY... HAVING-Klausel

ORDER BY-Klausel

- geschachtelte Abfragen

  in der SELECT-Klausel

  in der FROM-Klausel

  in der WHERE Klausel

  in der GROUP BY... HAVING-Klausel

2. Review

2.1. Überblick

- SQL-Standard unterscheidet in

  „scalar subqueries“ (Ergebnis der Abfrage ist ein einzelner Wert)

  „row subqueries“ (Ergebnis der Abfrage ist eine Liste/Menge von Werten)

  „table subqueries“ (Ergebnis der Abfrage ist eine Relation)

- nach Korrelation

  nicht-korreliert (trivial, Ergebnis ist Konstante/Liste von Konstanten)

  korrelierte geschachtelte Abfrage

    ohne Aggregation

    mit Aggregation

- Ort des Auftretens

  SELECT-Klausel

  FROM-Klausel

  WHERE-Klausel

  (GROUP BY... HAVING-Klausel)

2.2. Optimierungsansätze für geschachtelte Abfragen

2.2.1. Mit Standard-Operatoren

2.2.2. Mit erweiterten Operatoren

- G-Aggregation
- G-Join
- G-Restriction
- G-Outerjoin
- Apply
- SegmentApply
- NULL-rejecting Outerjoin

[GLJ01, EGLGJ07, ISO92] Die Auswertung von geschachtelten Abfragen wird im SQL-Standard [ISO92] definiert. Hierbei wird der innere Abfrageblock für jedes Tupel des äußeren Abfrageblocks ausgewertet. Enthält die Unterabfrage korrelierte Join-Prädikate, so werden die entsprechenden Werte aus dem Tupel des äußeren Abfrageblocks als Konstanten in die Prädikate der Subquery übernommen. [Kim82, GW87] Klassifikation von Prädikaten in Simple(Vergleich mit Konstante(n)), Nested, Join und Divisionsprädikat, Einschränkung bei Join-Prädikaten auf =-Operator,
nur eine Spalte in der Select-Klausel erlaubt in Subquery,
keine GROUP BY... HAVING-Klausel in Subquery [Bül87],
Innerer und und äußerer Abfrageblock,
Klassifkation in N =(kein Join-Prädikat, dass auf Relationen im äußeren Abfrageblock verweist, keine Aggregationsfunktion in Select-Klausel, ergibt eine Liste von Konstanten), A =(kein Join-Prädikat, dass auf Relationen im äußeren Abfrageblock verweist, Aggregationsfunktion in Select-Klausel, kann vollkommen unabhängig vom äußeren Abfrageblock ausgewertet werden, Ergebnis ist immer eine Konstante, J =(hat Join-Prädikat das die/eine Relation aus dem äußeren Abfrageblock referenziert, keine Aggregationsfunktion in Select-Klausel), JA =(hat Join-Prädikat mit Verweis auf Relation im äußeren Abfrageblock, Aggregationsfunktion), D =(ein Divisions-Prädikat, dass in einer der beiden Abfrageblöcke ein Join-Prädikat mit Verweis auf eine Relation im äußeren Abfrageblock, drückt die relationale Divisionsoperation aus),

nested-iteration method, vollständige Auswertung des inneren Abfrageblocks für jedes Tupel des äußeren Blocks

2.2.3. Algorithmus NEST-N-J

Alle FROM-Klausel kombinieren
Die Konjunktion der WHERE-Klauseln bilden
Das geschachtelte Prädikat $[R_i.C_h \text{ op } (\texttt{SELECT} R_j.C_m \dots)]$ ersetzen durch ein neues Join-Prädikat $[R_i.C_h \textit{ new-op } R_j.C_m]$, das mit der restlichen WHERE-Klausel per Konjunktion verbunden ist
Die SELECT-Klausel des äußersten Abfrageblocks behalten

```
SELECT A_1, ..., A_n
FROM R_1, ..., R_m
WHERE P_1, ..., P_l,
  X θ (
    SELECT
      T_i.B
    FROM
      T_1, ..., T_s
    WHERE
      Q_1, ..., Q_r
  )
```

wird transformiert in

```
SELECT A_1, ..., A_n
FROM R_1, ..., R_m, T_1, ..., T_s
WHERE P_1, ..., P_l,
  Q_1, ..., Q_r
  X θ' T_i.B
```

