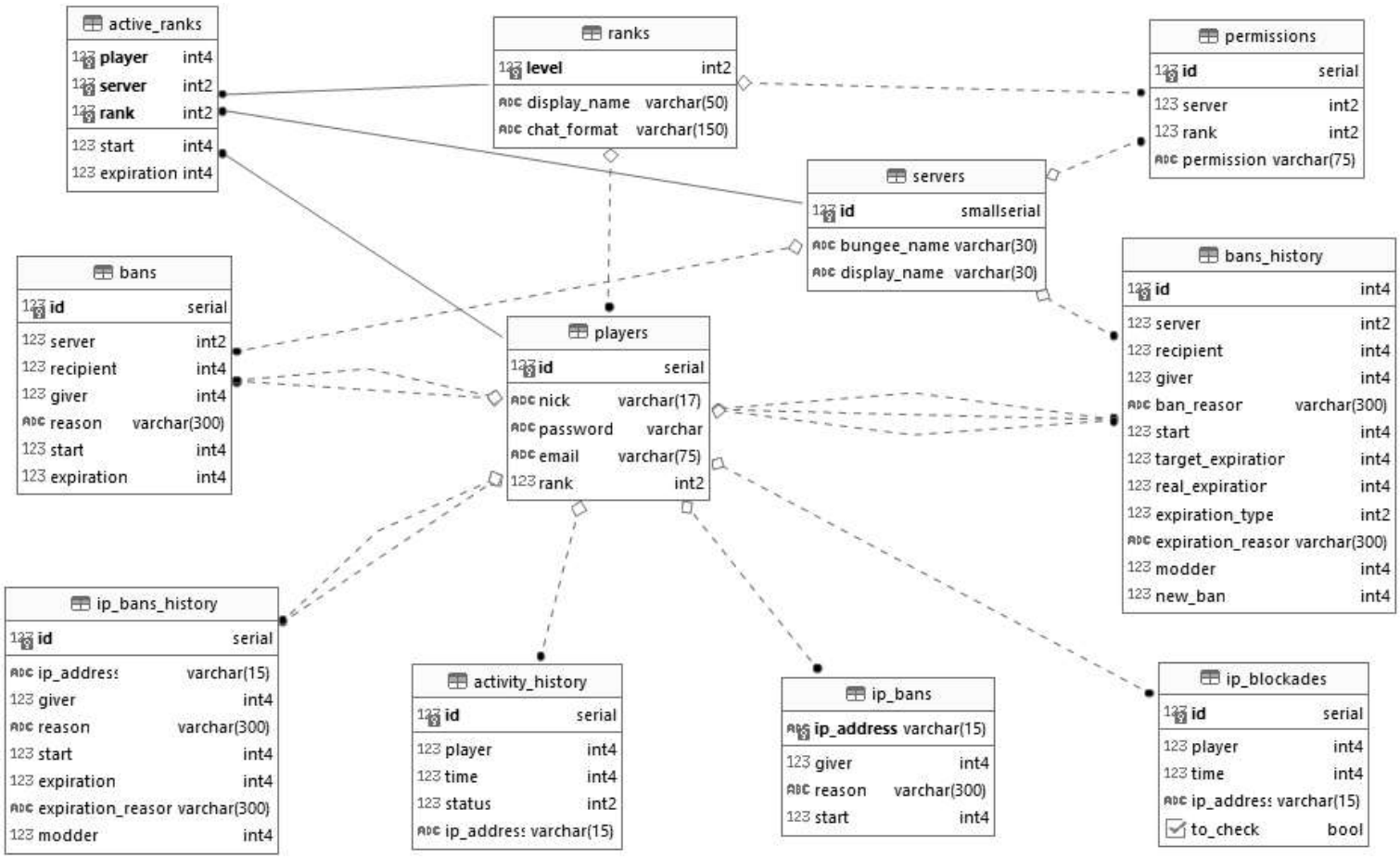
Server's Documentation

Created by **Thonem** and **KeroYs**

1. Data Base

1. ER diagram



2. Tables

1. servers

```
CREATE TABLE IF NOT EXISTS servers
(
    id           smallserial PRIMARY KEY,
    bungee_name  varchar(75) UNIQUE NOT NULL,

    CHECK (LENGTH(bungee_name) > 0)
);
```

```
INSERT INTO servers VALUES (-1, 'Sieć');
INSERT INTO servers VALUES (-2, 'Strona');
```

2. global_ranks

```
CREATE TABLE IF NOT EXISTS global_ranks
(
    id           smallserial PRIMARY KEY,
    level        smallint UNIQUE     NOT NULL,
    display_name varchar(75) UNIQUE  NOT NULL,
    chat_format  varchar(200)        NOT NULL,

    CHECK (level > 0),
    CHECK (LENGTH(display_name) > 0),
    CHECK (LENGTH(chat_format) > 0)
);
```

```
INSERT INTO global_ranks VALUES (-1, 32767, '&4Właściciel',
      '[{{RANK}}] {{NICK}} : {{MSG}}');
```

```
INSERT INTO global_ranks VALUES (-2, 1000, '&7Gracz', '[{{RANK}}]
     {{NICK}} : {{MSG}}');
```

