Avaliação Backend

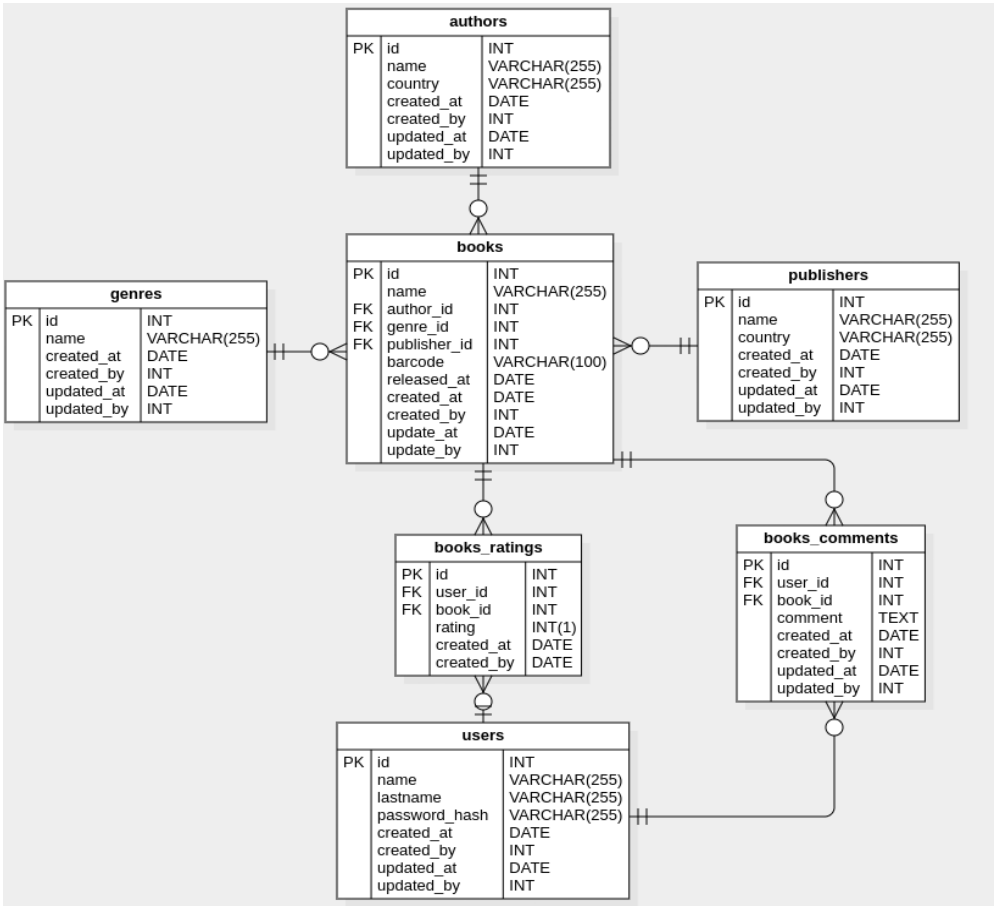
Questão 1

Implemente uma simples biblioteca de Publish/Subscribe em puro Go, capaz de realizar roteamento básico de mensagens com base nos nomes das queues, assim como, capaz de registrar todas as mensagens publicadas em ordem por queue em um sistema de arquivos. Demais requisitos são:

- Testes são obrigatórios e a cobertura deve ser superior a 80%.
- O uso de quaisquer bibliotecas externas é proibido, com exceção do Make para automatização da execução dos testes, caso deseje.
- Pontos extras se as biblioteca utilizar *generics* para suportar os tipos de mensagens.

Questão 2

Para as próximas perguntas, por favor considere o modelo relacional abaixo. Leve em conta que não existem indíces criados nas tabelas.



Questão 2.1

Descreva todos os índices que você acredita serem necessários para os cenários de busca mais comuns deste schema, listando os pontos positivos e negativos dos índices que acredite necessitar de tal análise. Se possível, descreva aspectos como cardinalidade e expectativa de redução dos conjuntos de dados quando estes índices forem usados.