$X \subset \{A_1, \dots, A_n\}$

$\theta \in \{\text{IN}, \text{NOT IN}, =, \neq, >, \geq, <, \leq\}$

$$\theta' = \begin{cases} = & \text{falls } \theta = \text{IN}, \\ \neq & \text{falls } \theta = \text{NOT IN}, \\ \theta & \text{sonst.} \end{cases}$$

2.2.4. Algorithmus NEST-JA2

1. temporäre Relation = Projektion der äußeren Relation(en) auf die Join-Spalten und Restriktion durch *simple* Prädikate, die in der äußeren WHERE-Klausel enthalten sind
2. temporäre Relation = Join aus 1. temporärer und innerer/en Relationen, falls Aggregationsfunktion = COUNT, dann müssen die simplen Prädikate auf die innere Relation vor dem Join angewandt werden. Ist die Aggregationsfunktion COUNT(*), dann muss COUNT über die/eine Join-Spalte ausgeführt werden. Das Join-Prädikat muss den selben Operator enthalten, wie die ursprüngliche geschachtelte Abfrage. Der Join-Operator in der ursprünglichen Abfrage muss in = geändert werden. In der SELECT-Klausel muss die Join-Spalte der äußeren Relation anstelle der Inneren Relation verwendet werden

```
SELECT A_1, ..., A_n
FROM R_1, ..., R_m
WHERE P_1(R_1), ..., P_k(R_1),
  P_1, ..., P_l,,
  R_i.X θ (
    SELECT AGGR(T_j.A)
    FROM T_1, ..., T_s
    WHERE Q_1, ..., Q_r,
      pred[R_1.Y op T_1.Z]
  )
```

Die Projektion der äußeren Relation

```
TEMP1 = (
  SELECT DISTINCT R_1.Y
  FROM R_1
  WHERE P_1(R_1), ..., P_k(R_1)
  )
```

```
TEMP2 = (
  SELECT T_j.A, T_1.Z
  FROM T_1, ..., T_s
  WHERE Q_1, ..., Q_r
  )
```

```
TEMP3 = (
  SELECT T_1.Y, AGGR(T_j.A) AS AggrResult
  FROM TEMP1, TEMP2
  WHERE pred[R_1.Y op T_1.Z]
  GROUP BY
    T_1.Y
  )
```

```
TEMP3 = (
  SELECT T_1.Y, AGGR(T_j.A) AS AggrResult
  FROM TEMP1 OUTER JOIN TEMP2
    ON pred[R_1.Y op T_1.Z]
```

```
  GROUP BY
    T1.Y
  )
```

```
SELECT A1, ..., An
FROM R1, ..., Rm,TEMP3
WHERE P1, ..., Pl,
  Ri.X θ TEMP3.AggrResult,
  R1.Y =Temp3.Z
```

$\theta \in \{=, \neq, >, \geq, <, \leq\}$

allgemeiner Algorithmus zur Entschachtelung mit temporären Tabellen, auf der Basis von Join- und Outerjoin-Operatoren und temporären Tabellen,

3. Entwurf

Begriffserklärung Schachtelunsgtiefe

Schritte des Optimierers

- Query-Rewriting
- Schema-Lookup
- Abfrageplan-Ermittlung
- Übersetzung Plan in ausführbare Syntax

3.1. Optimierung durch Auflösung der Korrelation

durch vorhanden Operatoren in SECONDO fast vollständig umsetzbar, nur Erweiterung um Full-Outerjoin-Operator notwendig
Vergleich nested-iteration/dekorrelierter Plan wird nicht implementiert:
Für den Vergleich der Kosten müsste die komplette Selektivitäts- und Kostenbestimmung für die notwendigen temporären Relationen simuliert werden (**ausführlicher beschreiben**)
kann nicht für alle Formen von Subqueries genutzt werden (Beispiele)
Ergebnis sollte eine Abfrage der Schachtelungstiefe 0 sein. Der Optimierer

