3 Engenharia Reversa do BOCA

Este capítulo detalha o processo de engenharia reversa conduzido para compreender o funcionamento interno do BOCA. De maneira sucinta, as principais funcionalidades do sistema foram mapeadas através de um diagrama de casos de uso que permitiu identificar que a) os tipos de usuários e b) a estruturação de objetos de aprendizagem no BOCA impõem barreiras para seu uso no contexto de ensino/aprendizagem. A partir daí, foi feita a inspeção do código-fonte e do banco de dados para avaliar a adequação do BOCA. Dito isso, começamos com uma visão geral desse sistema.

O BOCA foi criado como o objetivo de apoiar a realização de competições de programação (CAMPOS, 2004), como por exemplo, a Maratona de Programação da Sociedade Brasileira de Computação[1] e o Topcom[2]. Esse sistema pode ser instalado juntamente com o Maratona Linux, uma distribuição Linux que visa facilitar o processo de instalação (MORAIS; RIBAS, 2019).

Conforme ilustrado na Figura 4, o sistema é baseado em uma arquitetura monolítica composta de 3 módulos principais: aplicação Web (BOCA); banco de dados; e módulo de correção automática (JAIL). Tanto o BOCA quanto o JAIL foram desenvolvidos na linguagem de programação PHP[3], enquanto o SGBD (Sistema de Gerenciamento de Banco de Dados) utilizado é o PostgreSQL[4].

Em linhas gerais, instalação envolve 4 passos principais, sendo eles: 1) extração do arquivo .zip; 2) criação do banco de dados; 3) edição de arquivos de configuração; e 4) execução do *script* de inicialização do sistema. A instalação do JAIL consiste em executar um *script* que instala uma distribuição Linux específica para criar um ambiente seguro e protegido de compilação, execução e teste das submissões (programas).

Segundo as instruções de instalação[5], é necessário o cumprimento de alguns pré-requisitos, como por exemplo, PostgreSQL 8.2+/9.1+, Apache 2.2+ e PHP 5.3+. Além disso, uma série de pacotes devem ser instalados utilizando o gerenciador de pacotes APT (*Advanced Package Tool* ou, simplificando, *apt-get*) para que o sistema funcione corretamente e é recomendado que o sistema seja instalado em um computador dedicado (não utilizado por outros usuários).

Uma vez instalado, o usuário com perfil de administrador (*Admin*) pode acessar a aplicação Web para criar competições, especificar as linguagens de programação, cadas-

[1] http://maratona.sbc.org.br/
[2] https://topcom.pet.inf.ufes.br/
[3] https://www.php.net/
[4] https://www.postgresql.org/
[5] https://github.com/cassiopc/boca

