

Handout: Introduction to Hacking PostgreSQL

Neil Conway and Gavin Sherry

January 14, 2007

Contents

1	Parser Changes	2
2	Node changes	3
3	Analysis Phase Changes	4
4	System Catalog Changes	6
5	CREATE TRIGGER Changes	7
6	Trigger descriptor changes	7
7	Relation initialisation	8
8	Executor Changes	9
9	Low level executor hackery	12
10	pg_dump and psql	12
11	Regression test system	13
12	Documentation	14

1 Parser Changes

src/backend/parser/gram.y, circa line 2529:

```
1 CreateTrigStmt :
2     CREATE TRIGGER name
3     TriggerActionTime TriggerEvents ON
4     qualified_name TriggerForSpec TriggerWhen EXECUTE PROCEDURE
5     func_name '(' TriggerFuncArgs ')'
6     {
7         CreateTrigStmt *n = makeNode(CreateTrigStmt);
8         n->trigname = $3;
9         n->relation = $8;
10        n->funcname = $13;
11        n->args = $15;
12        n->before = $5;
13        n->row = $9;
14        memcpy(n->actions, $6, 4);
15        n->isconstraint = FALSE;
16        n->deferrable = FALSE;
17        n->initdeferred = FALSE;
18        n->constrel = NULL;
19        n->when = $10;
20        n->rtable = NIL;
21        $$ = (Node *)n;
22    }
23 ;
24
25 /* ... */
26
27 TriggerWhen :
28     WHEN '(' a_expr ')'          { $$ = $3; }
29     | /*EMPTY*/                 { $$ = NULL; }
30 ;
```

2 Node changes

src/include/nodes/parsenodes.h circa line 1200:

```
1  /*-----
2  *      Create/Drop TRIGGER Statements
3  *-----
4  */
5
6  typedef struct CreateTrigStmt
7  {
8      NodeTag      type;
9      char         *trigname;      /* TRIGGER's name */
10     RangeVar     *relation;      /* relation trigger is on */
11     List         *funcname;      /* qual. name of function to call */
12     List         *args;          /* list of (T_String) Values or NIL */
13     bool         before;        /* BEFORE/AFTER */
14     bool         row;           /* ROW/STATEMENT */
15     char         actions[4];     /* 1 to 3 of 'i', 'u', 'd', + trailing \0 */
16
17     /* The following are used for referential */
18     /* integrity constraint triggers */
19     bool         isconstraint;   /* This is an RI trigger */
20     bool         deferrable;     /* [NOT] DEFERRABLE */
21     bool         initdeferred;  /* INITIALLY {DEFERRED|IMMEDIATE} */
22     RangeVar     *constrrel;     /* opposite relation */
23     Node         *when;         /* WHEN clause qual */
24     List         *rtable;       /* range table for interpreting WHEN expr */
25 } CreateTrigStmt;
```

src/backend/nodes/copyfuncs.c circa line 2490:

```
1  static CreateTrigStmt *
2  _copyCreateTrigStmt(CreateTrigStmt *from)
3  {
4      CreateTrigStmt *newnode = makeNode(CreateTrigStmt);
5
6      COPY_STRING_FIELD(trigname);
7      COPY_NODE_FIELD(relation);
8      COPY_NODE_FIELD(funcname);
9      COPY_NODE_FIELD(args);
10     COPY_SCALAR_FIELD(before);
11     COPY_SCALAR_FIELD(row);
12     strcpy(newnode->actions, from->actions); /* in-line string field */
13     COPY_SCALAR_FIELD(isconstraint);
14     COPY_SCALAR_FIELD(deferrable);
15     COPY_SCALAR_FIELD(initdeferred);
16     COPY_NODE_FIELD(constrrel);
17     COPY_NODE_FIELD(when);
18     COPY_NODE_FIELD(rtable);
19
20     return newnode;
21 }
```

src/backend/nodes/equalfuncs.c circa line 1462:

```
1 static bool
2 _equalCreateTrigStmt (CreateTrigStmt *a, CreateTrigStmt *b)
3 {
4     COMPARE_STRING_FIELD(trigname);
5     COMPARE_NODE_FIELD(relation);
6     COMPARE_NODE_FIELD(funcname);
7     COMPARE_NODE_FIELD(args);
8     COMPARE_SCALAR_FIELD(before);
9     COMPARE_SCALAR_FIELD(row);
10    if (strcmp(a->actions, b->actions) != 0)    /* in-line string field */
11        return false;
12    COMPARE_SCALAR_FIELD(isconstraint);
13    COMPARE_SCALAR_FIELD(deferrable);
14    COMPARE_SCALAR_FIELD(initdeferred);
15    COMPARE_NODE_FIELD(constrel);
16    COMPARE_NODE_FIELD(when);
17    COMPARE_NODE_FIELD(rtable);
18
19    return true;
20 }
```