3. server_ranks

```
CREATE TABLE IF NOT EXISTS server_ranks
(
    id           smallserial PRIMARY KEY,
    level        smallint            NOT NULL,
    server       smallint            NOT NULL,
    display_name varchar(100)        NOT NULL,
    chat_format  varchar(300)        NOT NULL,

    UNIQUE (level, server),
    UNIQUE (server, display_name),

    CHECK (level > 0),
    CHECK (LENGTH(display_name) > 0),
    CHECK (LENGTH(chat_format) > 0),

    FOREIGN KEY (server) REFERENCES servers(id)
        ON DELETE CASCADE
);
```

4. players

```
CREATE TABLE IF NOT EXISTS players
(
    id       serial PRIMARY KEY,
```

```
    nick     varchar(16) UNIQUE               NOT NULL,
    password varchar                          NOT NULL,
    email    varchar(75) DEFAULT NULL UNIQUE,
    rank     smallint DEFAULT 1000            NOT NULL,

    CHECK (LENGTH(nick) > 2 AND nick NOT LIKE '%[^A-Za-z0-9_]%'),
    CHECK (LENGTH(password) >= 29),
    CHECK (LENGTH(email) > 4 AND email LIKE '%@%'),

    FOREIGN KEY (rank) REFERENCES global_ranks(id)
        ON DELETE SET DEFAULT
);
```

```
INSERT INTO players VALUES (-1, 'KONSOLA',
'----------------------------', DEFAULT, -1);
```

5. active_ranks

```
CREATE TABLE IF NOT EXISTS active_ranks
(
    id         serial PRIMARY KEY,
    player     int            NOT NULL,
    rank       smallint       NOT NULL,
    start      timestamp      NOT NULL,
    expiration timestamp,

    UNIQUE (player, rank),
```

```
    CHECK (expiration IS NULL OR start < expiration),

    FOREIGN KEY (player) REFERENCES players(id)
        ON DELETE CASCADE,
    FOREIGN KEY (rank) REFERENCES server_ranks(id)
        ON DELETE CASCADE
);
```

6. global_rank_permissions

```
CREATE TABLE IF NOT EXISTS global_rank_permissions
(
   rank       smallint    NOT NULL,
   server     smallint    NOT NULL,
   permission varchar(100) NOT NULL,

   PRIMARY KEY (rank, server, permission),

   CHECK (LENGTH(permission) > 0 AND permission SIMILAR TO
           '!?[A-Za-z0-9]+(.[A-Za-z0-9])*'),

   FOREIGN KEY (rank) REFERENCES global_ranks(id)
       ON DELETE CASCADE
       ON UPDATE CASCADE,

   FOREIGN KEY (server) REFERENCES servers(id)
       ON DELETE CASCADE
);
```

7. server_rank_permissions

```
CREATE TABLE IF NOT EXISTS server_rank_permissions
(
   rank       smallint    NOT NULL,
   permission varchar(100) NOT NULL,

   PRIMARY KEY (rank, permission),

   CHECK (LENGTH(permission) > 0 AND permission SIMILAR TO
            '!?[A-Za-z0-9]+(.[A-Za-z0-9])*'),

   FOREIGN KEY (rank) REFERENCES global_ranks(level)
       ON DELETE CASCADE
       ON UPDATE CASCADE
);
```

8. activity_history

```
CREATE TABLE IF NOT EXISTS activity_history
(
    id         serial PRIMARY KEY,
    player     int         NOT NULL,
    time       int         NOT NULL,
    status     smallint    NOT NULL,
    ip_address varchar(15) NOT NULL
);

ALTER TABLE activity_history ADD CONSTRAINT activity_history__ck__time
           CHECK (
    time > 0
);
ALTER TABLE activity_history ADD CONSTRAINT
           activity_history__ck__status CHECK (
    status > 0 AND status < 4
```

```
);
ALTER TABLE activity_history ADD CONSTRAINT
           activity_history__ck__ip_address CHECK (
    LENGTH(ip_address) > 6
);

ALTER TABLE activity_history ADD CONSTRAINT
           activity_history__fk__player FOREIGN KEY (player)
    REFERENCES players (id)
    ON DELETE CASCADE;
```

Status codes:

1 - Registered
2 - Successfully logged in to the server
3 - Unsuccessfully attempted to log in to the server
4 - Successfully logged in on the website
5 - Unsuccessfully attempted to log in on the website
6 - Password changed
7 - Email set

9. ip_blockades

```
CREATE TABLE IF NOT EXISTS ip_blockades
(
    id         serial PRIMARY KEY,
    player     int                  NOT NULL,
    time       int                  NOT NULL,
    ip_address varchar(15)          NOT NULL,
```

```
    to_check   boolean DEFAULT false NOT NULL
);

ALTER TABLE ip_blockades ADD CONSTRAINT ip_blockades__ck__time CHECK (
    time > 0
);
ALTER TABLE ip_blockades ADD CONSTRAINT ip_blockades__ck__ip_address
CHECK (
    LENGTH(ip_address) > 6
);

ALTER TABLE ip_blockades ADD CONSTRAINT ip_blockades__fk__player
FOREIGN KEY (player)
    REFERENCES players (id)
    ON DELETE CASCADE;
```