3.2. Integration der Übersetzung von geschachtelten Abfragen in den Optimierer

Schema-Lookup erweitern

4. Implementierung

4.1. Umschreiben quantifizierter Prädikate

Prädikate mit den Operatoren **EXISTS**, **NOT EXISTS**, **ANY** und **ALL** werden in äquivalente Prädikate mit Aggregationen überführt.

4.2. Entschachtelung von Subqueries

4.3. Schema-Lookup für geschachtelte Abfragen

4.4. Plan-Ermittlung

4.5. Übersetzung des Plans in SECONDO-Syntax

5. Leistungsbewertung

5.1. Laufzeitvergleich geschachtelter TPC-D Abfragen

	Laufzeit in ms	
Abfrage Nr.	iterativ	entschachtelt
Q2		
Q4		
Q7		
Q8		
Q9		
Q15		
Q16		
Q17		
Q18		
Q20		
Q21		
Q22		

Tabelle 5.1.: Laufzeit „kalt“

	Laufzeit	
Abfrage Nr.	iterativ	entschachtelt
Q2		
Q4		
Q7		
Q8		
Q9		
Q15		
Q16		
Q17		
Q18		
Q20		
Q21		
Q22		

Tabelle 5.2.: Laufzeit „warm“

Literaturverzeichnis

[BMM07] Brantner, M., N. May und G. Moerkotte: *Unnesting Scalar SQL Queries in the Presence of Disjunction*. In: *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, Seiten 46–55, Istanbul,, April 2007.

[Bül87] Bültzingsloewen, Günter von: *Translating and Optimizing SQL Queries Having Aggregates*. In: *VLDB '87: Proceedings of the 13th International Conference on Very Large Data Bases*, Seiten 235–243, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[CB05] Cao, Bin und Antonio Badia: *A nested relational approach to processing SQL subqueries*. In: *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, Seiten 191–202, New York, NY, USA, 2005. ACM.

[Clo97] Clocksin, William F.: *Clause and effect: Prolog programming for the working programmer*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[CM03] Clocksin, W. F. und Chris Mellish: *Programming in Prolog, 5th Edition*. Springer, 2003.

[Dat95] Date, C. J.: *An Introduction to Database Systems, 6th Edition*. Addison-Wesley, 1995.

[Day87] Dayal, Umeshwar: *Of Nests and Trees: A Unified Approach to Processing Queries That Contain Nested Subqueries, Aggregates, and Quantifiers*. In: *VLDB '87: Proceedings of the 13th International Conference on Very Large Data Bases*, Seiten 197–208, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[EGLGJ07] Elhemali, Mostafa, César A. Galindo-Legaria, Torsten Grabs und Milind M. Joshi: *Execution strategies for SQL subqueries*. In: *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, Seiten 993–1004, New York, NY, USA, 2007. ACM.

[GL94] Galindo-Legaria, César A.: *Outerjoins as disjunctions.* In: *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, Seiten 348–358, New York, NY, USA, 1994. ACM.

[GLJ01] Galindo-Legaria, César und Milind Joshi: *Orthogonal optimization of subqueries and aggregation.* In: *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, Seiten 571–581, New York, NY, USA, 2001. ACM.

[GLR97] Galindo-Legaria, César und Arnon Rosenthal: *Outerjoin simplification and reordering for query optimization.* ACM Trans. Database Syst., 22(1):43–74, 1997.

[GW87] Ganski, Richard A. und Harry K. T. Wong: *Optimization of nested SQL queries revisited.* SIGMOD Rec., 16(3):23–33, 1987.

[ISO92] ISO: *ISO/IEC 9075:1992: Title: Information technology — Database languages — SQL.* International Organization for Standardization, Geneva, Switzerland, 1992. Available in English only.

