POSTGRES System Catalogs

(Version of: $Date: 90/07/30 13:50:14 $)

(Printed: July 30, 1990)

1. Introduction

This note describes the current proposal for the POSTGRES system catalogs. Table 1 lists the names of all the catalogs, whether each is local to a database (i.e., local) or shared with all databases (i.e., global), and what the catalog contains.

References to the system catalog relations should always be in lower case and be prepended with ‘‘pg_’’. For example, the RELATION relation should be referenced as ‘‘pg_relation’’.

Name	**Scope**	**Entry for each...**
AGGREGATE	local	aggregate function.
AM	local	access method.
AMOP	local	(access method, op class, op).
ATTRIBUTE	local	relation attribute.
DATABASE	global	database of current databases.
DEFAULTS	global	default values.
DEMON	global	system demon.
INDEX	local	secondary index.
INHERITPROC	local	inherited procedure.
INHERITS	local	direct relation/parent relationship.
IPL	local	inheritance precedence list entry.
LANGUAGE	local	language in which a procedure can be written.
MAGIC	global	magic constant in source code.
	???	shouldn’t this be local??? --hirohama
OPCLASS	local	adt?? operator class.
OPERATOR	local	adt?? operator.
PARG	local	argument to a procedure.
PROC	local	procedure.
RELATION	local	relation.
RULE	local	rule.
RULEPLANS	local	plans needed by the rule manager.
SERVER	global	active POSTGRES backend..
STATISTIC	local	relation attribute.
TYPE	local	data type.
USER	global	valid user.
VARIABLE	global	special variable values.
VERSION	local	version.

Table 1: Catalog summary.

The primitive data types used in the catalog definitions are listed in Table 2. In addition to the types listed, fixed length arrays of primitive types are supported by the system. For example, an array of integers with five elements is specified by ‘‘int[5].’’ Attributes of type POSTQUEL procedure can also be defined. An attribute that contains a different procedure in every tuple is declared to be of type ‘‘postquel’’. An attribute that contains a parameterized procedure in every tuple (i.e., the procedure is the same for all entries) is declared to be of a user-defined type, where the type defines a parameterized query. In the descriptions below, type names of the form ‘‘t_*x*’’ are type procedures.

> *Attributes of type REGPROC procedure are currently being used in lieu of a form of compact POSTQUEL, since POSTQUEL procedures are not yet implemented. The value of a REGPROC attribute is the* proname *of a registered procedure, represented interally by the OID of the PROC relation tuple associated with this procedure.*

A naming convention is followed for all relation attributes. The name of each attribute in a relation begins with the first three characters of the relation name. For example, the attribute *relname* in the RELATION relation contains the name of the relation, version, or index being defined.

Eight kinds of entities are defined in the catalogs: users, relations, data types, operators, access methods, rules, demons, and procedures. Each instance of an entity has a unique identifier. Foreign keys are represented by the unique identifier of the instance in the appropriate relation. The type of the foreign key is specified as the entity name for documentation purposes. The actual value stored will be a ‘‘oid’’. Oid’s values will be unique across all relations (including history and versions) in the database.

The remainder of this note defines the catalogs for representing these entities.

Data Type	**Description**
bool	boolean (stored as byte).
byte	uninterpreted byte
int2	2 byte integer.
u_int2	unsigned 2 byte integer.
int4	4 byte integer.
u_int4	unsigned 4 byte integer.
float4	4 byte floating point.
float8	8 byte floating point.
char	character.
text	variable length text.
abstime	absolute date and time.
reltime	relative date and time.
regproc	register procedure.
postquel	POSTQUEL commands.

Table 2. Predefined data types.

2. Users

```
USER(
        usename = char[16],             /* user's name */
        usesysid = int2,                /* user system id (UNIX uid) */
        usecreatedb = bool,             /* can user create databases? */
        usetrace = bool,                /* can user set trace flags? */
        usesuper = bool,                /* can user be super user? */
        usecatupd = bool                /* can user update catalogs? */
)
DATABASE(
        datname = char[16],             /* database name */
        datdba = USER,                  /* database administrator */
        datpath = text                  /* directory of database */
)
DEFAULTS(
        defname = char[16],             /* default name */
        defvalue = byte[16]             /* default value */
)
VARIABLE(
        varname = char[16],             /* variable name */
        varvalue = byte[]               /* variable value */
)
MAGIC(
        magname = char[16],             /* constant name */
        magvalue = byte[16]             /* constant value */
)
```

One copy of these catalogs will be maintained on each host. They will be mapped to all user databases.