10. bans

```
CREATE TABLE IF NOT EXISTS bans
(
   id         serial PRIMARY KEY,
   server     smallint       NOT NULL REFERENCES servers (id),
   recipient  int            NOT NULL REFERENCES players (id) CHECK
            (recipient > 0),
```

```
    giver      int DEFAULT -1 NOT NULL REFERENCES players (id),
    reason     varchar(300)   NOT NULL CHECK (LENGTH(reason) > 0),
    start      timestamp      NOT NULL,
    expiration timestamp,
    CHECK (expiration IS NULL OR start < expiration)
);

CREATE INDEX ON bans (recipient);
CREATE INDEX ON bans (giver);
CREATE INDEX ON bans (start);
CREATE INDEX ON bans (expiration);
```

11. bans_history

```
CREATE TABLE IF NOT EXISTS bans_history
(
 id                int PRIMARY KEY,
 server            smallint       NOT NULL REFERENCES servers (id) ON
DELETE CASCADE,
 recipient         int            NOT NULL REFERENCES players (id) ON
DELETE CASCADE CHECK (recipient > 0),
 giver             int DEFAULT -1 NOT NULL REFERENCES players (id) ON
DELETE SET DEFAULT,
 ban_reason        varchar(300)   NOT NULL CHECK (LENGTH(ban_reason) > 0),
 start             timestamp      NOT NULL,
 target_expiration timestamp,
 actual_expiration timestamp      NOT NULL,
 historic_type     smallint       NOT NULL,
 expiration_reason varchar(300) CHECK (expiration_reason IS NULL OR
LENGTH(expiration_reason) > 0),
 modder            int DEFAULT -1 REFERENCES players (id) ON DELETE SET
DEFAULT,
 new_ban           int CHECK (new_ban > 0),
 CHECK (target_expiration IS NULL OR start < target_expiration),
 CHECK (start < actual_expiration),
```

```
 CHECK (target_expiration IS NULL OR actual_expiration <
target_expiration),
 CHECK (
     (historic_type = 1 AND target_expiration IS NOT NULL AND
expiration_reason IS NULL AND modder IS NULL AND new_ban IS NULL) OR
     (historic_type = 2 AND expiration_reason IS NOT NULL AND modder IS
NOT NULL AND new_ban IS NULL) OR
     (historic_type = 3 AND expiration_reason IS NOT NULL AND modder IS
NOT NULL AND new_ban IS NOT NULL)
 ),
 CHECK (new_ban <> id)
);

CREATE INDEX ON bans_history (recipient);
CREATE INDEX ON bans_history (giver);
CREATE INDEX ON bans_history (start);
```

expiration_reason:

1 - expired
2 - canceled
3 - modified

12. ip_bans

```
CREATE TABLE IF NOT EXISTS ip_bans
(
    ip_address varchar(15) PRIMARY KEY,
    giver      int DEFAULT -1 NOT NULL,
    reason     varchar(300)   NOT NULL,
    start      int            NOT NULL
);

ALTER TABLE ip_bans ADD CONSTRAINT ip_bans__ck__ip_address CHECK (
    LENGTH(ip_address) > 6
);
ALTER TABLE ip_bans ADD CONSTRAINT ip_bans__ck__reason CHECK (
    LENGTH(reason) > 0
);
ALTER TABLE ip_bans ADD CONSTRAINT ip_bans__ck__start CHECK (
    start > 0
);

ALTER TABLE ip_bans ADD CONSTRAINT ip_bans__fk__giver FOREIGN KEY
           (giver)
    REFERENCES players (id)
    ON DELETE SET DEFAULT;
```

13. ip_bans_history

```
CREATE TABLE IF NOT EXISTS ip_bans_history
(
    id                serial PRIMARY KEY,
    ip_address        varchar(15)    NOT NULL,
    giver             int DEFAULT -1 NOT NULL,
    ban_reason        varchar(300)   NOT NULL,
    start             int            NOT NULL,
    expiration        int            NOT NULL,
    expiration_reason varchar(300)   NOT NULL,
    modder            int DEFAULT -1 NOT NULL
);

ALTER TABLE ip_bans_history ADD CONSTRAINT
           ip_bans_history__ck__ip_address CHECK (
    LENGTH(ip_address) > 6
);
ALTER TABLE ip_bans_history ADD CONSTRAINT
           ip_bans_history__ck__ban_reason CHECK (
    LENGTH(ban_reason) > 0
);
ALTER TABLE ip_bans_history ADD CONSTRAINT
           ip_bans_history__ck__startUexpiration CHECK (
    start > 0 AND start < expiration
);
ALTER TABLE ip_bans_history ADD CONSTRAINT
           ip_bans_history__ck__expiration_reason CHECK (
    LENGTH(expiration_reason) > 0
);

ALTER TABLE ip_bans_history ADD CONSTRAINT ip_bans_history__fk__giver
           FOREIGN KEY (giver)
    REFERENCES players (id)
    ON DELETE SET DEFAULT;
ALTER TABLE ip_bans_history ADD CONSTRAINT ip_bans_history__fk__modder
           FOREIGN KEY (modder)
    REFERENCES players (id)
    ON DELETE SET DEFAULT;
```

14. friendships

```
CREATE TABLE IF NOT EXISTS friendships
(
    id serial PRIMARY KEY,
    friend_1 int NOT NULL REFERENCES players (id) ON DELETE CASCADE CHECK
            (friend_1 > 0),
    friend_2 INT NOT NULL REFERENCES players (id) ON DELETE CASCADE CHECK
            (friend_2 > 0)
);

CREATE INDEX ON friendships (friend_1);
CREATE INDEX ON friendships (friend_2);
```

2. Special Name Formatting

1. Styling

All the properties that are called ‘display name’ can contain standard Minecraft syntax like ‘&c’. User interface must style text accordingly or ignore it, erasing special characters and codes.