[Kim82] Kim, Won: *On optimizing an SQL-like nested query.* ACM Trans. Database Syst., 7(3):443–469, 1982.

[MG03] Microsoft, Goetz Graefe und Goetz Graefe: *Executing Nested Queries.* In: *In BTW*, Seiten 58–77, 2003.

[Mur89] Muralikrishna, M.: *Optimization and dataflow algorithms for nested tree queries.* In: *VLDB '89: Proceedings of the 15th international conference on Very large data bases*, Seiten 77–85, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[Mur92] Muralikrishna, M.: *Improved Unnesting Algorithms for Join Aggregate SQL Queries.* In: *VLDB '92: Proceedings of the 18th International Conference on Very Large Data Bases*, Seiten 91–102, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[RGL90] Rosenthal, Arnon und Cesar Galindo-Legaria: *Query graphs, implementing trees, and freely-reorderable outerjoins.* In: *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, Seiten 291–299, New York, NY, USA, 1990. ACM.

[RR98] Rao, Jun und Kenneth A. Ross: *Reusing invariants: a new strategy for correlated queries.* SIGMOD Rec., 27(2):37–48, 1998.

[SKS01] Silberschatz, Abraham, Henry F. Korth und S. Sudarshan: *Database System Concepts, 4th Edition*. McGraw-Hill Book Company, 2001.

[Tra98] Transaction Processing Performance Council (TPC): *TPC BENCHMARK^{TM}D*. http://www.tpc.org/tpcd/spec/tpcd_current.pdf, 1998. [Online; accessed 31-January-2009].

[WY76] Wong, Eugene und Karel Youssefi: *Decomposition - A Strategy for Query Processing*. ACM Trans. Database Syst., 1(3):223–241, 1976.

A. TPC-D Abfragen

Q2

```
SELECT
  s_acctbal,
  s_name,
  n_name,
  p_partkey,
  p_mfgr,
  s_address,
  s_phone,
  s_comment
FROM
  part,
  supplier,
  partsupp,
  nation,
  region
WHERE
  p_partkey = ps_partkey
  AND s_suppkey = ps_suppkey
  AND p_size = [SIZE]
  AND p_type LIKE '%[TYPE]'
  AND s_nationkey = n_nationkey
  AND n_regionkey = r_regionkey
  AND r_name = '[REGION]'
  AND ps_supplycost = (
    SELECT MIN(ps_supplycost)
    FROM
      partsupp, supplier,
      nation, region
    WHERE
      p_partkey = ps_partkey
      AND s_suppkey = ps_suppkey
      AND s_nationkey = n_nationkey
      AND n_regionkey = r_regionkey
      AND r_name = '[REGION]')
  ORDER BY
    s_acctbal DESC,
    n_name,
    s_name,
    p_partkey;
```

Q4

```
SELECT
  o_orderpriority,
  COUNT(*) AS order_count
FROM
  orders
WHERE
  o_orderdate >= DATE '[DATE]'
  AND o_orderdate < DATE '[DATE]' + INTERVAL '3' MONTH
  AND EXISTS (
    SELECT
      *
    FROM
      lineitem
    WHERE
      l_orderkey = o_orderkey
      AND l_commitdate < l_receiptdate
  )
GROUP BY
  o_orderpriority
  ORDER BY
  o_orderpriority;
```

Q7

```
SELECT
  supp_nation,
  cust_nation,
  l_year, SUM(volume) AS revenue
FROM (
  SELECT
    n1.n_name AS supp_nation,
    n2.n_name AS cust_nation,
    EXTRACT(YEAR FROM l_shipdate) AS l_year,
    l_extendedprice * (1 - l_discount) AS volume
  FROM
    supplier,
    lineitem,
    orders,
    customer,
    nation n1,
    nation n2
  WHERE
    s_suppkey = l_suppkey
    AND o_orderkey = l_orderkey
    AND c_custkey = o_custkey
    AND s_nationkey = n1.n_nationkey
    AND c_nationkey = n2.n_nationkey
    AND (
```

```
    (n1.n_name = '[NATION1]' AND n2.n_name = '[NATION2]')
    OR
    (n1.n_name = '[NATION2]' AND n2.n_name = '[NATION1]')
  )
  AND l_shipdate BETWEEN DATE '1995-01-01' AND DATE '1996-12-31'
) AS shipping
GROUP BY
  supp_nation,
  cust_nation,
  l_year
ORDER BY
  supp_nation,
  cust_nation,
  l_year;
```