Questão 2.2

Considere a query abaixo:

```
SELECT AVG(rating)
FROM books_ratings
WHERE book_id = ?
```

Levando em conta que a tabela `books_ratings` contém milhões de linhas e que a query acima é representativa do tipo de query feita aos ratings dos livros, liste quais medidas de otimização podem ser tomadas para reduzir o subset e agilizar o cálculo de agregação.

Questão 2.3

Considere a query abaixo:

```
SELECT br.book_id as book
  FROM books_ratings br, books b, genres g
 WHERE br.book_id = b.id
   AND b.genre_id = g.id
   AND g.id IN (?, ?, ?, ?, ?)
 GROUP BY br.book_id
 ORDER BY AVG(br.rating) DESC
LIMIT 50;
```

Levando em conta que todas as tabelas envolvidas contém milhões de linhas e que esta agregação é utilizada em todos os acessos à página inicial da aplicação, liste as medidas de otimização do banco de dados que podem ser tomadas, com seus prós e contras.

Questão 3

Considere o seguinte estudo de caso:

Uma plataforma de notícias, chamada absurd-news.net, provê aos seus usuários notícias sobre os mais variados assuntos 24 horas por dia. Para acessar de forma ilimitada a plataforma, o usuário necessita se cadastrar na plataforma e pagar um valor mensal não especificado. Os benefícios de assinar ao site são inúmeros, dentre os quais a possibilidade de acessar todo o seu histórico de notícias visualizadas nos últimos 6 meses. A plataforma recebe milhões de acessos por dia, considerando que é um dos sites mais acessados de seu continente de origem.

No entanto, a funcionalidade de visualização do histórico de notícias se tornou incrivelmente não responsiva nos últimos meses. Essa funcionalidade é provida por uma tabela relacional simples, chamada `history`, que contém os metadados da notícia, como o identificador único da notícia, assim como, o identificador único do usuário. A tabela contém uma chave única `(user_id, news_id)` que é indexada. A tabela contém mais de 200 milhões de registros, e mesmo uma busca simples de um histórico de um usuário leva mais de 1s para completar.

Com este cenário, o número de usuários da plataforma começou a cair, considerando que a funcionalidade de histórico, muito utilizada por mais de 90% dos usuários, não é responsiva em termos de latência.

Com este cenário em mente, a plataforma deseja que 2 requisitos não funcionais sejam atingidos:

- A funcionalidade de histórico deve funcionar mesmo que o banco de dados esteja completamente offline.
- O tempo de latência para qualquer requisição de busca de histórico deve ser inferior a 20ms em 95% dos casos.

Sugira as modificações necessárias para atingir estes objetivos.

Questão 4

Considere o seguinte estudo de caso:

Uma empresa, chamada X Company, produz uma solução para geração e envio de boletos de empresas prestadoras de serviços aos seus clientes finais. A aplicação funciona de maneira simples, a empresa se registra na plataforma, cadastra seus clientes, com suas respectivas informações (CPF, RG, nome, etc), e detalha o valor e periodicidade das cobranças. O sistema gera os boletos dentro do período de cobrança e 10 dias antes do vencimento do boleto, este será enviado ao cliente pagador dos serviços para pagamento. Dado a natureza da aplicação, o usuário final não precisa se cadastrar para ter acesso aos boletos, ele acessará o documento por meio de um link enviado no e-mail citado anteriormente. Esse link corresponde ao seguinte endpoint:

```
GET https://boletos.x-company.com/boletos/{id}
```

Desta forma, o cliente abre o link anexado ao email e acessa o boleto para pagamento, este contendo algumas informações sensíveis do usuário, como CPF e endereço.

Portanto, com base no cenário e aplicação proposta, identifique a existência (ou não) de vulnerabilidades de segurança e como, caso existam, estas podem ser usadas para gerar danos. Para esta análise, considere algumas requisições logadas por meio do monitoramento de rede da empresa.

```
GET https://boletos.x-company.com/boletos/13857
GET https://boletos.x-company.com/boletos/913851365
GET https://boletos.x-company.com/boletos/1359861
```