Example:

```
ex&cample
```

Result:

```
example
```

2. Ranks

‘Colour’ specifies the colour associated with the rank. It is a single character of one of 16 Minecraft colours.

‘Chat format’ property of the rank can be styled in the same way as described above. What is more, it can contains special structure that user interface must replace with specific values. The structure looks like this:

```
{{NAME}}
```

where ‘NAME’ is replaced with one of the following (case matters):

- COLOUR - colour of the rank with ‘&’ sign before it
- PLAYER - nick of the player that wields the rank
- RANK - display name of the rank
- MSG - content of the message sent on chat

If ‘MSG’ is not present in the entire ‘Char format’ formula, the content of the message should be injected at the end.

```
DROP TABLE ip_bans_history;
DROP TABLE ip_bans;
DROP TABLE bans_history;
DROP TABLE bans;
DROP TABLE ip_blockades;
DROP TABLE activity_history;
DROP TABLE permissions;
DROP TABLE active_ranks;
DROP TABLE players;
DROP TABLE ranks;
DROP TABLE servers;
```

TRIGGERS

```
CREATE FUNCTION server_ranks_on_insert() RETURNS trigger AS $$
BEGIN
  IF EXISTS(SELECT level FROM global_ranks WHERE level = NEW.level)
THEN
      RAISE EXCEPTION 'Cannot insert a new server rank with the same
level as an existing global rank';
  END IF;
```

```
  RETURN NEW;
END
$$ LANGUAGE plpgsql;
CREATE TRIGGER server_ranks_on_insert
  BEFORE INSERT ON server_ranks
  FOR EACH ROW
  EXECUTE FUNCTION server_ranks_on_insert();
```

```
CREATE FUNCTION server_ranks_on_update() RETURNS trigger AS $$
BEGIN
  IF EXISTS(SELECT level FROM global_ranks WHERE level = NEW.level)
THEN
      RAISE EXCEPTION 'Cannot update a server rank with the same level
as an existing global rank';
  END IF;

  RETURN NEW;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER server_ranks_on_update
  BEFORE UPDATE ON server_ranks
  FOR EACH ROW
  WHEN (OLD.level IS DISTINCT FROM NEW.level)
  EXECUTE FUNCTION server_ranks_on_update();
```

```
CREATE FUNCTION global_ranks_on_insert() RETURNS trigger AS $$
BEGIN
   IF EXISTS(SELECT level FROM server_ranks WHERE level = NEW.level)
THEN
       RAISE EXCEPTION 'Cannot insert a new global rank with the same
level as an existing server rank';
   END IF;

   RETURN NEW;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER global_ranks_on_insert
BEFORE INSERT ON global_ranks
FOR EACH ROW
EXECUTE FUNCTION global_ranks_on_insert();
```

```
CREATE FUNCTION global_ranks_on_update() RETURNS trigger AS $$
BEGIN
   IF EXISTS(SELECT level FROM server_ranks WHERE level = NEW.level)
THEN
       RAISE EXCEPTION 'Cannot update a global rank with the same
level as an existing server rank';
   END IF;

   RETURN NEW;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER global_ranks_on_update
BEFORE UPDATE ON global_ranks
FOR EACH ROW
WHEN (OLD.level IS DISTINCT FROM NEW.level)
EXECUTE FUNCTION global_ranks_on_update();
```

```
CREATE FUNCTION servers_on_delete() RETURNS trigger AS $$
BEGIN
   RAISE EXCEPTION 'Cannot remove network or website from servers';
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER servers_on_delete
   BEFORE DELETE ON servers
   FOR EACH ROW
   WHEN (OLD.id = -1 OR OLD.id = -2)
   EXECUTE FUNCTION servers_on_delete()
```

```
CREATE FUNCTION servers_on_update() RETURNS trigger AS $$
BEGIN
    RAISE EXCEPTION 'Cannot update the ID of a server';
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER servers_on_update
    BEFORE UPDATE ON servers
    FOR EACH ROW
    WHEN (OLD.id IS DISTINCT FROM NEW.id)
    EXECUTE FUNCTION servers_on_update();
```

```
CREATE FUNCTION ranks_on_delete() RETURNS trigger AS $$
BEGIN