Q8

```
SELECT
  o_year,
  SUM(
    CASE
      WHEN nation = '[NATION]'
      THEN volume
      ELSE 0
    END
  ) / SUM(volume) AS mkt_share
FROM (
  SELECT
    EXTRACT(YEAR FROM o_orderdate) AS o_year,
    l_extendedprice * (1-l_discount) AS volume,
    n2.n_name AS nation
  FROM
    part,
    supplier,
    lineitem,
    orders,
    customer,
    nation n1,
    nation n2,
    region
  WHERE
    p_partkey = l_partkey
    AND s_suppkey = l_suppkey
    AND l_orderkey = o_orderkey
    AND o_custkey = c_custkey
    AND c_nationkey = n1.n_nationkey
    AND n1.n_regionkey = r_regionkey
    AND r_name = '[REGION]'
    AND s_nationkey = n2.n_nationkey
    AND o_orderdate BETWEEN DATE '1995-01-01' AND DATE '1996-12-31'
    AND p_type = '[TYPE]'
```

```
) AS all_nations
GROUP BY
  o_year
ORDER BY
  o_year;
```

Q9

```
SELECT
  nation,
  o_year,
  SUM(amount) AS sum_profit
FROM (
  SELECT
    n_name AS nation,
    EXTRACT(year from o_orderdate) AS o_year,
    l_extendedprice*(1-l_discount)-ps_supplycost*l_quantity AS amount
  FROM
    part,
    supplier,
    lineitem,
    partsupp,
    orders,
    nation
  WHERE
    s_suppkey = l_suppkey
    AND ps_suppkey = l_suppkey
    AND ps_partkey = l_partkey
    AND p_partkey = l_partkey
    AND o_orderkey = l_orderkey
    AND s_nationkey = n_nationkey
    AND p_name LIKE '%[COLOR]%'
) AS profit
GROUP BY
  nation,
  o_year
ORDER BY
  nation,
  o_year DESC;
```

Q15

```
SELECT
  s_suppkey,
  s_name,
  s_address,
  s_phone,
  total_revenue
FROM
```

```
  supplier,
  revenue[STREAM_ID]
WHERE
  s_suppkey = supplier_no
  AND total_revenue = (
    SELECT
    MAX(total_revenue)
    FROM
    revenue[STREAM_ID]
  )
ORDER BY
  s_suppkey;
```

Q16

```
SELECT
  p_brand,
  p_type,
  p_size,
  COUNT(DISTINCT ps_suppkey) AS supplier_cnt
FROM
  partsupp,
  part
WHERE
  p_partkey = ps_partkey
  AND p_brand <> '[BRAND]'
  AND p_type NOT LIKE '[TYPE]%'
  AND p_size IN ([SIZE1], [SIZE2], [SIZE3], [SIZE4], [SIZE5], [SIZE6],
  [SIZE7], [SIZE8])
  AND ps_suppkey NOT IN (
    SELECT
      s_suppkey
    FROM
      supplier
    WHERE
      s_comment LIKE '%Customer%Complaints%'
  )
GROUP BY
  p_brand,
  p_type,
  p_size
ORDER BY
  supplier_cnt DESC,
  p_brand,
  p_type,
  p_size;
```

Q17

```
SELECT
  SUM(l_extendedprice) / 7.0 AS avg_yearly
FROM
  lineitem,
  part
WHERE
  p_partkey = l_partkey
  AND p_brand = '[BRAND]'
  AND p_container = '[CONTAINER]'
  AND l_quantity < (
    SELECT
      0.2 * AVG(l_quantity)
    FROM
      lineitem
    WHERE
      l_partkey = p_partkey
  );
```

