The Art of PostgreSQL

Turn Thousands of Lines of Code into Simple Queries

Postgres Extensibility

Université Libre de Bruxelles

PostgreSQL Extensiblity

The Design of POSTGRES

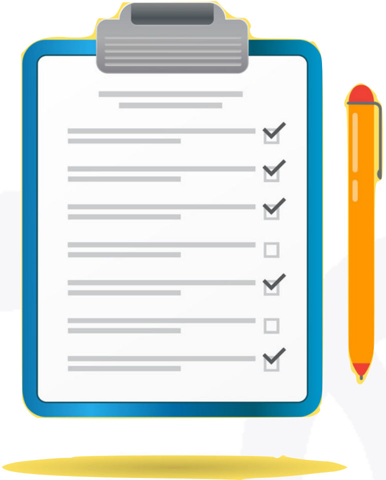
This paper presents the preliminary design of a new database management system, called POSTGRES, that is the successor to the INGRES relational database system. The main design goals of the new system are to:

1. provide better support for complex objects,
2. provide user extendibility for data types, operators and access methods,
3. provide facilities for active databases (i.e., alerters and triggers) and inferencing including forward- and backward-chaining,
4. simplify the DBMS code for crash recovery,
5. produce a design that can take advantage of optical disks, workstations composed of multiple tightly-coupled processors, and custom designed VLSI chips, and
6. make as few changes as possible (preferably none) to the relational model.

The paper describes the query language, programming language interface, system architecture, query processing strategy, and storage system for the new system.

PostgreSQL Extensibility

- *SQL can be Object Oriented (when using Postgres)*
- *Extensible SQL*
- *Understanding Postgres data types*
- *And Operators*
- *And Operator Classes*
- *And Postgres Indexing APIs*
- *And Extensions*
- *Then developing a new extension, in C, re-using Postgres internals as much as possible*



Extensibility - SQL

Extensibility - SQL

```
select col1, col2 from table where col1 = ’something’;
```

For Postgres to return those, it must first understand their data types

non decorated literal, Postgres will need to find the data type on it's own

Extensibility - SQL

```
SELECT col
  FROM table
WHERE stamped > date ’today’ - interval ’1 day’;
```

decoration of the literal value ⇒ decorated literal

↳ prefixed with the name of the data type

Object Oriented SQL when using Postgres

Extensibility - SQL

```
select x,
       1 + x as "1+",
       '127.0.0.1'::inet + x as "ip address",
       date 'today' + x as date
  from (values (0), (1), (2), (3)) as t(x);
```

add an integer will have a different effect depending on the datatype of the other operand

```
 x | 1+ | ip address |    date
═══╪════╪════════════╪════════════
 0 |  1 | 127.0.0.1  | 2018-03-22
 1 |  2 | 127.0.0.2  | 2018-03-23
 2 |  3 | 127.0.0.3  | 2018-03-24
 3 |  4 | 127.0.0.4  | 2018-03-25
(4 rows)
```

Note: same type of catalog look up for a user created datatype and for base Postgres type

// different method depending on the type of object

Extensibility - SQL

ip4r deals with ranges of IP addresses

```
select iprange, locid
  from geolite.blocks
where iprange >>= '91.121.37.122';
```

? → new operator for "contains" → does the range contain the IP address

```
          iprange           | locid
----------------------------+-------
 91.121.0.0-91.121.159.255 | 75 (1 row)
```

comes from the extension (not natively in Postgres)

```
Time: 1.220 ms
```

Operator Classes

Postgres has no hardcoded knowledge, it is sorted in different catalogs that it has to look-up to understand the queries

organize a set of functions as an API allowing for creating indexes

```
select amopopr::regoperator
  from pg_opclass c
  join pg_am am
     on am.oid = c.opcmethod
   join pg_amop amop
     on amop.amopfamily = c.opcfamily
  where opcintype = 'ip4r'::regtype
    and am.amname = 'gist';
```

access method

① here it is looking for the contain operator

```
     amopopr
----------------
 >>=(ip4r,ip4r)
 <<=(ip4r,ip4r)
  >>(ip4r,ip4r)
  <<(ip4r,ip4r)
  &&(ip4r,ip4r)
   =(ip4r,ip4r)

(6 rows)
```