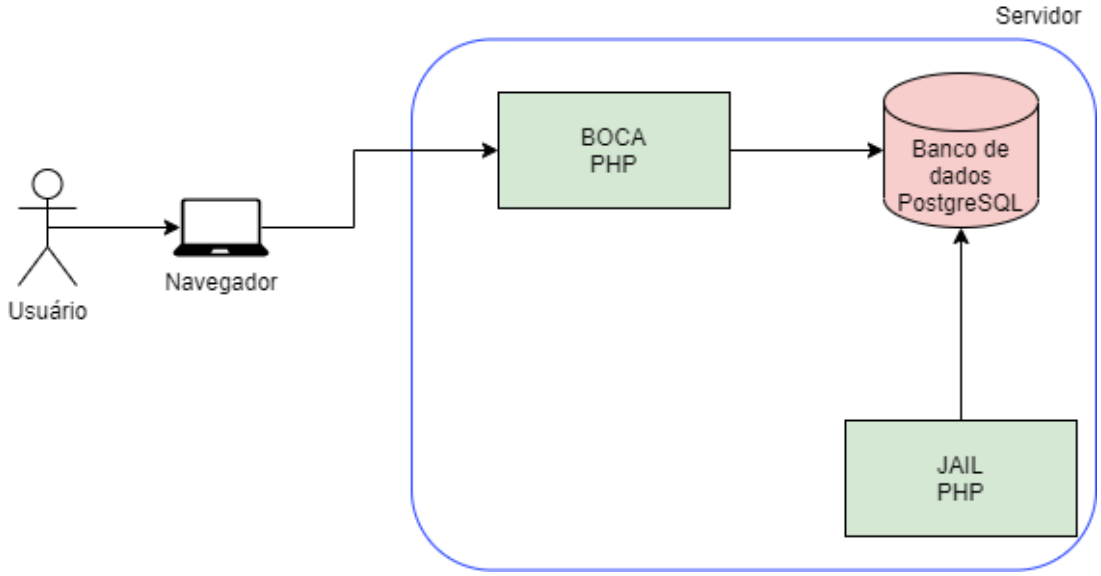

Figura 4 – Arquitetura originalmente monolítica do BOCA composta de aplicação Web (BOCA), banco de dados e módulo de correção automática (JAIL).

trar problemas (*Problems*) e equipes, além de visualizar as submissões feitas, para citar algumas. Como exemplo, para iniciar uma competição o administrador deve começar pela especificação do nome, de uma data de início e da duração da competição (vide Figura 5).

Além do usuário administrador, o BOCA permite ainda a criação de outros usuários, como por exemplo, os do tipo *Staff*, responsáveis por ajudar em tarefas relacionadas à organização da competição, equipe ou time (*Team*), que englobam os participantes do evento, e juiz (*Judge*), responsáveis por corrigir submissões (chamadas de *Runs* no BOCA)[6].

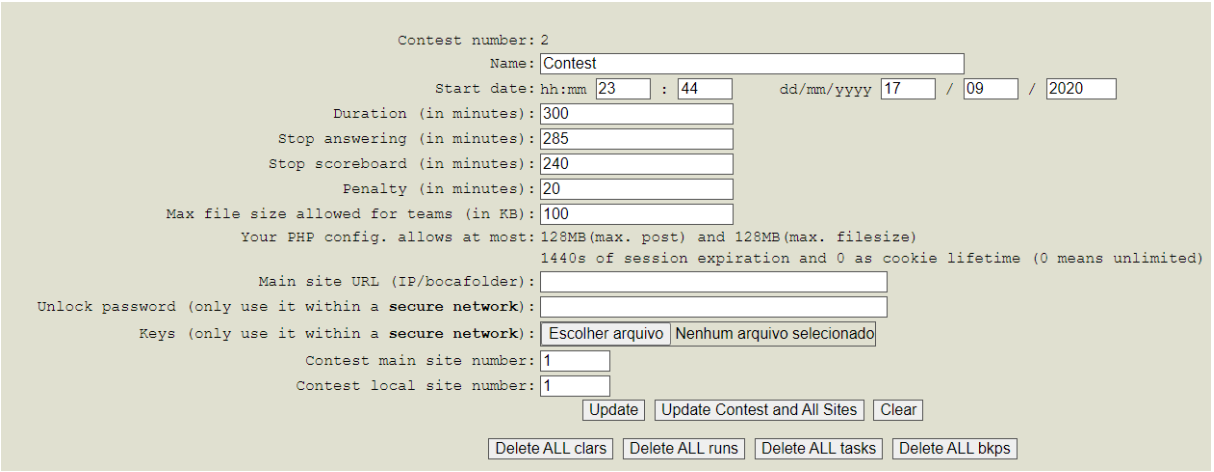

Figura 5 – Interface de criação de competição no BOCA.

Já o banco de dados PostgreSQL é utilizado para fazer a persistência tanto dos dados quanto dos arquivos de problemas/desafios (além da sua descrição, cada problema no

[6] Existem ainda os perfis de usuário *site* e placar (*Score*), responsáveis por gerenciar competições em várias localidades e o placar, respectivamente.

BOCA é composto por uma coleção de arquivos de configuração, todos esses compactados em um único arquivo que é armazenado diretamente no banco).