Q20

```
SELECT
  s_name,
  s_address
FROM
  supplier, nation
WHERE
  s_suppkey IN (
    SELECT
      ps_suppkey
    FROM
      partsupp
    WHERE
    ps_partkey IN (
      SELECT
      p_partkey
      FROM
      part
      WHERE
      p_name LIKE '[COLOR]%'
    )
    AND ps_availqty > (
      SELECT
        0.5 * SUM(l_quantity)
      FROM
        lineitem
      WHERE
        l_partkey = ps_partkey
        AND l_suppkey = ps_suppkey
        AND l_shipdate >= DATE('[DATE]')
        AND l_shipdate < DATE('[DATE]') + INTERVAL '1' YEAR
    )
```

```
  )
  AND s_nationkey = n_nationkey
  AND n_name = '[NATION]'
ORDER BY
  s_name;
```

Q21

```
SELECT
  s_name,
  COUNT(*) AS numwait
FROM
  supplier,
  lineitem l1,
  orders,
  nation
WHERE
  s_suppkey = l1.l_suppkey
  AND o_orderkey = l1.l_orderkey
  AND o_orderstatus = 'F'
  AND l1.l_receiptdate > l1.l_commitdate
  AND EXISTS (
    SELECT
      *
    FROM
      lineitem l2
    WHERE
      l2.l_orderkey = l1.l_orderkey
      AND l2.l_suppkey <> l1.l_suppkey
  )
  AND NOT EXISTS (
    SELECT
      *
    FROM
      lineitem l3
    WHERE
      l3.l_orderkey = l1.l_orderkey
      AND l3.l_suppkey <> l1.l_suppkey
      AND l3.l_receiptdate > l3.l_commitdate
  )
  AND s_nationkey = n_nationkey
  AND n_name = '[NATION]'
GROUP BY
  s_name
ORDER BY
  numwait DESC,
  s_name;
```

Q22

```
SELECT
  cntrycode,
  COUNT(*) AS numcust,
  SUM(c_acctbal) AS totacctbal
FROM (
  SELECT
    SUBSTRING(c_phone from 1 for 2) AS cntrycode,
    c_acctbal
  FROM
    customer
  WHERE
    SUBSTRING(c_phone from 1 for 2) IN
    ('[I1]','[I2]','[I3]','[I4]','[I5]','[I6]','[I7]')
    AND c_acctbal > (
      SELECT
        AVG(c_acctbal)
      FROM
        customer
      WHERE
        c_acctbal > 0.00
        AND SUBSTR (c_phone from 1 for 2) IN
        ('[I1]','[I2]','[I3]','[I4]','[I5]','[I6]','[I7]')
    )
    AND NOT EXISTS (
      SELECT
        *
      FROM
        orders
      WHERE
        o_custkey = c_custkey
    )
) AS custsale
GROUP BY
  cntrycode
ORDER BY
  cntrycode;
```

B. SECONDO SQL-Dialekt

```
sql-clause  ->  let objectname mquery.
                |let(objectname, mquery, secondo-rest-query).
                |sql mquery.
                |sql(mquery, secondo-rest-query).

aggr        ->  groupattr | groupattr as newname | aggr2

aggr2       ->  count(distinct-clause *) as newname
                | aggrop(ext-attr-expr) as newname
                |aggregate(ext-attr-expr, aggrfun, datatype, datatype-constant )
                 as newname

aggrop      ->  min | max | sum | avg | extract | count

aggr-clause ->  aggr | [aggr, aggr-list]

aggr-fun    ->  (∗)|(+)| union_new | intersection_new | ...
                % any name fun of a binary SECONDO-operator or function object
                with syntax fun : T × T -> T
                which should be associative and commutative. Infix-operators
                must be enclosed in round parentheses.
aggr-list   ->  aggr | aggr, aggr-list
attr        ->  attrname | var:attrname
```