3 Analysis Phase Changes

src/backend/parser/analyze.c, circa line 2087:

```
1 /*
2  * transformCreateTrigStmt -
3  *   transform a CREATE TRIGGER statement. Most of the work we do is
4  *   transforming the statement's WHEN clause, if any.
5  */
6 static Query *
7 transformCreateTrigStmt(ParseState *pstate, CreateTrigStmt *stmt)
8 {
9     Query      *qry;
10    Relation    rel;
11    RangeTblEntry *oldrte;
12    RangeTblEntry *newrte;
13    int        i;
14
15    qry = makeNode(Query);
16    qry->commandType = CMD_UTILITY;
17    qry->utilityStmt = (Node *) stmt;
18
19    /* If there's no WHEN clause, we're done. */
20    if (!stmt->when)
21        return qry;
22
23    /*
24     * Note that we acquire and keep an exclusive lock on the target table
```

```

25     * only if there's a WHEN clause; if there's no WHEN, we acquire the same
26     * lock in CreateTrigger(), so the effect should be the same.
27     */
28     rel = heap_openrv(stmt->relation, AccessExclusiveLock);
29
30     /*
31     * Setup RTEs for the NEW and OLD relations in the main pstate, for use
32     * in parsing the trigger qualification. We initially add "OLD" with RT
33     * index 1, and "NEW" with RT index 2, and then change them to use the
34     * correct varnos below.
35     */
36     Assert(pstate->p_rtable == NIL);
37     oldrte = addRangeTableEntryForRelation(pstate, rel,
38                                           makeAlias("*OLD*", NIL),
39                                           false, false);
40     newrte = addRangeTableEntryForRelation(pstate, rel,
41                                           makeAlias("*NEW*", NIL),
42                                           false, false);
43
44     for (i = 0; stmt->actions[i] != '\0'; i++)
45     {
46         if (stmt->actions[i] == 'd' || stmt->actions[i] == 'u')
47         {
48             addRTEtoQuery(pstate, oldrte, false, true, true);
49             break;
50         }
51     }
52
53     for (i = 0; stmt->actions[i] != '\0'; i++)
54     {
55         if (stmt->actions[i] == 'i' || stmt->actions[i] == 'u')
56         {
57             addRTEtoQuery(pstate, newrte, false, true, true);
58             break;
59         }
60     }
61
62     /* process WHEN clause as though it was a WHERE clause */
63     stmt->when = transformWhereClause(pstate, stmt->when, "WHEN");
64
65     if (list_length(pstate->p_rtable) != 2)
66         ereport(ERROR,
67                 (errmsg("trigger WHEN condition may not contain "
68                       "references to other relations")));
69
70
71     stmt->rtable = list_copy(pstate->p_rtable);
72
73     /* aggregates not allowed */
74     if (pstate->p_hasAggs)
75         ereport(ERROR,

```

```

76         (errcode(ERRCODE_GROUPING_ERROR),
77         errmsg("trigger WHEN condition may not contain aggregate functions"))\
       →);
78
79     /* subselects are not allowed either, at least for now */
80     if (pstate->p_hasSubLinks)
81         ereport(ERROR,
82                 (errcode(ERRCODE_INVALID_OBJECT_DEFINITION),
83                 errmsg("trigger WHEN condition may not contain subqueries"))\
84                 →);
85
86     /*
87     * Rewrite the WHEN expression to give the right varno to the NEW and OLD
88     * relations, so that these relations can be treated specially by the
89     * executor.
90     */
91     ChangeVarNodes(stmt->when, 1, TRIG_OLD_VARNO, 0);
92     ChangeVarNodes(stmt->when, 2, TRIG_NEW_VARNO, 0);
93
94     /* Close relation, but keep the exclusive lock */
95     heap_close(rel, NoLock);
96
97     return qry;
98 }