    RAISE EXCEPTION 'Cannot remove the basic or the highest rank from
ranks';
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER ranks_on_delete
    BEFORE DELETE ON ranks
    FOR EACH ROW
    WHEN (OLD.level = 10 OR OLD.level = 1000)
    EXECUTE FUNCTION ranks_on_delete();
```

```
CREATE FUNCTION players_on_update() RETURNS trigger AS $$
BEGIN
    IF OLD.id IS DISTINCT FROM NEW.id THEN
        RAISE EXCEPTION 'Cannot change the ID of a player';
    ELSE
        RAISE EXCEPTION 'Cannot change the rank of the CONSOLE
player';
    END IF;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER players_on_update
    BEFORE UPDATE ON players
    FOR EACH ROW
    WHEN (OLD.id IS DISTINCT FROM NEW.id OR (OLD.id = -1 AND OLD.rank
IS DISTINCT FROM NEW.rank))
    EXECUTE FUNCTION players_on_update();
```

```
CREATE FUNCTION active_ranks_on_update() RETURNS trigger AS $$
BEGIN
    RAISE EXCEPTION 'Cannot change the start of an active rank';
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER active_ranks_on_update
    BEFORE UPDATE ON active_ranks
    FOR EACH ROW
    WHEN (OLD.start IS DISTINCT FROM NEW.start)
    EXECUTE FUNCTION active_ranks_on_update();
```

```
CREATE FUNCTION permissions_on_update() RETURNS trigger AS $$
BEGIN
    RAISE EXCEPTION 'Cannot change the ID, the server or the rank of a
permission';
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER permissions_on_update
    BEFORE UPDATE ON permissions
    FOR EACH ROW
    WHEN (OLD.id IS DISTINCT FROM NEW.id OR OLD.server IS DISTINCT
FROM NEW.server OR OLD.rank IS DISTINCT FROM NEW.rank)
    EXECUTE FUNCTION permissions_on_update();
```

```
CREATE FUNCTION activity_history_on_update() RETURNS trigger AS $$
BEGIN
    RAISE EXCEPTION 'Cannot change activity history';
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER activity_history_on_update
    BEFORE UPDATE ON activity_history
    FOR EACH ROW
    EXECUTE FUNCTION activity_history_on_update();
```

```
CREATE FUNCTION ip_blockades_on_update() RETURNS trigger AS $$
BEGIN
    RAISE EXCEPTION 'Cannot change an ip_blockade except for to_check
value';
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER ip_blockades_on_update
    BEFORE UPDATE ON ip_blockades
    FOR EACH ROW
    WHEN (OLD.id IS DISTINCT FROM NEW.id OR OLD.player IS DISTINCT
FROM NEW.player
        OR OLD.time IS DISTINCT FROM NEW.time OR OLD.ip_address IS
DISTINCT FROM NEW.ip_address)
    EXECUTE FUNCTION ip_blockades_on_update();
```

```
CREATE FUNCTION bans_on_update() RETURNS trigger AS $$
```

```
BEGIN
    RAISE EXCEPTION 'Cannot modify a ban using UPDATE';
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER bans_on_update
    BEFORE UPDATE ON bans
    FOR EACH ROW
    EXECUTE FUNCTION bans_on_update();
```

```
CREATE FUNCTION bans_history_on_update() RETURNS trigger AS $$
BEGIN
    RAISE EXCEPTION 'Cannot modify bans history';
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER bans_history_on_update
    BEFORE UPDATE ON bans_history
    FOR EACH ROW
    EXECUTE FUNCTION bans_history_on_update();
```

```
CREATE FUNCTION ip_bans_on_update() RETURNS trigger AS $$
BEGIN
    RAISE EXCEPTION 'Cannot modify an ip ban';
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER ip_bans_on_update
    BEFORE UPDATE ON ip_bans
    FOR EACH ROW
    EXECUTE FUNCTION ip_bans_on_update();
```

```
CREATE FUNCTION ip_bans_history_on_update() RETURNS trigger AS $$
BEGIN
    RAISE EXCEPTION 'Cannot modify ip bans history';
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER ip_bans_history_on_update
    BEFORE UPDATE ON ip_bans_history
    FOR EACH ROW
    EXECUTE FUNCTION ip_bans_history_on_update();
```

