

Aula 1**CONCEITO DE FRAMEWORK HIBERNATE**

Você já deve conhecer a área de orientação a objeto e todos os benefícios que ela traz para o dia a dia do desenvolvimento de software. Mas, e se tudo que você precisasse gravar no banco de dados já viesse diretamente das classes que você escreveu no próprio código?

42 minutos

INTRODUÇÃO

Você já deve conhecer a área de orientação a objeto e todos os benefícios que ela traz para o dia a dia do desenvolvimento de software. Mas, e se tudo que você precisasse gravar no banco de dados já viesse diretamente das classes que você escreveu no próprio código? Nesse sentido, abordaremos o framework de persistência de dados Hibernate ORM e como essa poderosa ferramenta poderá se tornar uma aliada na criação de aplicações mais robustas e confiáveis.

O Hibernate almeja tirar do desenvolvedor a responsabilidade pela escrita e validação das consultas que serão feitas no banco de dados. Criado para ser uma ponte entre a aplicação e o banco, o Hibernate fará as abstrações necessárias nessa parte, liberando o desenvolvedor para pensar apenas em suas aplicações.

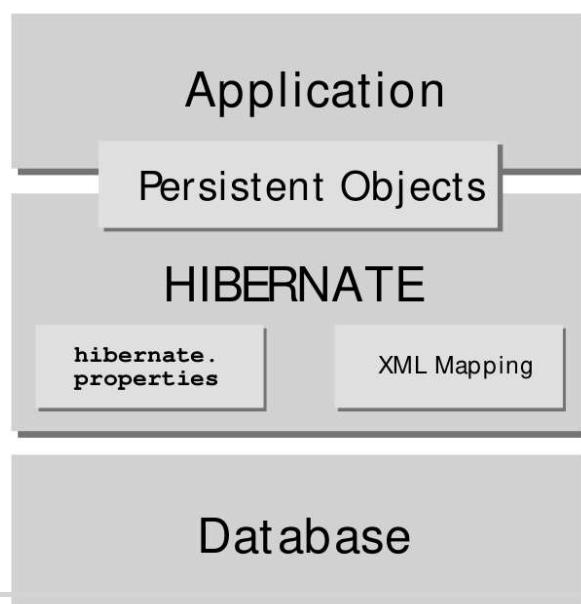
ARQUITETURA GERAL DO HIBERNATE: COMPONENTES FUNÇÕES E FORMAS DE INTEGRAÇÃO**O que é o Hibernate**

O Hibernate é um robusto framework de persistência de dados *open source* que nos ajuda a mapear objetos do código e a transformá-los em um banco relacional clássico. Suas abstrações e potenciais de reuso facilitam a vida do desenvolvedor, que não precisará mais criar as consultas diretamente em *Java Database Connectivity (JDBC)* toda vez. Ele age entre a aplicação e o banco de dados, gerenciando conexões, tabelas do banco e a forma como as consultas são feitas em *Hibernate Query Language (HQL)*.

Ele realiza todas as principais ações com dados, como INSERT, UPDATE, DELETE, etc, e apenas não realiza mudanças estruturais no banco, o que fugiria um pouco de sua própria natureza, que é transformar em banco objetos que foram criados no código. Para encontrar exemplos da sintaxe do HQL, busque, dentro da pasta de instalação do Hibernate, uma pasta chamada *grammar*.

Arquitetura do Hibernate

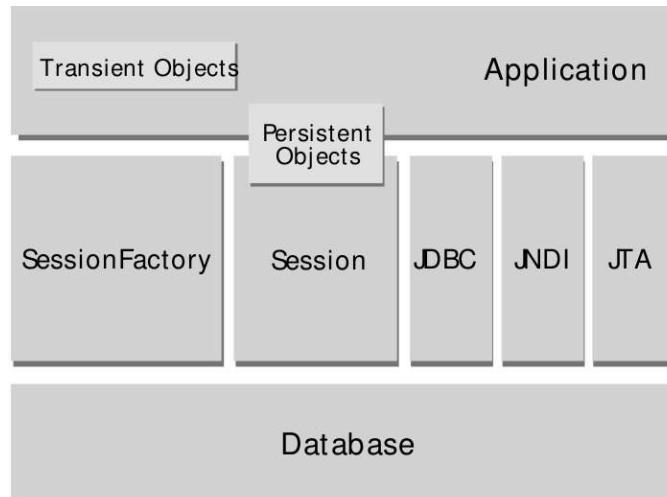
Figura 1 | Visão de alto nível da arquitetura do Hibernate



Fonte: KING *et al.* (2008, p. 29).