```

4 System Catalog Changes

src/include/catalog/pg_trigger.h, circa line 33:

```

1 CATALOG(pg_trigger,2620)
2 {
3     Oid          tgrelid;          /* triggered relation */
4     NameData     tgname;          /* trigger ' name */
5     Oid          tgfoid;          /* OID of function to be called */
6     int2         tgtype;          /* BEFORE/AFTER UPDATE/DELETE/INSERT
7                                     * ROW/STATEMENT */
8     bool         tgenabled;        /* trigger is enabled/disabled */
9     bool         tgisconstraint;  /* trigger is a RI constraint */
10    NameData     tgconstrname;     /* RI constraint name */
11    Oid          tgconstrrelid;    /* RI table of foreign key definition */
12    bool         tgdeferrable;     /* RI trigger is deferrable */
13    bool         tginitdeferred;   /* RI trigger is deferred initially */
14    int2         tgnargs;          /* # of extra arguments in tgargs */
15
16    /* VARIABLE LENGTH FIELDS: */
17    int2vector   tgattr;           /* reserved for column-specific triggers */
18    bytea        tgargs;           /* first\000second\000tgnargs\000 */
19    text         tgqual;           /* string form of qualification clause */
20 } FormData_pg_trigger;
21
22 /* ... */

```

```

23
24 /* _____
25 *      compiler constants for pg_trigger
26 * _____
27 */
28 #define Natts_pg_trigger          14
29 #define Anum_pg_trigger_tgrelid   1
30 #define Anum_pg_trigger_tgname    2
31 #define Anum_pg_trigger_tgfoid    3
32 #define Anum_pg_trigger_tgtype    4
33 #define Anum_pg_trigger_tgenabled 5
34 #define Anum_pg_trigger_tgisconstraint 6
35 #define Anum_pg_trigger_tgconstrname 7
36 #define Anum_pg_trigger_tgconstrelid 8
37 #define Anum_pg_trigger_tgdeferrable 9
38 #define Anum_pg_trigger_tginitdeferred 10
39 #define Anum_pg_trigger_tgnargs   11
40 #define Anum_pg_trigger_tgattr    12
41 #define Anum_pg_trigger_tgargs    13
42 #define Anum_pg_trigger_tgqual    14

```

5 CREATE TRIGGER Changes

src/backend/commands/trigger.c, circa line 331, in CreateTrigger():

```

1 values[Anum_pg_trigger_tgqual - 1] = DirectFunctionCall1(textin,
2                                CStringGetDatum(nodeToString(stmt->when)));

```

6 Trigger descriptor changes

src/include/utils/re1.h, circa line 68:

```

1 typedef struct TriggerDesc
2 {
3     /*
4      * Index data to identify which triggers are which. Since each trigger
5      * can appear in more than one class, for each class we provide a list of
6      * integer indexes into the triggers array.
7      */
8     #define TRIGGER_NUM_EVENT_CLASSES 3
9
10    uint16    n_before_statement [TRIGGER_NUM_EVENT_CLASSES];
11    uint16    n_before_row       [TRIGGER_NUM_EVENT_CLASSES];
12    uint16    n_after_row        [TRIGGER_NUM_EVENT_CLASSES];
13    uint16    n_after_statement  [TRIGGER_NUM_EVENT_CLASSES];
14    int       *tg_before_statement [TRIGGER_NUM_EVENT_CLASSES];
15    int       *tg_before_row       [TRIGGER_NUM_EVENT_CLASSES];
16    int       *tg_after_row        [TRIGGER_NUM_EVENT_CLASSES];
17    int       *tg_after_statement  [TRIGGER_NUM_EVENT_CLASSES];
18

```

```

19  /* The actual array of triggers is here */
20  Trigger    *triggers;
21  int        numtriggers;
22 } TriggerDesc;

```

src/include/utils/rel.h, circa line 45:

```

1  /*
2  * Likewise, this struct really belongs to trigger.h, but for convenience
3  * we put it here.
4  */
5  typedef struct Trigger
6  {
7      Oid        tgoid;          /* OID of trigger (pg-trigger row) */
8      /* Remaining fields are copied from pg-trigger, see pg-trigger.h */
9      char       *tgname;
10     Oid        tgfoid;
11     int16      tgtype;
12     bool       tgenabled;
13     bool       tgisconstraint;
14     Oid        tgconstrrelid;
15     bool       tgdeferrable;
16     bool       tginitdeferred;
17     int16      tgnargs;
18     int16      tgnattr;
19     int16      *tgattr;
20     char       **tgargs;
21     Node       *when;
22 } Trigger;