3. Relations

```
RELATION(
        relname = char[16],             /* relation name */
        relowner = USER,                /* relation owner */
        relam = AM,                     /* access method */
        relpages = u_int4,              /* # pages */
        reltuples = u_int4,             /* # tuples */
        relexpires = abstime,           /* time after which tuples are deleted */
        relpreserved = reltime,         /* time after which tuples are deleted */
        relhasindex = bool,             /* is relation indexed? */
        relisshared = bool,             /* is relation shared? */
        relkind = char,                 /* kind of relation ...
                                         * oneof(``index'', ``relation'', ``special'',
                                         * ``uncataloged'', ``version'') */
        relarch = char,                 /* relation archive mode...
                                         * oneof(``heavy'', ``light'', ``none'') */
        relnatts = u_int2,              /* current number of attributes */
        relkey = int2[8],               /* attribute number of relation keys */
        relkeyop = OPERATOR[8],         /* oid's of key attr op's */
        reldesc = t_reldesc             /* cached relation descriptor */
        reclusterindex = RELATION           /* index on which relation is clustered */
)

ATTRIBUTE(
        attrelid = RELATION,                /* relation containing attribute */
        attname = char[16],                 /* attribute name */
        atttypid = TYPE,                    /* attribute type */
        attdefrel = RELATION,               /* relation that defines the attribute */
        attnvals = u_int4,                  /* # distinct values in column */
        atttyparg = TYPE,                   /* type arg for arrays */
        attlen = int2,                      /* length of attribute (negative means var.) */
        attnum = u_int2,                    /* attribute number (temp. int2--exec. bug) */
        attbound = u_int2,                  /* upper bound if array */
        attbyval = bool,                    /* passed by value (copied from type rel.) */
        attcanindex = bool,                 /* indexable domain */
        attproc = PROC                      /* procedure for attribute (spquel) */
)

INHERITS(
        inhrel = RELATION,                  /* inherting relation */
        inhparent = RELATION,               /* inherits from this parent */
        inhseqnum = int4                    /* inherits clause order */
)

INDEX(
        indexrelid = RELATION,              /* index relation */
        indrelid = RELATION,                /* relation being indexed */
        indkey = u_int2[8],                 /* index key attribute numbers */
        indclass = OPCLASS[8],              /* index key classes */
        indisclustered = bool,              /* is relation clustered on this index? */
        indisarchived = bool,               /* is the index archival? */
        inddesc = t_inddesc                 /* cached index descriptor */
)

VERSION(
        verrelid = RELATION,                /* version relation */
        verbaseid = RELATION,               /* base relation */
        vertime = abstime                   /* time for base relation */
)

STATISTIC(
        starelid = RELATION,                /* relation being described */
        staattnum = u_int2,                 /* attribute # */
        staop = OPERATOR,                   /* operator */
        stalokey = text,                    /* low value (updated by demon) */
        stahikey = text                     /* hi value (updated by demon) */
)
```

4. Data Types

```
TYPE(
        typname = char[16],               /* data type name */
        typowner = USER,                  /* data type owner */
        typlen = int2,                    /* rep size in bytes */
        typprtlen = int2,                 /* print rep size in bytes */
        typbyval = bool,                  /* passed by value */
        typisproc = bool,                 /* is type a procedure? */
        typprocid = PROC,                 /* proc oid if it is a procedure */
        typelem = TYPE,                   /* array element type */
        typinput = regproc,               /* blank portal input procedure */
        typoutput = regproc,              /* blank portal output procedure */
        typreceive = regproc,             /* input procedure for remote machine front ends */
        typsend = regproc,                /* output procedure for remote machine front ends*/
        typdefault = text                 /* default value */
)

OPERATOR(
        oprname = char[16],               /* operator name */
        oprowner = USER,                  /* operator owner */
        oprprec = u_int2,                 /* precedence (should be u_int1) */
        oprkind = char,                   /* kind of operator...
                                           * oneof(‘‘binary’’, ‘‘leftunary’’, ‘‘rightunary’’) */
        oprisleft = bool,                 /* is operator left associative? */
        oprcanhash = bool,                /* can hash-join use this operator? */
        oprleft = TYPE,                   /* left operand type */
        oprright = TYPE,                  /* right operand type */
        oprresult = TYPE,                 /* result type */
        oprcom = OPERATOR,                /* commutative operator */
        oprnegate = OPERATOR,             /* negated operator */
        oprlsortop = OPERATOR,            /* left merge-sort operator */
        oprrsortop = OPERATOR,            /* right merge-sort operator */
        oprcode = regproc,                /* code for operator */
        oprrest = regproc,                /* proc to estimate restriction selectivity */
        oprjoin = regproc                 /* proc to estimate join selectivity */
)

OPCLASS(
        opcname = char[16]                /* operator class name */
)
```