Além do suporte à correção manual de submissões por usuários com perfil de juiz, quando habilitado, o JAIL se encarrega de compilar, executar e verificar a resposta gerada com base nas instruções definidas nos arquivos de configuração do problema.

3.1 Levantamento das Funcionalidades

Para conhecer melhor as funcionalidades oferecidas pelo sistema BOCA, foi necessário utilizá-lo de maneira exploratória. Complementarmente, essa análise se baseou em relatos informais de um professor que utiliza a ferramenta em suas disciplinas. Utilizando os conhecimentos obtidos nas disciplinas de Engenharia de Software, como primeiro produto desse processo criou-se o diagrama de casos de uso apresentado na Figura 6. A partir desse diagrama de casos de uso podemos observar que, apesar de nos últimos anos o uso do BOCA ter sido considerado no apoio ao ensino/aprendizagem em disciplinas de programação (SOUZA, 2014; MOREIRA, 2014), não há um mapeamento trivial dos casos de uso para tal contexto.

O primeiro obstáculo para o uso do BOCA no contexto de ensino/aprendizagem está relacionado aos **tipos de usuários** disponíveis no sistema. Como exemplo, considere a figura do professor de uma disciplina de programação. Normalmente, tal ator contempla tanto os casos de Administrador quanto os de Juiz, visto que é normalmente o professor é responsável por gerenciar os objetos de aprendizagem (nesse caso, problemas) e fazer a correção das submissões, respectivamente. Além disso, se em um primeiro momento podemos dizer que o aluno realiza casos de uso similares aos de time, é importante ressaltar que semanticamente o primeiro designa uma única pessoa, enquanto o segundo define um grupo de competidores. Cabe ainda mencionar que os papéis *Staff* e *Score* podem ser considerados desnecessários no contexto de ensino aprendizagem[7].

Outra adversidade da aplicação da ferramenta diz respeito à **estruturação de objetos de aprendizagem**. Normalmente, um professor não organiza competições (muito menos incluindo múltiplas localidades ou *sites*), mas costuma disponibilizar exercícios relativos a um dado tópico do conteúdo utilizando listas de exercícios. Apesar do sistema não limitar a inserção de problemas, o BOCA não foi projetado para permitir a organização desses por tópico ou rótulos[8]. De acordo com relatos informais tanto de professores quanto de alunos que já utilizaram essa ferramenta em disciplinas de programação na Ufes, a

[7] Dentre as tarefas do usuário *Staff* no Topcom estão a impressão do código-fonte ou a entrega de balões inflados quando as equipes acertas questões. Como se pode observar, essas tarefas são incompatíveis com o ensino de disciplinas de programação.

[8] Competições de programação competitiva costumam conter um número baixo de desafios (por volta de uma dezena), não sendo tão relevante uma estruturação multinível.

Figura 6 – Diagrama de casos de uso criado a partir da utilização do BOCA.

medida que aumenta a disponibilidade de objetos de aprendizagem, maiores tendem a ser os contratempos para organizar e acessar esse conteúdo.

3.2 Inspeção do Código-Fonte e do Banco de Dados

O passo seguinte consistiu na inspeção do código-fonte com o objetivo de compreender o funcionamento interno e a organização dos componentes de *software* para estimar o esforço de alteração frente os requisitos discutidos acima. Nesse processo foram necessários vários testes de criação de *Problems* e a submissão de *Runs* para descobrir não só o funcionamento do BOCA, banco de dados e do JAIL, como também como acontece a comunicação entre eles. Um resultado importante foi a constatação de que o BOCA (aplicação Web) e o JAIL utilizam o banco de dados como uma fila de *Runs* (submissões) a serem corrigidos (Figura 7).