② look-up to find the code of the operator

③ look-up if there is an index on the column

when reading a query, Postgres does catalog look-ups to recognize and understand the different parts of the query

Extensibility - SQL

```
   select id, name, pos,
          pos <@> point(-6.25,53.34) as miles
     from pubnames
 order by pos <-> point(-6.25,53.34)
    limit 10;
```

"k- nearest neighbours"

Extensibility - Catalogs

```
  SELECT CASE WHEN o.oprkind='l' THEN NULL
              ELSE pg_catalog.format_type(o.oprleft, NULL)
          END AS "Left arg type",

         CASE WHEN o.oprkind='r' THEN NULL
              ELSE pg_catalog.format_type(o.oprright, NULL)
          END AS "Right arg type",

         pg_catalog.format_type(o.oprresult, NULL) AS "Result type",

         oprcode::regprocedure as function

    FROM pg_catalog.pg_operator o
         LEFT JOIN pg_catalog.pg_namespace n ON n.oid = o.oprnamespace
         LEFT JOIN pg_catalog.pg_type l ON l.oid = o.oprleft

   WHERE o.oprname = '+'
     AND l.typname in ('integer', 'inet', 'date')
     AND pg_catalog.pg_operator_is_visible(o.oid)

ORDER BY 1, 2, 3, 4;
```

Extensibility - Catalogs

```
 Left arg type │     Right arg type     │         Result type         │                 function
═══════════════╪════════════════════════╪═════════════════════════════╪══════════════════════════════════════════
 date          │ integer                │ date                        │ date_pli(date,integer)
 date          │ interval               │ timestamp without time zone │ date_pl_interval(date,interval)
 date          │ time with time zone    │ timestamp with time zone    │ datetimetz_pl(date,time with time zone)
 date          │ time without time zone │ timestamp without time zone │ datetime_pl(date,time without time zone)
 inet          │ bigint                 │ inet                        │ inetpl(inet,bigint)
(5 rows)
```

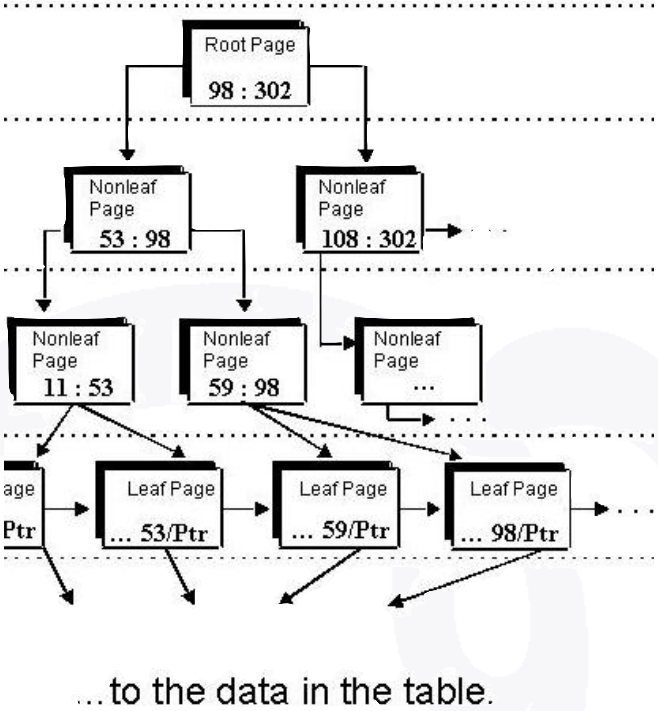
Extensibility - Data Types and Indexes

- *Data types*
- *Input/Output Functions*
- *Casts (implicit/explicit)*
- *Operator Classes*
- *Operator Families*