Questão 5

Considere o seguinte bloco de código:

```
// cacheValue is a simple wrapper of the cache data to support internal TTL
implementation.
type cacheValue struct {
	data []byte
	ttl  time.Time
}

// newCacheValue is the default constructor for the cacheValue struct.
func newCacheValue(data []byte, ttl time.Time) *cacheValue {
	return &cacheValue{data, ttl}
```

```
}

// Repository provides key value storage implementation on memory.
type Repository[E cache.Cacheable] struct {
	options cache.DriverOptions
	dic     map[string]*cacheValue
}

// NewRepository is the default constructor for the Repository[E] struct.
func NewRepository[E cache.Cacheable](options DriverOptions) Repository[E] {
	return Repository[E]{options, make(map[string]*cacheValue, 0), sync.RWMutex{}}
}

// Encode provides serialization and compression for any write cache operation.
func (r *Repository[E]) Encode(entity E,
	compression Compression,
	serialization Serialization[E]) ([]byte, error) {
	s, err := serialization.Serialize(entity)
	if err != nil {
		return nil, err
	}
	return compression.Encode(s)
}

// Decode provides deserialization and decompression for any read cache operation.
func (r *Repository[E]) Decode(chunk []byte,
	compression Compression,
	serialization Serialization[E]) (*E, error) {
	b, err := compression.Decode(chunk)
	if err != nil {
		return nil, err
	}

	return serialization.Deserialize(b)
}

// StandardizeKey standardizes the key format.
func (r *Repository[E]) StandardizeKey(key string) string {
	switch r.options.GetEnvironment() {
	case repo.EnvSandbox:
		return strings.ToLower(fmt.Sprintf("%s-%s", repo.EnvSandbox, key))
	case repo.EnvProduction:
		return strings.ToLower(fmt.Sprintf("%s-%s", repo.EnvProduction, key))
	default:
		return strings.ToLower(fmt.Sprintf("%s-%s", repo.EnvDevelopment, key))
	}
}

// Set inserts a key/value into the storage.
func (r *Repository[E]) Set(ctx context.Context,
	key string,
	value E,
	compression Compression,
	serialization Serialization[E],
	ttl time.Duration) error {
```

```
	key = r.StandardizeKey(key)
	b, err := r.Encode(value, compression, serialization)
	if err != nil {
		return err
	}

	r.dic[key] = newCacheValue(b, time.Now().Add(ttl))
	return nil
}

// Get retrieves a value from the storage via key.
func (r *Repository[E]) Get(ctx context.Context,
	key string,
	compression Compression,
	serialization Serialization[E]) (*E, error) {

	key = r.StandardizeKey(key)
	value := r.dic[key]
	if value == nil {
		return nil, nil
	}
	if time.Now().After(value.ttl) {
		return nil, nil
	}

	entity, err := r.Decode(value.data, compression, serialization)
	if err != nil {
		if _, err := r.Delete(ctx, key); err != nil {
			return nil, err
		}
	}

	return entity, nil
}

// Delete removes a key from the storage.
func (r *Repository[E]) Delete(ctx context.Context, key string) (bool, error) {
	key = r.StandardizeKey(key)

	if r.dic[key] == nil {
		return false, nil
	}

	delete(r.dic, key)
	return true, nil

}
```

Com base neste código, descartando as interfaces não declaradas no bloco, aponte os problemas contidos na implementação.

Questão 6

Considere o seguinte estudo de caso:

Atualmente você trabalha em uma empresa multinacional de serviços de telecomunicação, chamada ConnectMore. A empresa está planejando a mudança na modalidade de trabalho de seus colaboradores para *work from home*. No entanto, a empresa marca o tempo de trabalho de seus colaboradores de maneira física, com uma máquina de ponto com identificação biométrica.

Considerando que o novo modelo de trabalho necessita de uma solução para ponto digital, a empresa lhe convoca para a criação desta plataforma. Os requisitos funcionais e não funcionais principais são:

- A plataforma terá três funcionalidades, a marcação do ponto, a consulta histórica das marcações e uma área restrita para gestores moderarem modificações requisitadas por seus liderados.
- Todos os usuários serão autenticados.
- A aplicação não pode sob nenhuma condição perder marcações dos colaboradores, já que isto acarretará em punições legais para a empresa.
- Considerando o grande número de usuários e a natureza periódica do uso da aplicação, o sistema deve ser capaz de suportar grandes surtos de demanda.
- O usuário deve sempre ser capaz de marcar seu ponto e visualizar suas próprias marcações do dia em menos de 200ms.
- O sistema deve ser capaz de funcionar a nível de marcação do ponto mesmo que o banco de dados esteja fora do ar.
- É aceitável uma demora relativa entre uma marcação de ponto ser feita e aparecer na consulta histórica para os gestores.