FUNCTIONS

```
CREATE FUNCTION add_global_rank(_level smallint, _colour char(1),
_display_name varchar(75), _chat_format varchar(200))
RETURNS smallint AS $$
DECLARE
   _id smallint;
BEGIN
   INSERT INTO global_ranks
       VALUES (DEFAULT, _level, _colour, _display_name, _chat_format)
       RETURNING id INTO _id;
   RETURN _id;
END
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION update_global_rank(_id smallint, _level smallint,
_display_name varchar(75), _chat_format varchar(200))
RETURNS void AS $$
BEGIN
   UPDATE global_ranks SET
           level = _level,
           display_name = _display_name,
           chat_format = _chat_format
       WHERE id = _id;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION delete_global_rank(_id int)
RETURNS void AS $$
BEGIN
    DELETE FROM global_ranks WHERE id = _id;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION get_global_perms(_rank smallint, _server smallint)
RETURNS table (server smallint, perm varchar(100)) AS $$
DECLARE
   _query text;
BEGIN
   _query = 'SELECT server, permission FROM global_rank_permissions
WHERE rank = $1';

   IF _server IS NOT NULL THEN
       _query = _query || ' AND server = $2';
   END IF;
   _query = _query || ';';

   RETURN QUERY EXECUTE _query
       USING _rank, _server;
END
$$ LANGUAGE plpgsql;
```

```
CREATE TYPE global_rank_perm AS (
   server smallint,
   perm varchar(100)
);
```

```
CREATE FUNCTION update_global_perms(_rank smallint, _perms
global_rank_perm[])
RETURNS void AS $$
BEGIN
   DELETE FROM global_rank_permissions
       WHERE rank = _rank;

   INSERT INTO global_rank_permissions
       SELECT _rank, server, perm FROM UNNEST(_perms);
END
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION get_server_ranks()
RETURNS table (_id smallint, _level smallint, _server smallint,
_display_name varchar(75)) AS $$
   SELECT id, level, server, display_name FROM server_ranks;
$$ LANGUAGE SQL;
```

```
CREATE FUNCTION add_server_rank(_level smallint, _server smallint,
_display_name varchar(75), _chat_format varchar(200))
```

```
RETURNS smallint AS $$
DECLARE
   _id smallint;
BEGIN
   INSERT INTO server_ranks
       VALUES (DEFAULT, _level, _server, _display_name, _chat_format)
       RETURNING id INTO _id;

   RETURN _id;
END
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION update_server_rank(_id smallint, _level smallint,
_server smallint, _display_name varchar(75),
   _chat_format varchar(200))
RETURNS void AS $$
BEGIN
   UPDATE server_ranks SET
       level = _level,
       server = _server,
       display_name = _display_name,
       chat_format = _chat_format
       WHERE id = _id;
END
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION delete_server_rank(_id smallint)
RETURNS void AS $$
BEGIN
   DELETE FROM server_ranks WHERE id = _id;
END
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION get_server_perms(_rank smallint)
RETURNS table (_perm varchar(100)) AS $$
BEGIN
   SELECT permission FROM server_rank_permissions
       WHERE rank = _rank;
END
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION update_server_perms(_rank smallint, _perms
varchar(100)[])
```

```
RETURNS void AS $$
BEGIN
   DELETE FROM server_rank_permissions
       WHERE rank = _rank;

   INSERT INTO server_rank_permissions
       SELECT _rank, perm FROM UNNEST(_perms) AS perm;
END
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION create_player(_nick varchar(16), _password varchar,
_time int, _ip_address varchar(15))
RETURNS int AS $$
DECLARE
   _id int;
BEGIN
   INSERT INTO players VALUES (DEFAULT, _nick, _password, DEFAULT, 10)
RETURNING id INTO _id;
   EXECUTE log_activity(_id, _time, 1, _ip_address);
   RETURN _id;
END
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION set_email(_nick varchar(16), _email varchar(75))
RETURNS void AS $$
   UPDATE players SET email = _email WHERE lower(nick) = lower(_nick);
$$ LANGUAGE SQL;
```

```
CREATE FUNCTION change_password(_nick varchar(16), _new_password
varchar)
RETURNS void AS $$
```

```
   UPDATE players SET password = _new_password WHERE lower(nick) =
lower(_nick);
$$ LANGUAGE SQL;
```

```
CREATE FUNCTION change_rank(_nick varchar(16), _new_rank smallint)
RETURNS void AS $$
  UPDATE players SET rank = _new_rank WHERE lower(nick) =
lower(_nick);
$$ LANGUAGE SQL;
```

```
CREATE FUNCTION get_player_with_password(_nick varchar(16))
    RETURNS table (id int, nick varchar(16), rank smallint, password
varchar) AS $$
    SELECT id, nick, rank, password FROM players WHERE nick = _nick;
$$ LANGUAGE SQL;
```

```
CREATE FUNCTION log_activity(_player int, _time int, _status smallint,
_ip_address varchar(15))
RETURNS void AS $$
 INSERT INTO activity_history VALUES (DEFAULT, _player, _time,
_status, _ip_address)
$$ LANGUAGE SQL;
```

```
CREATE FUNCTION get_player(_id int)
RETURNS table (__id int, __nick varchar(16), __rank smallint) AS $$
BEGIN
    RETURN QUERY SELECT id, nick, rank FROM players WHERE id = _id;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION active_bans_page_query(_page int, _size int, _server
smallint, _recipient int, _giver int,
 _start_from int, _start_to int, _order_by varchar, _asc bool)
RETURNS text AS $$
DECLARE
 _query text;
 _has_where bool = false;
BEGIN
 _query = 'SELECT b.id, b.server, r.id AS recipient_id, r.nick AS
recipient_nick, r.rank AS recipient_rank, b.start ' ||
          'FROM bans AS b ' ||
          'INNER JOIN players AS r ON b.recipient = r.id';

 IF _server IS NOT NULL THEN
```

```
    _query = _query || ' WHERE b.server = ' || _server;