Nessa configuração, a aplicação Web (BOCA) é responsável por inserir as submissões

enviadas pela população de usuários do tipo *Team* no final da fila (1), enquanto o módulo de correção automática JAIL se encarrega de retirá-las para atendimento (2). Tecnicamente, o JAIL é capaz de atender (corrigir) um problema por vez e quando ocioso, fica em um laço indefinido (*loop* infinito) verificando a cada 10s a chegada de novas submissões (ou seja, novos registros no banco de dados). Então, quando em umas dessas consultas se verifica a existência de submissões não corrigidas, a mais antiga é retirada da fila para receber atendimento[9] (3). Uma vez finalizado o atendimento, a submissão em questão recebe o status de "corrigida" no banco de dados (4), sendo possível o usuário verificar essa alteração na próxima vez que atualizar a interface gráfica da aplicação no seu dispositivo.

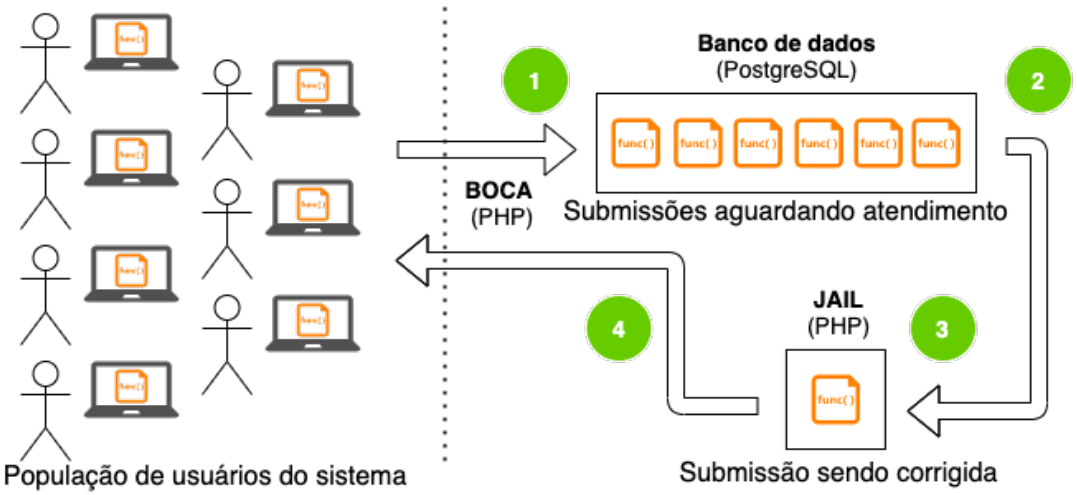

Figura 7 – Funcionamento da fila de submissões do BOCA.

Lançando mão da ferramenta Adminer[10] foi possível acessar o banco de dados e exportar o arquivo SQL (*Structured Query Language*) de criação do seu esquema. Em seguida, ao importar esse arquivo na ferramenta *online* Draw.io[11], foi gerado o esquema conceitual apresentado na Figura 22 (consulte o Apêndice A para a versão completa com os atributos/colunas). Nela destacamos as seguintes tabelas/entidades do sistema:

- `contesttable`: tabela usada para armazenar os dados de competições (*Contests*);
- `usertable`: tabela usada para armazenar os dados de usuários (incluindo o seu tipo);
- `problemtable`, tabela na qual são armazenados os dados de problemas (*Problems*) ou exercícios de programação; e
- `runtable`: tabela na qual são armazenados dados de submissões (*Runs*).

[9] Na realidade, a "retirada" da fila é feita de forma lógica através da mudança do estado de uma coluna da tabela *runtable* no banco de dados

[10] https://www.adminer.org/

[11] http://draw.io

As demais tabelas, usadas para armazenar dados de competições multilocal (`sitetable` e `sitetimetable`), de linguagens de programação (`langtable`), de clarificações (`clartable`), de tarefas (`tasktable`), de correção das submissões (`answertable`), de cópia de segurança (`bcktable`), e de registros de eventos no sistema (`logtable`), apesar de necessárias para o funcionamento do BOCA, foram omitidas dessa discussão por se destinarem a outros aspectos que vão além do escopo deste projeto de graduação.