Extensibility - Indexes

Extensibility - Indexes

- *BTree*
- *GiST, Generalized Search Tree*
- *SP-GiST, Space Partitioned Tree*
- *GIN, Generalized Inverted Index*
- *And more (BRIN, bloom, …)*



Extensibility - Indexes and constraints

```
CREATE TABLE reservation
(
  room      text,
  professor text,
  during    period,

  EXCLUDE USING gist
  (
      room with =,
    during with &&
  )
);
```

twice the same room
with overlapping periods

```
CREATE TABLE circles (
  c circle,
  EXCLUDE USING gist (c WITH &&)
);
```

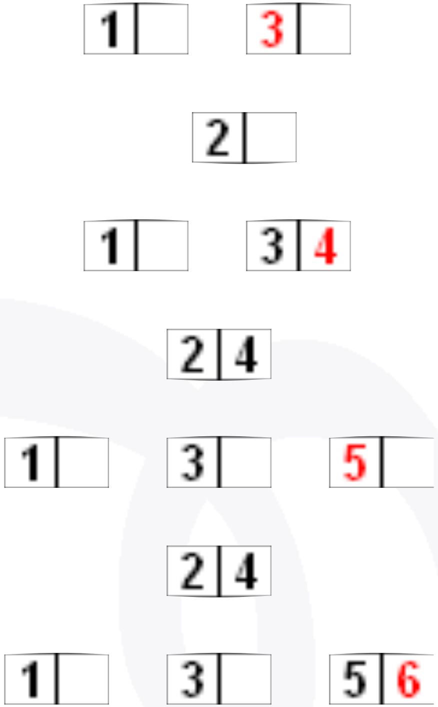
overlapping

Indexes - BTree

- *Built for Speed*
- *Unique concurrency tricks*
- *Balanced*
- *Support Function:* `cmp`
- *Operators:* `<=  <  =  >  >=`

//strcmp

total ordering



generalized
⇒ slower than B-tree
(that are focused

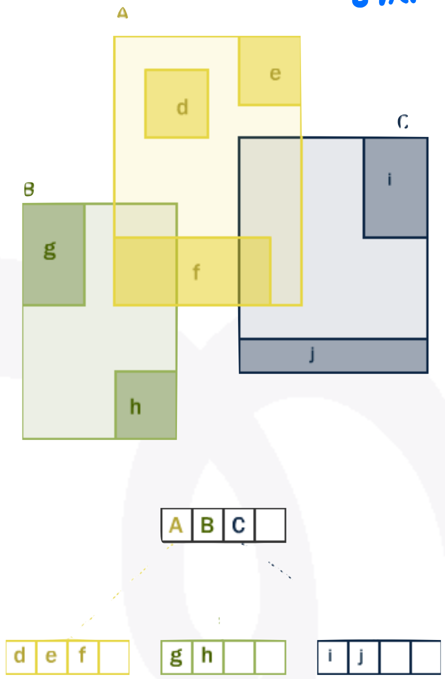
can't be implemented
using B-tree
because not
strictly greater,...

Indexes - GiST

- *Built for comfort*
- *Balanced*
- *Support Functions:*
  - consistent, same, union
  - penalty, pick split
  - compress, decompress
- *Operators:*

@> <@ && = &< &> <<| ...