```

7 Relation initialisation

src/backend/commands/trigger.c circa line 956 in RelationBuildTriggers()

```

1      /* get the trigger's WHEN clause, if any */
2      tmp = heap_getattr(htup, Anum_pg_trigger_tgqual,
3                          RelationGetDescr(tgrel), &isnull);
4      Assert(!isnull);
5
6      when_str = DatumGetCString(DirectFunctionCall1(textout,
7                                                          PointerGetDatum(tmp)))\
8
9      /*
10     * XXX: we leak the node here because FreeTriggerDesc() has no
11     * ability to do a deep free of a Node.
12     *
13     * Ideally, the Node would be created in its own context which
14     * we could just reset. Since we create the node in the
15     * CacheMemoryContext the effect of this leak will be long lived
16     */

```

17
18

```
build->when = (Node *) stringToNode(when_str);
```

8 Executor Changes

src/backend/commands/trigger.c, circa line 3483:

```
1 /*
2  * There is some inefficiency here. Subsequent calls to setup_trigger_qual()
3  * during the same command will end up iterating through the array of
4  * triggers. Ideally, this redundant work should be avoided.
5  */
6 static void
7 setup_trigger_qual(ResultRelInfo *ri, EState *estate,
8                   bool before, int event)
9 {
10     TriggerDesc      *trigdesc = ri->ri_TrigDesc;
11     MemoryContext     old_cxt;
12     TrigQualState     *qual_state;
13     int               i;
14     int               ntrigs;
15     int               tgindx;
16
17     old_cxt = MemoryContextSwitchTo(estate->es_query_cxt);
18     if (ri->ri_TrigQuals == NULL)
19     {
20         ri->ri_TrigQuals = palloc(sizeof(TrigQualState));
21         ri->ri_TrigQuals->quals = palloc0(trigdesc->numtriggers * sizeof(List\
22     → *));
23     }
24     qual_state = ri->ri_TrigQuals;
25
26     if (before)
27     {
28         ntrigs = trigdesc->n_before_row[event];
29         tgindx = trigdesc->tg_before_row[event];
30     }
31     else
32     {
33         ntrigs = trigdesc->n_after_row[event];
34         tgindx = trigdesc->tg_after_row[event];
35     }
36
37     for (i = 0; i < ntrigs; i++)
38     {
39         Trigger *trigger;
40         int      trig_idx;
41         List     *qual;
42
43         trig_idx = tgindx[i];
44         trigger = &trigdesc->triggers[trig_idx];
```

```

44     if (!trigger->when)
45         continue;
46
47     if (qual_state->quals[trig_idx])
48         continue;
49
50     qual = make_ands_implicit((Expr *) trigger->when);
51     qual_state->quals[trig_idx] = (List *) ExecPrepareExpr((Expr *) qual,
52                                                         estate);
53 }
54
55 MemoryContextSwitchTo(old_cxt);
56 }
57

```

src/backend/commands/trigger.c, circa line 3542:

```

1  /*
2  * Test if a trigger qualification evaluates true for the
3  * input tuple(s)
4  */
5
6  static bool
7  test_trig_qual(EState *estate, Relation rel, HeapTuple oldtuple,
8                HeapTuple newtuple, List *qual, int event)
9  {
10     ExprContext *econtext = GetPerTupleExprContext(estate);
11     TupleDesc tupdesc = RelationGetDescr(rel);
12
13     if (event == TRIGGER_EVENT_INSERT ||
14         event == TRIGGER_EVENT_UPDATE)
15     {
16         if (econtext->ecxt_newtuple == NULL)
17             econtext->ecxt_newtuple = MakeSingleTupleTableSlot(tupdesc);
18         ExecClearTuple(econtext->ecxt_newtuple);
19         ExecStoreTuple(newtuple, econtext->ecxt_newtuple,
20                       InvalidBuffer, false);
21     }
22     if (event == TRIGGER_EVENT_UPDATE ||
23         event == TRIGGER_EVENT_DELETE)
24     {
25         if (econtext->ecxt_oldtuple == NULL)
26             econtext->ecxt_oldtuple = MakeSingleTupleTableSlot(tupdesc);
27         ExecClearTuple(econtext->ecxt_oldtuple);
28         ExecStoreTuple(oldtuple, econtext->ecxt_oldtuple,
29                       InvalidBuffer, false);
30     }
31
32     return ExecQual(qual, econtext, false);
33 }