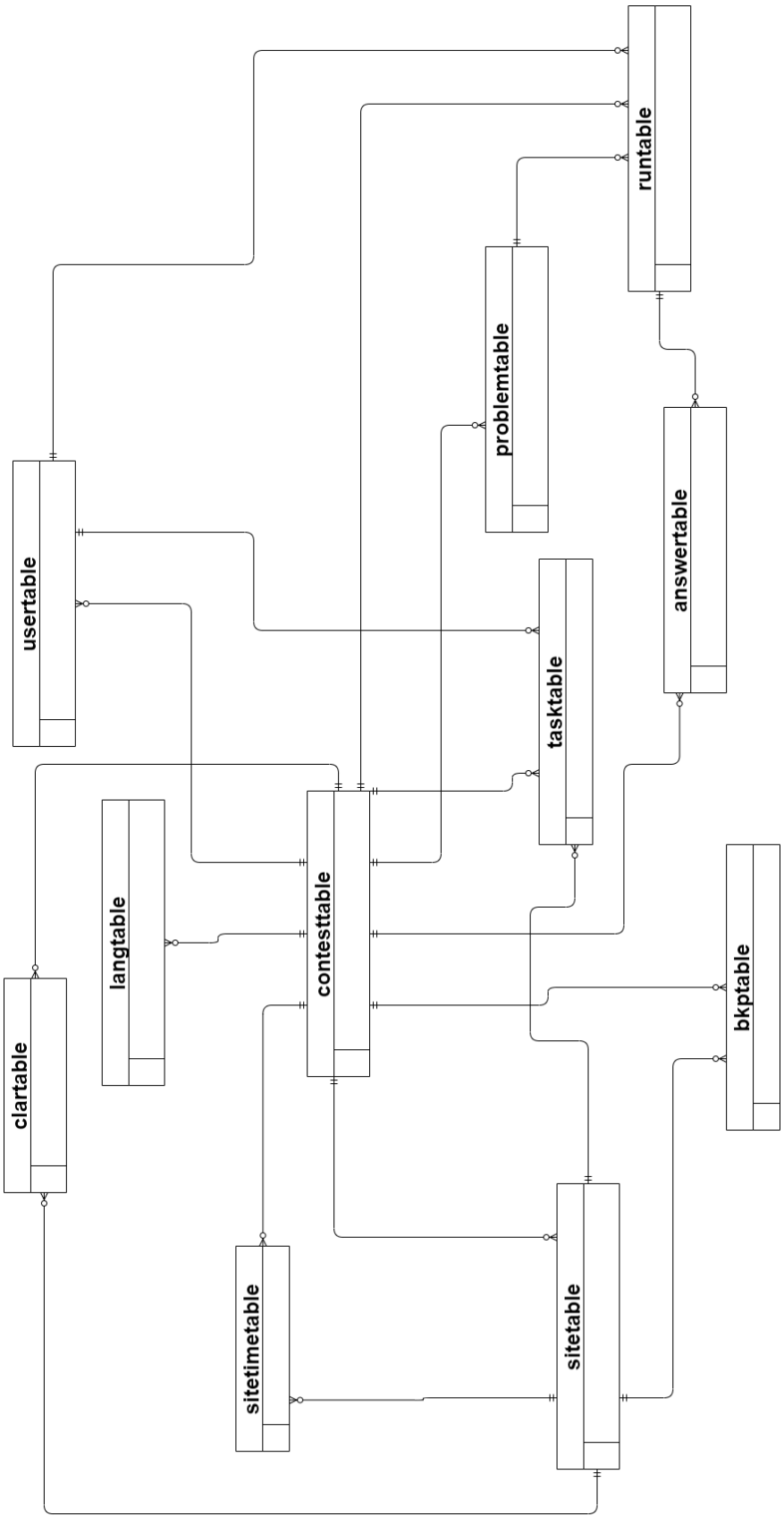

Figura 8 – Esquema conceitual simplificado do banco de dados do BOCA.