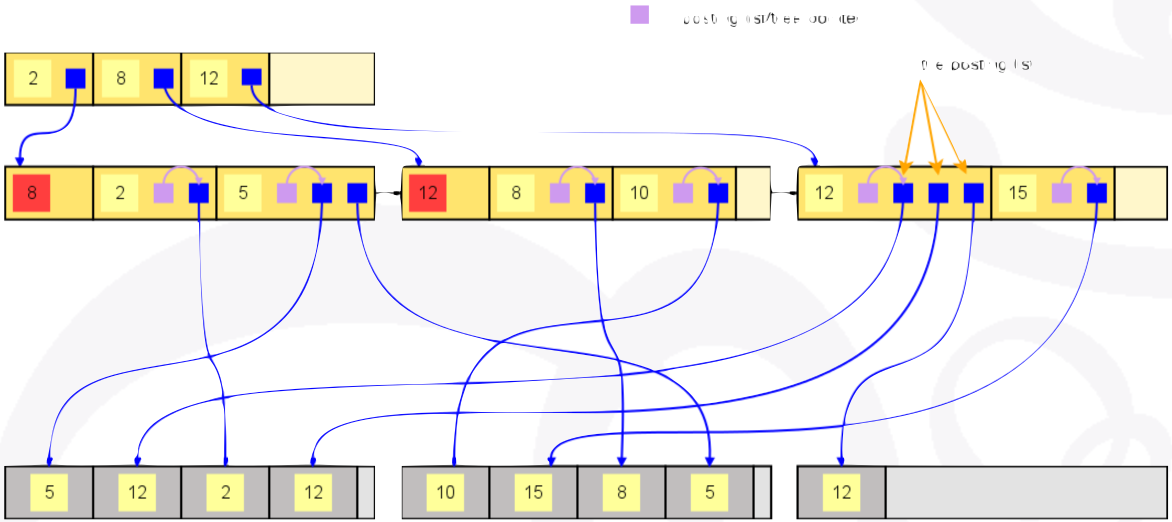
R-tree Hierarchy



Indexes - GIN

- *Built for Text Search, arrays, JSON*
- *Balanced*
- *Support Functions:*
  - compare, consistent
  - extractValue, extractQuery
- *Operators:*

`@>  <@  &&  =`

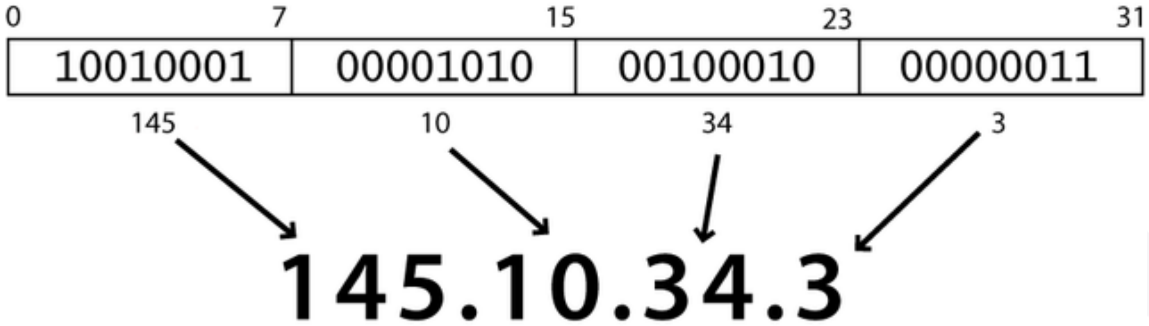


Postgres Extensions

Geolocation with Postgres

- *Using ip4r extension*
- *GiST Indexes*
- *kNN searches*

Ip address ranges



0 7 15 23 31

10010001 00001010 00100010 00000011

145 10 34 3

145.10.34.3

Ip address ranges

```
table geolite.blocks limit 10;

         iprange          | locid
--------------------------+-------
 1.0.0.0/24               |    17
 1.0.1.0-1.0.3.255        |    49
 1.0.4.0/23               | 14409
 1.0.6.0/23               |    17
 1.0.8.0/21               |    49
 1.0.16.0/20              | 14614
 1.0.32.0/19              | 47667
 1.0.64.0/18              |   111
 1.0.128.0-1.0.147.255    |   209
 1.0.148.0/24             | 22537
(10 rows)
```