```

```

1  setup_trigger_qual(s(relineno, estate, true, TRIGGER_EVENT_INSERT);
2  qual_state = relinfo->ri-TrigQuals;
3
4  LocTriggerData.type = T_TriggerData;
5  LocTriggerData.tg_event = TRIGGER_EVENT_INSERT |
6  TRIGGER_EVENT_ROW |
7  TRIGGER_EVENT_BEFORE;
8  LocTriggerData.tg_relation = relinfo->ri-RelationDesc;
9  LocTriggerData.tg_newtuple = NULL;
10 LocTriggerData.tg_newtuplebuf = InvalidBuffer;
11
12 for (i = 0; i < ntrigs; i++)
13 {
14     Trigger *trigger = &trigdesc->triggers[tgindx[i]];
15
16     if (!trigger->tgenabled)
17         continue;
18
19     /* Check the trigger's WHEN clause, if any */
20     if (trigger->when)
21     {
22         bool res;
23
24         res = test_trig_qual(estate, relinfo->ri-RelationDesc,
25                             NULL, trigtuple,
26                             qual_state->quals[tgindx[i]],
27                             TRIGGER_EVENT_INSERT);
28
29         if (!res)
30             continue;
31     }
32
33     LocTriggerData.tg_trigtuple = oldtuple = newtuple;
34     LocTriggerData.tg_trigtuplebuf = InvalidBuffer;
35     LocTriggerData.tg_trigger = trigger;
36     newtuple = ExecCallTriggerFunc(&LocTriggerData,
37                                   tgindx[i],
38                                   relinfo->ri-TrigFunctions,
39                                   relinfo->ri-TrigInstrument,
40                                   GetPerTupleMemoryContext(estate));
41     if (oldtuple != newtuple && oldtuple != trigtuple)
42         heap_freetuple(oldtuple);
43     if (newtuple == NULL)
44         break;
45 }

```

Note: also see identical modifications made to ExecBRDeleteTriggers() and ExecBRUpdateTriggers().

9 Low level executor hackery

src/backend/commands/trigger.c circa line 460 in CreateTrigger()

```
1  if (stmt->when)
2  {
3      ChangeVarNodes(stmt->when, TRIG_OLD_VARNO, 1, 0);
4      ChangeVarNodes(stmt->when, TRIG_NEW_VARNO, 2, 0);
5      recordDependencyOnExpr(&myself, stmt->when, stmt->rtable,
6                             DEPENDENCY_NORMAL);
7  }
```

src/backend/executor/execQual.c circa line 463 in ExecEvalVar()

```
1  case TRIG_OLD_VARNO: /* old tuple in trigger context */
2      slot = econtext->ecxt_oldtuple;
3      break;
4
5  case TRIG_NEW_VARNO: /* new tuple in trigger context */
6      slot = econtext->ecxt_newtuple;
7      break;
```

10 pg_dump and psql

src/backend/utils/adt/ruleutils.c circa line 519 in pg_get_triggerdef()

```
1  /* handle WHEN clause */
2  when_text = heap_getattr(ht_trig, Anum_pg_trigger_tgqual,
3                          RelationGetDescr(tgrel), &isnull);
4  Assert(!isnull);
5
6  when_str = DatumGetCString(DirectFunctionCall1(textout, when_text));
7  node = (Node *) stringToNode(when_str);
8
9  if (node)
10 {
11     Relation rel;
12     RangeTblEntry *oldrte;
13     RangeTblEntry *newrte;
14     deparse_context context;
15     deparse_namespace dpns;
16
17     rel = heap_open(trigrec->tgrelid, AccessShareLock);
18
19     appendStringInfo(&buf, "WHEN ");
20
21     oldrte = addRangeTableEntryForRelation(NULL, rel,
22                                           makeAlias("OLD", NIL),
23                                           false, false);
24
25     newrte = addRangeTableEntryForRelation(NULL, rel,
26                                           makeAlias("NEW", NIL),
```

```

27         false , false );
28
29     ChangeVarNodes(node , TRIG_OLD_VARNO, 1 , 0);
30     ChangeVarNodes(node , TRIG_NEW_VARNO, 2 , 0);
31
32     context.buf = &buf;
33     context.namespaces = list_make1(&dpns);
34     context.varprefix = true;
35     context.prettyFlags = 0;
36     context.indentLevel = PRETTYINDENTSTD;
37
38     dpns.rtable = list_make2(oldrte , newrte);
39     dpns.outer_varno = dpns.inner_varno = 0;
40     dpns.outer_rte = dpns.inner_rte = NULL;
41
42     get_rule_expr(node, &context , false);
43     appendStringInfo(&buf, " ");
44     heap_close(rel , AccessShareLock);
45 }