Com base nestes requisitos, desenhe um diagrama da arquitetura básica da aplicação contendo os componentes e suas relações. Não é necessário desenhar os endpoints ou entrar em detalhes relativos ao código da aplicação, ainda que citar as tecnologias/linguagens a serem utilizadas seja desejável.

Questão 7

Considere o bloco de código abaixo:

```
// transcode handlers the file transcoding API.
func (v VideoHandler) transcode(w http.ResponseWriter, r *http.Request) {
	ctx := r.Context()

	filename := r.URL.Query().Get("filename")
	if key == "" {
		w.WriteHeader(http.StatusBadRequest)
		return
	}

	err := os.Setenv("FILE_TO_TRANSCODE", filename)
	if err != nil {
		w.WriteHeader(http.StatusBadRequest)
		return
	}

	err := v.ffmpeg.Transcode() // calls ffmpeg natively using the os env
FILE_TO_TRANSCODE as filename to be used.
	if err != nil {
		w.WriteHeader(http.StatusBadRequest)
		return
	}

	w.WriteHeader(http.StatusNoContent)

}
```

Descreva o funcionamento deste método e identifique os bugs contidos no bloco.

Questão 8

Crie um diagrama entidade-relacionamento seguindo as premissas abaixo:

- O schema deve conter ao menos 7 tabelas.
- O schema deve conter todos os tipos de relacionamentos possíveis.
- O diagrama deve seguir todas as normas definidas no modelo relacional.
- A escolha do domínio usado como referência para as tabelas fica a seu critério, mas descreva uma apresentação resumida sobre ele.

Questão 9

Considere o seguinte estudo de caso:

Um portal sobre lore de livros, chamado wiki-lore-books.com, provê a seus usuários a possibilidade de acessar milhões de explicações detalhadas sobre histórias de livros em vários idiomas, junto com colaborações dos artistas com o contexto e significado de trechos de suas autorias. Qualquer usuário pode adicionar conteúdos sobre um livro, no entanto, antes destes irem ao ar, passam por um processo de moderação. A autenticação dos moderadores é feita por JWT (JSON Web Tokens), que são armazenados dentro de um cookie no navegador da sessão restrita da plataforma. O endpoint para contribuição pode ser visualizado abaixo:

```
POST https://wiki-lore-books.com/books/127858729/colab
{
    "title": "The effect of the spice on Paul Atreides",
    "contribution": "Paul is sensitive to the spice, as the substance awakens his
powers"
}
```

O conteúdo integral da colaboração do usuário é enviado para o moderador por meio deste endpoint. Existem limitações apenas no tamanho dos campos do endpoint, que é de 4096 caracteres para ambos os campos, e que estes devem ser obrigatoriamente preenchidos.

Dado estas informações, identifique a existência (ou não) de vulnerabilidades no processo, e caso existam, de que forma podem ser utilizadas para causar dano à plataforma.

Questão 10

Considere o bloco de código exposto a seguir:

```
// Find retrieves an entity from the database via its identifier.
func (r Repository[E]) Find(ctx context.Context,
      id string,
      table string,
      transaction Transaction[sqlx.Tx, sqlx.DB]) (*E, error) {
      var row *sqlx.Row
      var entity E

      fields := strings.Join(r.GetFields(entity), ",")
```

```
	statement := fmt.Sprintf("SELECT %s FROM %s WHERE id = %s", fields, table, id)

	if transaction != nil {
		row = transaction.GetDriver().QueryRowxContext(ctx, statement)
	} else {
		conn := r.manager.FindByType(sql.ConnectionTypeRead)
		row = conn.GetDriver().QueryRowxContext(ctx, statement)
	}

	if err := row.StructScan(&entity); err != nil {
		if err == stdsql.ErrNoRows {
			return nil, nil
		}
		return nil, err
	}

	return &entity, nil

}
```

Descreva o funcionamento deste método e identifique os bugs contidos no bloco.