Geolocation in Postgres

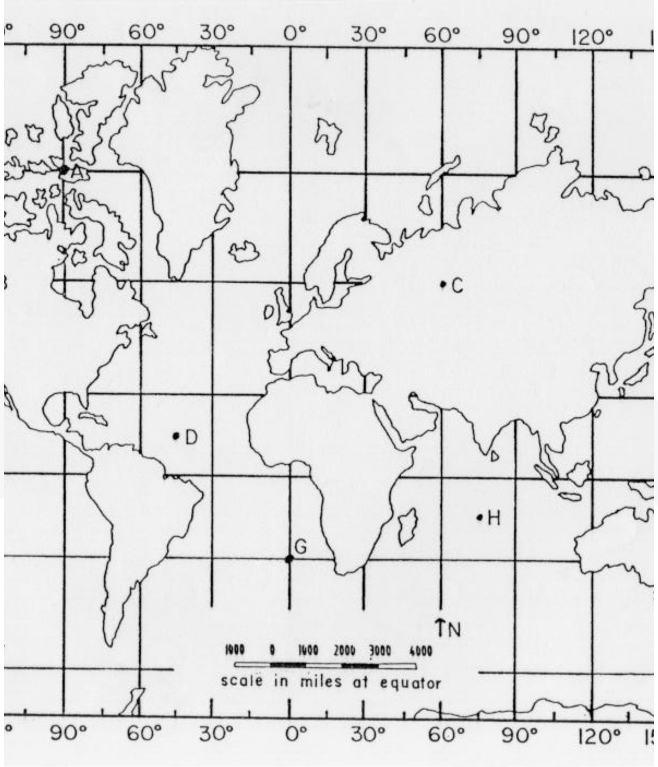
```
select *
  from geolite.blocks
  join geolite.location
       using(locid)
 where iprange
   >>= '74.125.195.147';
```

```
-[ RECORD 1 ]------------------------------
locid      | 2703
iprange    | 74.125.189.24-74.125.255.255
country    | US
region     | CA
city       | Mountain View
postalcode | 94043
location   | (-122.0574,37.4192)
metrocode  | 807
areacode   | 650

Time: 1.335 ms
```

Geolocation - type point

```
CREATE TABLE pubnames
    (
      id   bigint,
      pos  POINT,
      name text
);
```



How far is the nearest pub

```
create extension cube;
create extension earthdistance;
```

```
  select name,
         pos <@> point(-6.25,53.34) miles
    from pubnames
order by pos <-> point(-6.25,53.34)
limit 3;
```

```
  name   |       pos
---------+----------------
 Ned's   | (-6.25,53.34)
 Sub Lo  | (-6.25,53.34)
 O'Neil  | (-6.25,53.34)
(3 rows)

Time: 0.849 ms
```

Geolocation: ip4r meets earthdistance

Ten nearest pubs

```
with geoloc as
 (
  select location as l
    from location
    join blocks using(locid)
   where iprange
         >>=
         ’212.58.251.195’
 )
  select name,
         pos <@> l miles
    from pubnames, geoloc
order by pos <-> l limit 10;
```

```
        name        | miles
--------------------+-------
 Blue Anchor        | 0.299
 Dukes Head         | 0.360
 Blue Ball          | 0.337
 Bell (aka The Rat) | 0.481
 on the Green       | 0.602
 Fox & Hounds       | 0.549
 Chequers           | 0.712
 Sportsman          | 1.377
 Kingswood Arms     | 1.205
 Tattenham Corner   | 2.007
(10 rows)

Time: 3.275 ms
```