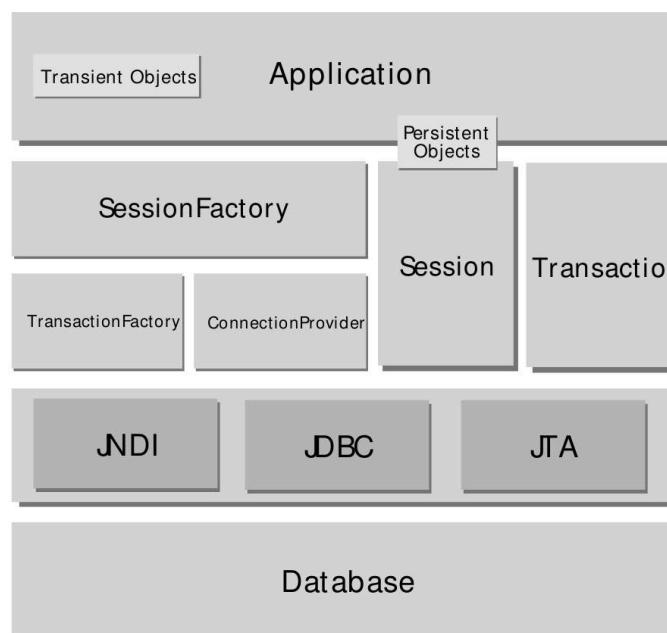
O Hibernate tem uma arquitetura muito flexível e sua documentação oficial apresenta duas abordagens: a mínima e a comprehensiva. Na primeira (Figura 2), as APIs (Application Programming Interface) do Hibernate não são utilizadas, pois o próprio aplicativo gerencia as conexões JDBC e as transações em geral:

Figura 2 | Abordagem mínima

Fonte: KING *et al.* (2008, p. 30).

Já na segunda (Figura 3), deixa-se o framework a cargo do gerenciamento de transações e conexões com o banco:

Figura 3 | Arquitetura comprehensiva

Fonte: KING *et al.* (2008, p. 30).

Sobre os objetos que estão presentes nas figuras, a documentação do Hibernate diz o seguinte:

“

Objetos persistentes e coleções

Objetos, de vida curta, single threaded contendo estado persistente e função de negócios.

Esses podem ser JavaBeans/POJOs, onde a única coisa especial sobre eles é que são associados a (exatamente uma) Session. Quando a Session é fechada, eles são separados e liberados para serem usados dentro de qualquer camada da aplicação (Ex. diretamente como objetos de transferência de dados de e para a camada de apresentação).

Objetos e coleções desanexados e transientes

Instâncias de classes persistentes que ainda não estão associadas a uma Session.

Eles podem ter sido instanciados pela aplicação e não persistidos (ainda) ou eles foram instanciados por uma Session encerrada.

— (KING et al., 2008, p. 31)

Já a Session representa uma conversa entre a aplicação e o banco. Com um cache de primeiro nível obrigatório, ela controla o que pode ser acessado e por quem.

Cache

Em ciência da computação, cache é uma forma de devolver, de forma mais rápida, dados que são acessados várias vezes. Tendo isso em vista, pode-se dizer que o Hibernate possui dois tipos diferentes de cache: o de primeiro nível e o de segundo nível.

O primeiro nível serve para garantir que os objetos estarão vivos apenas quando a sessão estiver aberta, comportamento que é bem-vindo, pois permite que várias instâncias acessem o objeto ao mesmo tempo, mantendo as propriedades ACID do banco.