    _has_where = true;
END IF;

IF _recipient IS NOT NULL THEN
    IF _has_where THEN
        _query = _query || ' AND ';
    ELSE
        _query = _query || ' WHERE ';
        _has_where = true;
    END IF;

    _query = _query || 'recipient = ' || _recipient;
END IF;

IF _giver IS NOT NULL THEN
    IF _has_where THEN
        _query = _query || ' AND ';
    ELSE
        _query = _query || ' WHERE ';
        _has_where = true;
    END IF;

    _query = _query || 'giver = ' || _giver;
END IF;

IF _start_from IS NOT NULL THEN
    IF _has_where THEN
        _query = _query || ' AND ';
    ELSE
        _query = _query || ' WHERE ';
        _has_where = true;
    END IF;

    _query = _query || 'start >= ' || _start_from;
END IF;

IF _start_to IS NOT NULL THEN
    IF _has_where THEN
        _query = _query || ' AND ';
    ELSE
        _query = _query || ' WHERE ';
    END IF;

    _query = _query || 'start <= ' || _start_to;
```

```
 END IF;

 _query = _query || ' ORDER BY server DESC, ' || _order_by;
 IF _asc THEN
     _query = _query || ' ASC';
 ELSE
     _query = _query || ' DESC';
 END IF;

 _query = _query || ' LIMIT ' || _size || ' OFFSET ' || (_page * _size);

 RETURN _query;
END;
$$ LANGUAGE plpgsql;



CREATE FUNCTION active_bans_total_query(_server smallint, _recipient int,
_giver int, _start_from int,
  _start_to int)
RETURNS text AS $$
DECLARE
  _query text;
  _has_where bool = false;
BEGIN
  _query = 'SELECT CAST(COUNT(*) AS int) AS id, NULL AS server, NULL AS
recipient, NULL AS start ' ||
           'FROM bans AS b ' ||
           'INNER JOIN players AS r ON b.recipient = r.id';

  IF _server IS NOT NULL THEN
      _query = _query || ' WHERE b.server = ' || _server;
      _has_where = true;
  END IF;

  IF _recipient IS NOT NULL THEN
      IF _has_where THEN
          _query = _query || ' AND ';
      ELSE
          _query = _query || ' WHERE ';
          _has_where = true;
      END IF;

      _query = _query || 'recipient = ' || _recipient;
  END IF;
```

```
  IF _giver IS NOT NULL THEN
      IF _has_where THEN
          _query = _query || ' AND ';
      ELSE
          _query = _query || ' WHERE ';
          _has_where = true;
      END IF;

      _query = _query || 'giver = ' || _giver;
  END IF;

  IF _start_from IS NOT NULL THEN
      IF _has_where THEN
          _query = _query || ' AND ';
      ELSE
          _query = _query || ' WHERE ';
          _has_where = true;
      END IF;

      _query = _query || 'start >= ' || _start_from;
  END IF;

  IF _start_to IS NOT NULL THEN
      IF _has_where THEN
          _query = _query || ' AND ';
      ELSE
          _query = _query || ' WHERE ';
      END IF;

      _query = _query || 'start <= ' || _start_to;
  END IF;

  RETURN _query;
END;
$$ LANGUAGE plpgsql;



CREATE FUNCTION get_active_bans_page(_page int, _size int, _server
smallint, _recipient int, _giver int,
  _start_from int, _start_to int, _order_by varchar, _asc bool)
RETURNS table (id int, server smallint, recipient varchar(16), start int)
AS $$
DECLARE
  _query text;
```

```
BEGIN
  _query = active_bans_total_query(_server, _recipient, _giver,
_start_from, _start_to);
  _query = _query || ' UNION ALL ' || active_bans_page_query(_page, _size,
_server, _recipient,
      _giver, _start_from, _start_to, _order_by, _asc) || ';';

  EXECUTE _query;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE VIEW active_bans_details AS
   SELECT b.id AS banId, b.server, r.id AS recipientId, r.nick AS
recipientNick, r.rank AS recipientRank,
          g.id AS giverId, g.nick AS giverNick, g.rank AS giverRank,
b.reason, b.start, b.expiration
   FROM bans AS b
   INNER JOIN players AS r ON b.recipient = r.id
   INNER JOIN players AS g ON b.giver = g.id;
```

```
CREATE FUNCTION get_active_ban(_id int)
RETURNS table (id int, server smallint, recipientId int, recipientNick
varchar(16), recipientRank smallint,
   giverId int, giverNick varchar(16), giverRank smallint, reason
varchar(300), start int, expiration int) AS $$
BEGIN
  SELECT * FROM active_bans_details WHERE banId = _id;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION modify_ban(_id int, _server smallint, _reason
varchar(300), _length interval, _modification_reason varchar(300), _modder
int)
RETURNS VOID AS $$
DECLARE
   _ban bans%rowtype;
   _expiration timestamp;
   _new_ban int;
BEGIN
   DELETE FROM bans WHERE id = _id RETURNING * INTO _ban;

   IF _server IS NULL THEN
       _server = _ban.server;
   END IF;
```

```
    IF _reason IS NULL THEN
        _reason = _ban.reason;
    END IF;

    IF _length IS NULL THEN
        _expiration = _ban.expiration;
    ELSE
        _expiration = _ban.start + _length;
    END IF;

    INSERT INTO bans VALUES (DEFAULT, _server, _ban.recipient, _ban.giver,
_reason, _ban.start, _expiration) RETURNING id INTO _new_ban;

    INSERT INTO bans_history VALUES (_ban.id, _ban.server, _ban.recipient,
_ban.giver, _ban.reason, _ban.start,
                                     _ban.expiration, NOW()::timestamp, 3,
_modification_reason, _modder, _new_ban);
END;
$$ LANGUAGE plpgsql;

CREATE FUNCTION expire_bans()
RETURNS VOID AS $$
DECLARE
    _now timestamp = NOW();
BEGIN
    WITH expired AS (
        DELETE FROM bans WHERE expiration <= _now
        RETURNING id, server, recipient, giver, reason, start, expiration
    ) INSERT INTO bans_history SELECT id, server, recipient, giver, reason,
start, expiration, expiration, 1, NULL, NULL, NULL FROM expired;
END;
$$ LANGUAGE plpgsql;



CREATE FUNCTION cancel_ban(_id int, _cancel_reason varchar(300), _modder