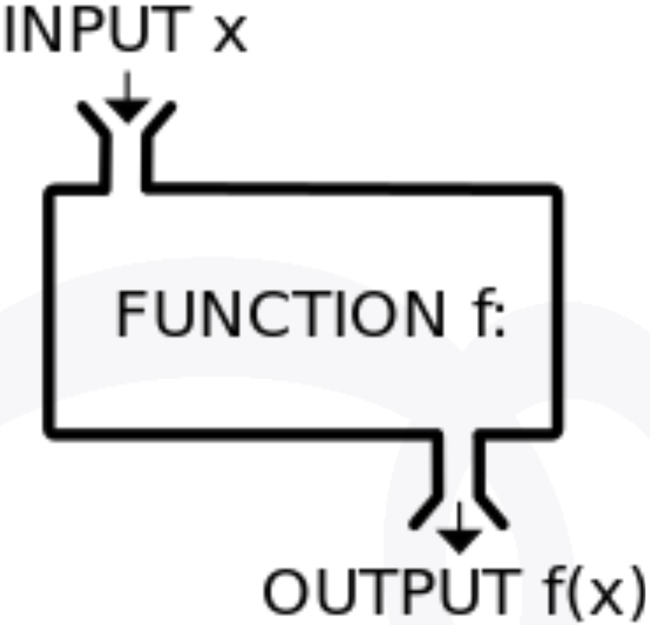
Create Extension

Extension base36

https://github.com/dimitri/base36

New integer data type: base36

- *Internally a bingint*
- *Visually, a base36 number*
- *Re-use Postgres internals*
- *Provide new I/O functions*



A new integer data type: base36

```
create extension base36;
```

```
 i  | x
----+----
  0 | 0
  1 | 1
  2 | 2
  3 | 3
  4 | 4
  5 | 5
  6 | 6
  7 | 7
  8 | 8
  9 | 9
 10 | A
```

```
   i   |  x
-------+-----
 10000 | 7PS
 10001 | 7PT
 10002 | 7PU
 10003 | 7PV
 10004 | 7PW
 10005 | 7PX
 10006 | 7PY
 10007 | 7PZ
 10008 | 7Q0
 10009 | 7Q1
 10010 | 7Q2
```

```
     i     |   x
-----------+--------
 100000000 | 1NJCHS
 100000001 | 1NJCHT
 100000002 | 1NJCHU
 100000003 | 1NJCHV
 100000004 | 1NJCHW
 100000005 | 1NJCHX
 100000006 | 1NJCHY
 100000007 | 1NJCHZ
 100000008 | 1NJCI0
 100000009 | 1NJCI1
 100000010 | 1NJCI2
```

Using the base36 extension

```
create extension base36;

create table demo(i bigint, x base36);

insert into demo(i, x)
     select n, n::bigint
      from generate_series(0, 10) t(n);


insert into demo(i, x)
     select n, n::bigint
       from generate_series(10000, 10010) t(n);


insert into demo(i, x)
     select n, n::bigint
       from generate_series(100000000, 100000010) t(n);


create index on demo(x);
```

Extensibility - I/O functions

```
CREATE OR REPLACE FUNCTION base36_in(cstring) RETURNS base36
AS ’$libdir/base36’
LANGUAGE C IMMUTABLE STRICT;


CREATE OR REPLACE FUNCTION base36_out(base36) RETURNS cstring
AS ’$libdir/base36’
LANGUAGE C IMMUTABLE STRICT;
```

Extensibility - I/O functions

```
CREATE OR REPLACE FUNCTION base36_recv(internal) RETURNS base36
AS ’$libdir/base36’
LANGUAGE C IMMUTABLE STRICT;


CREATE OR REPLACE FUNCTION base36_send(base36) RETURNS bytea
AS ’$libdir/base36’
LANGUAGE C IMMUTABLE STRICT;
```

Extensibility - I/O functions

```
#include "postgres.h"
#ifndef PG_VERSION_NUM
#error "Unsupported too old PostgreSQL version”
#endif

#if PG_VERSION_NUM / 100 != 903 \
 && PG_VERSION_NUM / 100 != 904
#error "Unknown or unsupported PostgreSQL version"
#endif


PG_MODULE_MAGIC;
```

Extensibility - I/O functions

```
static inline
base36 base36_from_str(const char *str)
{
    /* ... C code here ... */
}

static inline
char *base36_to_str(base36 c)
{
    /* ... C code here ... */
}
```