Já o segundo nível serve para guardar consultas que acontecem bastante na aplicação. Além disso, ele é compartilhado entre todas as sessões ativas no banco no momento. O SessionFactory (Figura 2) que controla aquela classe deve indicar explicitamente que ela poderá ter esse cache.

VIDEOAULA: ARQUITETURA GERAL DO HIBERNATE: COMPONENTES FUNÇÕES E FORMAS DE INTEGRAÇÃO

O vídeo mostrará o conceito básico sobre o que é Hibernate ORM, conceituando o que é um framework de persistência de dados, além de falar sobre a própria estrutura do Hibernate, como o que são as sessions e a session factory, e as potencialidades, como uso de diferentes níveis de cache para a persistência dos objetos.

Videoaula: Arquitetura Geral do Hibernate: componentes funções e formas de integração

Para visualizar o objeto, acesse seu material digital.

INSTALAÇÃO E CONFIGURAÇÃO: EXEMPLO BÁSICO POJO COM BANCO CRIADO, PERSISTÊNCIA E INTEGRAÇÃO COM JDBC E JEE

A persistência de dados em Java pode ser feita de diferentes formas, sendo algumas delas o Java Persistence API (JPA) ou o Hibernate ORM, nosso objeto de estudo. Em ambos utilizamos os chamados Plain Old Java Object (POJO) para a persistência, que são nada mais que objetos simples, os quais não necessitam de classes ou interfaces externas.

A organização da persistência de dados através de POJOs abstrai uma responsabilidade outrora assumida pelos Java Beans ou Enterprise Java Beans, que eram formas de acesso aos dados da aplicação com regras e convenções mais rígidas e que necessitavam de padrões diferentes para ferramentas diferentes.

Os POJOs não atendem a um padrão de projeto específico, são apenas objetos normais, com pouca ou nenhuma notação, que carregam parte da regra de negócio da aplicação e que auxiliam na versatilidade do código. Exemplo de uma classe POJO:

```

1  public class Carro implements java.io.Serializable {
2
3      private String nome;
4
5      private String cor;
6
7      public Carro() {
8
9
10     public Carro(String nome, String cor) {
11         this.nome = nome;
12         this.cor = cor;
13     }
14
15     public String getCor() {
16         return cor;
17     }
18
19     public void setCor(String cor) {
20         this.cor = cor;
21     }
22
23     public String getNome() {
24         return nome;
25     }
26
27     public void setNome(String nome) {
28         this.nome = nome;
29     }
30 }
```

(PLAIN..., 2020, [s. p.]).

Na configuração da persistência, você pode criar as tabelas no banco MySQL manualmente ou então delegar essa responsabilidade ao framework. No exemplo adiante, deixamos esse trabalho a cargo do próprio Hibernate (linha 19 do arquivo persistence.xml).

Nele, estamos utilizando *create-drop*, que tentará criar as tabelas caso não existam, porém também é possível utilizar o *update*. Atente-se ao uso desses recursos e faça suas escolhas com sabedoria, pois as consequências podem ser diversas. Caso deseje apenas criar as tabelas manualmente e não utilizar a criação automatizada, o seguinte código SQL (Structured Query Language) deve ser rodado no banco:

```

1  create table CARRO (
2      id INT NOT NULL auto_increment,
3      nome_do_carro VARCHAR(100) default NULL,
4      cor_do_carro VARCHAR(20) default NULL,
5      PRIMARY KEY (id)
6 );
```

O mapeamento do Hibernate para criarmos a classe "Carro", com as propriedades nome e cor, seria algo relativamente simples, sendo necessário apenas que o banco onde a informação será persistida já exista e esteja configurado.