```

11 Regression test system

src/test/regress/sql/triggers.sql circa line 298:

```

1  — test the WHEN clause for trigger definitions
2  CREATE OR REPLACE FUNCTION notify_trig() RETURNS trigger LANGUAGE plpgsql AS \
  →$$
3  begin
4      raise notice '%s invoked: new.a = %, new.b = %', tg_name, new.a, new.b;
5      return new;
6  end$$;
7
8  CREATE TABLE when_test (a int , b int);
9
10 CREATE TRIGGER t1 BEFORE INSERT ON when_test FOR EACH ROW
11 WHEN (new.a > 5)
12 EXECUTE PROCEDURE notify_trig();
13
14 INSERT INTO when_test VALUES (NULL, 0);    — shouldn't fire
15 INSERT INTO when_test VALUES (10, 100);   — should fire
16
17 CREATE TRIGGER t2 BEFORE UPDATE ON when_test FOR EACH ROW
18 WHEN (new.b > 50)
19 EXECUTE PROCEDURE notify_trig();
20
21 UPDATE when_test SET b = b + 50;    — should fire once
22 UPDATE when_test SET b = b + 1;    — should fire twice
23
24 DROP TABLE when_test;
25 DROP FUNCTION notify_trig();

```

src/test/regress/expected/triggers.out circa line 389:

```
1  — test the WHEN clause for trigger definitions
2  CREATE OR REPLACE FUNCTION notify_trig() RETURNS trigger LANGUAGE plpgsql AS \
   →$$
3  begin
4      raise notice '%s invoked: new.a = %, new.b = %', tg_name, new.a, new.b;
5      return new;
6  end$$;
7  CREATE TABLE when_test (a int, b int);
8  CREATE TRIGGER t1 BEFORE INSERT ON when_test FOR EACH ROW
9  WHEN (new.a > 5)
10 EXECUTE PROCEDURE notify_trig();
11 INSERT INTO when_test VALUES (NULL, 0);    — shouldn't fire
12 INSERT INTO when_test VALUES (10, 100);    — should fire
13 NOTICE: t1s invoked: new.a = 10, new.b = 100
14 CREATE TRIGGER t2 BEFORE UPDATE ON when_test FOR EACH ROW
15 WHEN (new.b > 50)
16 EXECUTE PROCEDURE notify_trig();
17 UPDATE when_test SET b = b + 50;    — should fire once
18 NOTICE: t2s invoked: new.a = 10, new.b = 150
19 UPDATE when_test SET b = b + 1;    — should fire twice
20 NOTICE: t2s invoked: new.a = <NULL>, new.b = 51
21 NOTICE: t2s invoked: new.a = 10, new.b = 151
22 DROP TABLE when_test;
23 DROP FUNCTION notify_trig();
```

12 Documentation

doc/src/sgml/ref/create_trigger.sgml circa line 29:

```
1 <synopsis>
2 CREATE TRIGGER <replaceable class="PARAMETER">name</replaceable> { BEFORE | \
   →AFTER } { <replaceable class="PARAMETER">event</replaceable> [ OR ... ] }
3     ON <replaceable class="PARAMETER">table</replaceable> [ FOR [ EACH ] { \
   →ROW | STATEMENT } ]
4     [ WHEN ( <replaceable class="PARAMETER">expr</> ) ]
5     EXECUTE PROCEDURE <replaceable class="PARAMETER">funcname</replaceable> \
   → ( <replaceable class="PARAMETER">arguments</replaceable> )
6 </synopsis>
7
8 <!-- ... -->
9
10 <varlistentry>
11     <term><replaceable class="parameter">expr</replaceable></term>
12     <listitem>
13         <para>
14             An SQL expression which returns <type>boolean</type> result.
15         </para>
16
17         <para>
```

18 <literal>**INSERT**</literal> triggers may refer only to the
19 <literal>**NEW**</literal> table. <literal>**DELETE**</literal> triggers
20 may only refer to the <literal>**OLD**</literal> table.
21 <literal>**UPDATE**</literal> triggers may refer to both. The
22 expression may not refer to any other tables.
23 </para>
24
25 <para>
26 This feature is not supported on <literal>**FOR STATEMENT**</literal> triggers.
27 </para>
28 </listitem>
29 </varlistentry>