Extensibility - I/O functions

```
Datum base36_in(PG_FUNCTION_ARGS);
Datum base36_out(PG_FUNCTION_ARGS);
Datum base36_recv(PG_FUNCTION_ARGS);
Datum base36_send(PG_FUNCTION_ARGS);
Datum base36_cast_to_text(PG_FUNCTION_ARGS);
Datum base36_cast_from_text(PG_FUNCTION_ARGS);
Datum base36_cast_to_bigint(PG_FUNCTION_ARGS);
Datum base36_cast_from_bigint(PG_FUNCTION_ARGS);
```

Extensibility - I/O functions

```
PG_FUNCTION_INFO_V1(base36_in);

Datum base36_in(PG_FUNCTION_ARGS)
{
    char *str = PG_GETARG_CSTRING(0);
    PG_RETURN_INT64(base36_from_str(str));
}

PG_FUNCTION_INFO_V1(base36_out);

Datum base36_out(PG_FUNCTION_ARGS)
{
    base36 c = PG_GETARG_INT64(0);
    PG_RETURN_CSTRING(base36_to_str(c));
}
```

Extensibility - I/O functions

```
CREATE TYPE base36
(
        INPUT    = base36_in,
        OUTPUT   = base36_out,
        RECEIVE  = base36_recv,
        SEND     = base36_send,
        LIKE     = bigint,
        CATEGORY = ’N’
);

COMMENT ON TYPE base36
     IS ’bigint written in base36: [0-9A-Z]+’;
```

Extensibility - I/O functions

```
CREATE FUNCTION text(base36)
RETURNS text
AS ’$libdir/base36’,
’base36_cast_to_text’
LANGUAGE C IMMUTABLE STRICT;
```

```
CREATE CAST (text as base36)
  WITH FUNCTION base36(text)
AS IMPLICIT;

CREATE CAST (base36 as text)
  WITH FUNCTION text(base36);

CREATE CAST (bigint as base36)
    WITHOUT FUNCTION
         AS IMPLICIT;

CREATE CAST (base36 as bigint)
    WITHOUT FUNCTION
         AS IMPLICIT;
```

Reuse Postgres Internals

```
CREATE OR REPLACE FUNCTION base36_eq(base36, base36)
RETURNS boolean LANGUAGE internal IMMUTABLE AS ’int8eq’;

CREATE OR REPLACE FUNCTION base36_ne(base36, base36)
RETURNS boolean LANGUAGE internal IMMUTABLE AS ’int8ne’;

CREATE OR REPLACE FUNCTION base36_lt(base36, base36)
RETURNS boolean LANGUAGE internal IMMUTABLE AS ’int8lt’;

CREATE OR REPLACE FUNCTION base36_le(base36, base36)
RETURNS boolean LANGUAGE internal IMMUTABLE AS ’int8le’;
```

Reuse Postgres Internals

```
CREATE OR REPLACE FUNCTION base36_gt(base36, base36)
RETURNS boolean LANGUAGE internal IMMUTABLE AS ’int8gt’;

CREATE OR REPLACE FUNCTION base36_ge(base36, base36)
RETURNS boolean LANGUAGE internal IMMUTABLE AS ’int8ge’;

CREATE OR REPLACE FUNCTION base36_cmp(base36, base36)
RETURNS integer LANGUAGE internal IMMUTABLE AS ’btint8cmp’;
```

Extensibility - Operators

```
CREATE OPERATOR = (
          LEFTARG = base36,
         RIGHTARG = base36,
        PROCEDURE = base36_eq,
       COMMUTATOR = ’=’,
          NEGATOR = ’<>’,
         RESTRICT = eqsel,
             JOIN = eqjoinsel
);

COMMENT ON OPERATOR =(base36, base36) IS ’equals?’;
```

Extensibility - Operator Class

```
CREATE OPERATOR CLASS btree_base36_ops
DEFAULT FOR TYPE base36 USING btree
AS
     OPERATOR        1 <,
     OPERATOR        2 <=,
     OPERATOR        3 =,
     OPERATOR        4 >=,
     OPERATOR        5 >,
     FUNCTION        1 base36_cmp(base36, base36);
```