5. Access Methods

```
AM(
        amname = char[16],                /* access method name */
        amowner = USER,                   /* access method owner */
        amkind = char,                    /* kind of access method...
                                           * oneof(‘‘hashed’’, ‘‘ordered’’, ‘‘special’’) */
        amgettuple = regproc,             /* gettuple proc (required)  */
        aminsert = regproc,               /* insert proc (required) */
        amdelete = regproc,               /* delete proc (required) */
        amgetattr = regproc,              /* getattr proc (optional) */
        amsetlock = regproc,              /* setlock proc (optional) */
        amsettid = regproc,               /* settid proc (optional) */
        amfreetuple = regproc,            /* freetuple proc (optional) */
        ambeginscan = regproc,            /* beginscan proc (optional) */
        amrescan = regproc,               /* rescan proc (optional) */
        amendscan = regproc,              /* endscan proc (optional) */
        ammarkpos = regproc,              /* markpos proc (optional) */
        amrestrpos = regproc,             /* restrpos proc (optional) */
        amopen = regproc,                 /* open proc (optional) */
        amclose = regproc,                /* close proc (optional) */
        ambuild = regproc,                /* constructor proc (optional) */
        amcreate = regproc,               /* create proc (optional) */
        amdestroy = regproc               /* destroy proc (optional) */
        amnumslots = u_int2,              /* number of slots for this access method */
        amsortorder = u_int2,             /* slot number which defines ordering of tuples */
        ambuildoperator = u_int2,         /* use this operator to sort tuples
                                           * before building index */
)

AMOP(
        amopamid = AM,                    /* oid of access method */
        amopclaid = OPCLASS,              /* operator class */
        amopoprid = OPERATOR,             /* access method operator */
        amopstrategy = u_int2,            /* strategy number for access method */
        amopselect = regproc,             /* operator selectivity */
        amopnpages = regproc              /* # pages to be examined */
)
```

6. Rules

```
RULE(
        rulname = char[16],               /* rule name */
        rulowner = USER,                  /* rule owner */
        rulserver = SERVER,               /* server that invoked rule */
        rulportal = char[16],             /* portal name if alerter */
        rulpriority = u_int2,             /* rule priority */
        rulstatus = char,                 /* rule implementation status...
                                           * oneof(‘‘early’’, ‘‘late’’, ‘‘either-early’’,
                                           *      ‘‘either-late’’) */
        rulkind = char,                   /* kind of rule...
                                           * oneof(‘‘once’’, ‘‘never’’, ‘‘always’’) */
        rulstat1 = regproc,               /* statistic (to be filled in by spiros */
                                          /* postquel/PROC ??? */
        rulstat2 = regproc                /* statistic (to be filled in by spiros */
                                          /* postquel/PROC ??? */
)

RULEPLANS(
        rplrulid = RULE,                  /* the OID of the corresponding rule */
        rplnum = u_int2,                /* each rule may have many plans... */
        rplcode = postquel              /* plan code  <implemented as text for the time being> */
    )
```

7. Procedures

```
    PROC(
        proname = char[16],             /* procedure name */
        proowner = USER,                /* procedure owner */
        prolang = LANGUAGE,             /* proc language */
        proisinh = bool,                /* is procedure inheritable? */
        proistrusted = bool,            /* is proc spawned or called? */
        proiscachable = bool,           /* is value precomputable? */
        pronargs = u_int2,              /* number of args */
        prorettype = TYPE,              /* procedure return type */
        prosrc = text,                  /* procedure source */
        probin = byte[],                /* procedure binary */
        prodesc = t_procdesc            /* cached procedure descriptor */
    )

    LANGUAGE(
        lanname = char[16],             /* language name */
        lancompiler = text              /* compiler filename */
    )

    PARG(
        parproid = PROC,                /* procedure oid */
        parnum = u_int2,                /* arg # */
        parbound = u_int1,              /* array bound */
        partype = TYPE                  /* arg type */
    )

    AGGREGATE(
        aggname = char[16],             /* aggregate name */
        aggowner = USER,                /* aggregate owner */
        aggfun1 = regproc,              /* transition function */
        aggfun2 = regproc               /* completion function */
    )
    IPL(
        iplrel = RELATION,              /* relation for which ipl is defined */
        iplinherits = RELATION,         /* inherits from relation */
        iplseqnum = int4                /* ipl sequence number */
    )

    INHERITPROC(
        inhproname = char[16],          /* procedure name */
        inhargrel = RELATION,           /* argument relation */
        inhdefrel = RELATION,           /* defining relation */
        inhproc = PROC                  /* proc definition */
    )
```

8. Demons

```
DEMON(
        demserid = SERVER,          /* server oid */
        demname = char[16],         /* demon name */
        demowner = USER,            /* demon owner */
        demcode = regproc           /* demon code */
)

SERVER(
        sername = char[16],         /* unique process name */
        serpid = u_int2,            /* unix process id */
        serport = u_int2            /* unix communication port for this process */
)
```

Every POSTGRES backend (i.e., server) will have an entry in SERVER. Every demon and the POSTMASTER will have an entry in DEMON.