4 AlgoChallenger: Projeto e Implementação

Definidos os principais requisitos para adequação do BOCA ao ensino/aprendizagem de programação (suporte aos usuários do tipo professor e aluno e suporte à organização de exercícios em grupos, conforme visto no Capítulo 3), o passo seguinte foi considerar a refatoração de parte do seu código-fonte. Visando facilitar esse processo, consideramos a virtualização do sistema utilizando conteinerização, de modo que os módulos principais ficassem hospedados em 3 contêineres distintos, cada qual com seu (micro)serviço (vide Figura 9). Por mais que a instalação do BOCA seja relativamente fácil, ainda é necessária a formatação do disco rígido para instalação da distribuição do Maratona Linux no servidor, e a utilização de contêineres resolve esse problema.

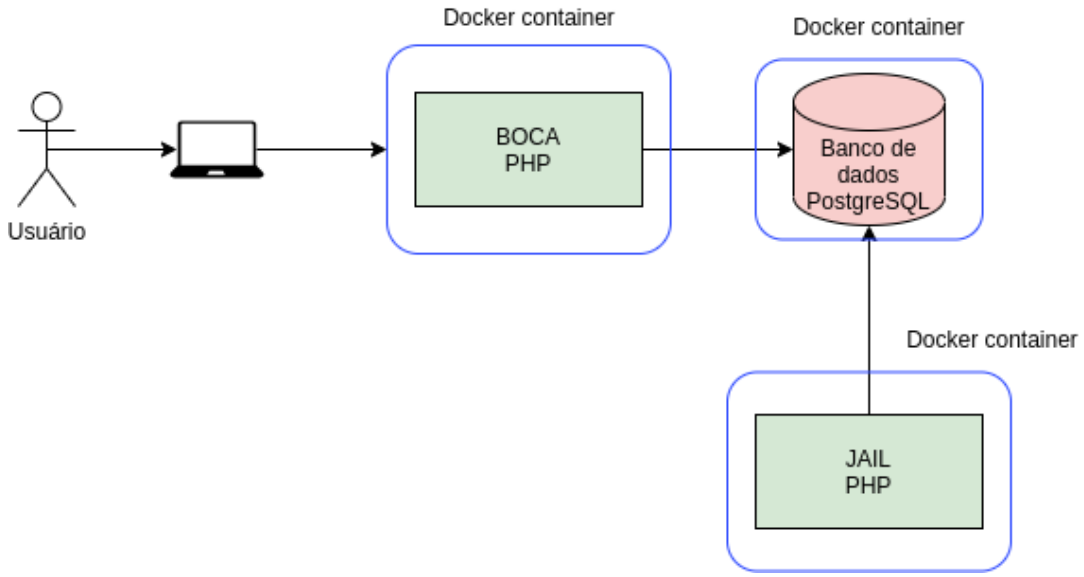

Figura 9 – Reengenharia da arquitetura do BOCA utilizando conteinerização (Docker).

A escolha pela utilização da plataforma Docker se baseou na sua popularidade, disponibilidade de ferramentas e documentação. A partir daí, foi necessário estudar os arquivos instalação do BOCA e especificar sua arquitetura baseada em contêineres através de um arquivo Docker Compose (.yml). Nessa implementação, decidiu-se por criar uma imagem de contêiner (especificada através de um Dockerfile) com a instalação comum ao módulo BOCA (aplicação Web) e ao JAIL pelo fato de diminuir o tempo de implantação do sistema como um todo.