int)
RETURNS VOID AS $$
BEGIN
  WITH canceled AS (
      DELETE FROM bans WHERE id = _id
      RETURNING server, recipient, giver, reason, start, expiration
  ) INSERT INTO bans_history SELECT _id, server, recipient, giver, reason,
start, expiration, NOW(), 2, _cancel_reason, _modder, NULL FROM canceled;
END;
$$ LANGUAGE plpgsql;



CREATE FUNCTION modify_ban(_id int, _server smallint, _recipient int,
_reason varchar(300), _length interval,
```

```
 _modification_reason varchar(300), _modder int)
RETURNS INT AS $$
DECLARE
 _ban bans%rowtype;
 _expiration timestamp;
 _new_ban int;
BEGIN
 DELETE FROM bans WHERE id = _id RETURNING * INTO _ban;

 IF _server IS NULL THEN
     _server = _ban.server;
 END IF;

 IF _recipient IS NULL THEN
     _recipient = _ban.recipient;
 END IF;

 IF _reason IS NULL THEN
     _reason = _ban.reason;
 END IF;

 IF _length IS NULL THEN
     _expiration = _ban.expiration;
 ELSE
     _expiration = _ban.start + _length;
 END IF;

 INSERT INTO bans VALUES (DEFAULT, _server, _recipient, _ban.giver,
_reason, _ban.start, _expiration) RETURNING id INTO _new_ban;

 INSERT INTO bans_history VALUES (_ban.id, _ban.server, _ban.recipient,
_ban.giver, _ban.reason, _ban.start,
                                  _ban.expiration, NOW(), 3,
_modification_reason, _modder, _new_ban);

 RETURN _new_ban;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION get_ranks_with_perms(_server smallint)
RETURNS table (level smallint, colour char(1), display_name
varchar(100), chat_format varchar(300), permission varchar(100)) AS $$
BEGIN
```

```
   RETURN QUERY
       SELECT r.level, r.colour, r.display_name, r.chat_format,
p.permission
           FROM global_ranks AS r
           LEFT OUTER JOIN global_rank_permissions AS p ON r.level =
p.rank AND p.server = _server
       UNION ALL
       SELECT r.level, r.colour, r.display_name, r.chat_format,
p.permission
           FROM server_ranks AS r
           LEFT OUTER JOIN server_rank_permissions AS p ON r.level =
p.rank
           WHERE r.server = _server;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION players_total_query(_nick varchar(16))
RETURNS text AS $$
BEGIN
 return 'SELECT CAST(COUNT(*) AS int) AS id, NULL AS nick, NULL AS
rank' ||
        ' FROM players' ||
        ' WHERE lower(nick) LIKE ''' || LOWER(_nick) || '%''';
END
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION players_page_query(_page int, _size int, _nick
varchar(16))
RETURNS text AS $$
DECLARE
_query text;
BEGIN
_query = 'SELECT id, nick, rank FROM players';

IF _nick IS NOT NULL THEN
    _query = _query || ' WHERE LOWER(nick) LIKE ''' || LOWER(_nick) ||
'%''';
END IF;

_query = _query || ' ORDER BY nick ASC' ||
         ' LIMIT ' || _size || ' OFFSET ' || (_page * _size);
```

```
RETURN _query;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION get_players_page(_page int, _size int, _nick
varchar(16))
RETURNS table (id int, nick varchar(16), rank smallint) AS $$
DECLARE
    _query text;
BEGIN
    _query = players_total_query(_nick) || ' UNION ALL (' ||
             players_page_query(_page, _size, _nick) || ');';

    RETURN QUERY EXECUTE _query;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION update_global_rank(_id int, _level smallint,
_display_name varchar(75),
    _chat_format varchar(200))
RETURNS void AS $$
BEGIN
    UPDATE global_ranks SET
            level = _level,
            display_name = _display_name,
            chat_format = _chat_format
        WHERE id = _id;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION update_global_rank(_id int, _level smallint,
_display_name varchar(75), _chat_format varchar(200))
RETURNS void AS $$
BEGIN
    UPDATE global_ranks SET
            level = _level,
            display_name = _display_name,
            chat_format = _chat_format
        WHERE id = _id;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION delete_global_rank(_id int)
RETURNS void AS $$
BEGIN
    DELETE FROM global_ranks WHERE id = _id;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION get_global_rank_perms(_rank smallint, _server
smallint)
RETURNS table (_r smallint, _s smallint, _p varchar(100)) AS $$
DECLARE
    _query text;
BEGIN
    _query = 'SELECT rank, server, permission FROM
global_rank_permissions';

    IF _rank IS NOT NULL AND _server IS NOT NULL THEN
        _query = _query || ' WHERE rank = ' || _rank || ' AND server =
' || _server;
    ELSIF _rank IS NOT NULL THEN
        _query = _query || ' WHERE rank = ' || _rank;
    ELSIF _server IS NOT NULL THEN
        _query = _query || ' WHERE server = ' || _server;
    END IF;
    _query = _query || ';';

    RETURN QUERY _query;
END
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION get_global_ranks()
RETURNS table (_id smallint, _level smallint, _display_name
varchar(75), _chat_format varchar(200)) AS $$
BEGIN
    RETURN QUERY SELECT id, level, display_name, chat_format
        FROM global_ranks;
END
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION get_global_ranks()
RETURNS table (_id smallint, _level smallint, _colour char(1),
_display_name varchar(75), _chat_format varchar(200)) AS $$
BEGIN
```

```
    RETURN QUERY SELECT id, level, colour, display_name, chat_format
        FROM global_ranks;
END
$$ LANGUAGE plpgsql;

CREATE FUNCTION get_global_perms(_server smallint)
RETURNS table (_rank smallint, _permission varchar(100)) AS $$
BEGIN
    RETURN QUERY SELECT rank, permission
        FROM global_rank_permissions
        WHERE server = _server;
END
$$ LANGUAGE plpgsql;
```