Porém, logo após essa alteração percebeu-se a dificuldade que seria modificar e manter o código devido à sua baixa legibilidade e estruturação (vale destacar que a maior parte do código foi desenvolvida há mais de uma década e tem sido pontualmente atualizado

[1] Observação baseada no código disponível em https://github.com/cassiopc/boca

APÊNDICE A – Diagramas

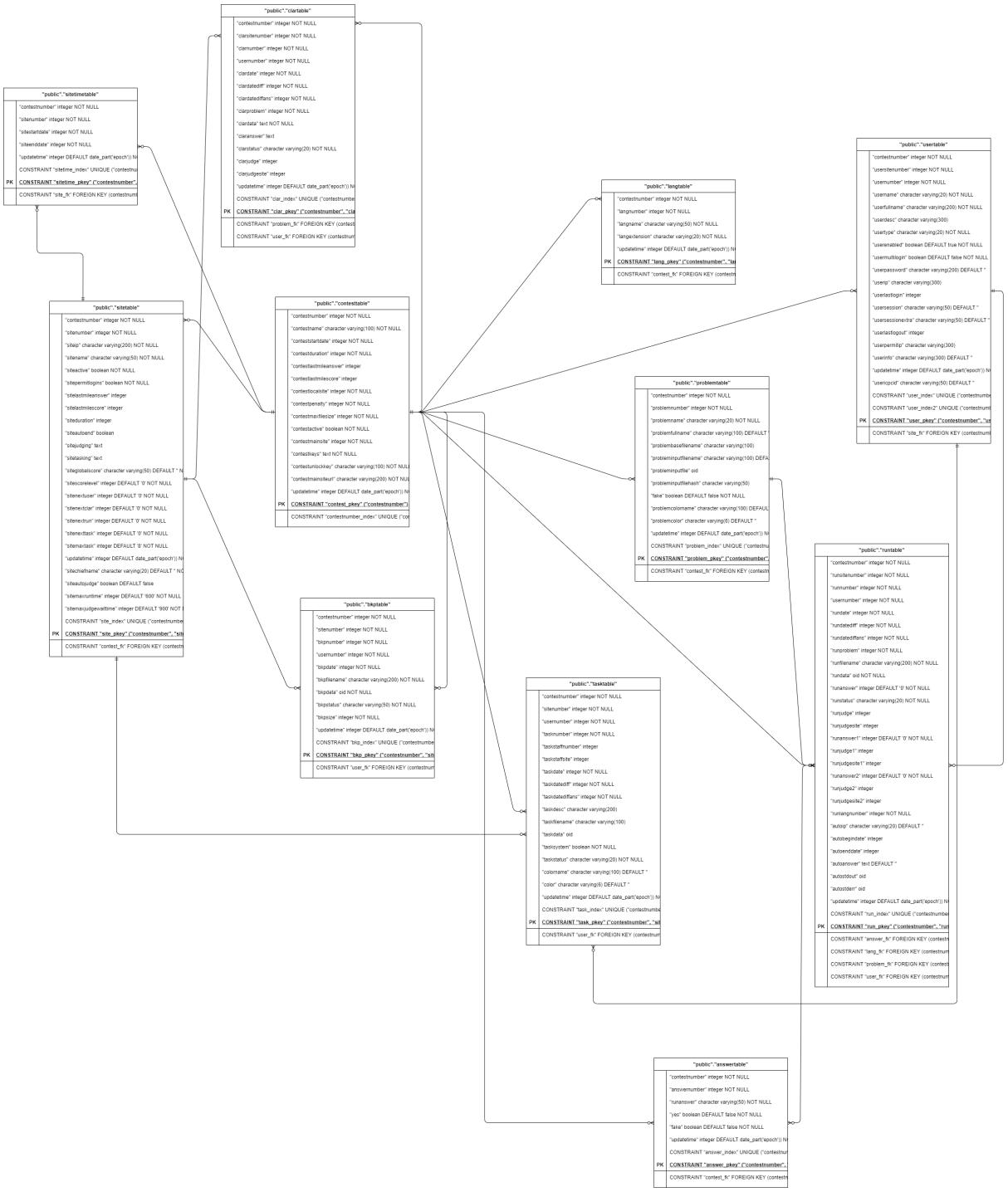

Figura 22 – Esquema conceitual do banco de dados do BOCA.