Estrutura do arquivo persistence.xml:

```

1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE hibernate-mapping PUBLIC
3  "-//Hibernate/Hibernate Mapping DTD//EN"
4  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
5
6  <hibernate-mapping>
7  <class name = "Carro" table = "CARRO">
8
9  <meta attribute = "class-description">
10     Essa classe tem os atributos do carro
11 </meta>
12
13 <id name = "id" type = "int" column = "id">
14 <generator class="native"/>
15 </id>
16
17 <property name = "corDoCarro" column = "cor_do_carro" type = "string"/>
18 <property name = "nomeDoCarro" column = "nome_do_carro" type = "string"/>
19 <property name="hibernate.hbm2ddl.auto" value="create-drop" />
20
21 </class>
22 </hibernate-mapping>

```

A configuração do Hibernate também é feita em um arquivo XML e, em nosso arquivo hibernate.cfg.xml, podemos ver algumas propriedades importantes como o *Dialect*, que permite adaptar a sintaxe das consultas que o Hibernate gerará para diferentes tipos de bancos de dados.

```

1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE hibernate-configuration SYSTEM
3  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
4  <hibernate-configuration>
5  <session-factory>
6
7  <property name = "hibernate.dialect">
8  org.hibernate.dialect.MySQLDialect
9  </property>
10
11 <property name = "hibernate.connection.driver_class">
12 com.mysql.jdbc.Driver
13 </property>
14
15 <property name = "hibernate.connection.url">
16 dbc:mysql://localhost/CARRO
17 </property>
18
19 <property name = "hibernate.connection.username">
20 admin_chevrolet
21 </property>
22
23 <property name = "hibernate.connection.password">
24 presidente123
25 </property>
26
27 <!-- Lista de arquivos XML de mapeamento -->
28 <mapping resource = "persistence.xml"/>
29
30 </session-factory>
31 </hibernate-configuration>

```

Além da propriedade *Dialect* citada anteriormente temos especificado também o *Driver* que fará a conexão com o banco através do protocolo JDBC, que é um padrão de comunicação utilizado para integração da aplicação com qualquer banco de dados relacional e o qual garante que as consultas SQL serão transferidas em sequência para o banco através de sua API.

A ideia por trás do Hibernate é transformar as classes, já criadas pelo desenvolvedor, em tabelas de um banco de dados relacional, seguindo o mapeamento proposto. Não sendo exclusividade de aplicações Java, existem outros frameworks de persistência de dados em outras linguagens de programação como o Doctrine (PHP) e o Entity Framework (C#).

Como as consultas serão montadas em HQL e traduzidas automaticamente para SQL em tempo de execução, essa abordagem permite que a aplicação migre de um banco MySQL para outro PostgreSQL ou Oracle, por exemplo, apenas com a mudança da propriedade *Dialect*.

Como o Hibernate já encapsula as consultas, é perfeitamente possível mudar de um tipo de banco de dados relacional para outro com alterações mínimas na aplicação.

VIDEOAULA: INSTALAÇÃO E CONFIGURAÇÃO: EXEMPLO BÁSICO POJO COM BANCO CRIADO, PERSISTÊNCIA E INTEGRAÇÃO COM JDBC E JEE

O vídeo mostrará o que é uma classe POJO e como ela centraliza as responsabilidades no Hibernate, sendo a fonte do mapeamento que virará o banco de dados. Além disso, temos uma forma básica de configuração da ferramenta e o arquivo de mapeamento dos atributos de uma classe que fará a integração para o banco de dados.

Videoaula: Instalação e configuração: exemplo básico POJO com banco criado, persistência e integração com JDBC e JEE

Para visualizar o objeto, acesse seu material digital.

MAPEAMENTO OBJETO-RELACIONAL (ORM): CONCEITOS DO ORM; ENTENDIMENTO DA CORRESPONDÊNCIA ENTRE AS ENTIDADES DE CÓDIGO E BANCO E SUAS RELAÇÕES; EXEMPLO DE CRIAÇÃO DO BANCO A PARTIR DO ORM

Dentro do banco de dados, os dados dispostos como objetos são bem diferentes daqueles dispostos como entidades relacionais, porém podemos conectar um mundo ao outro através de frameworks que fazem essa ponte, como é o caso do Hibernate.

Essa diferença de abordagens nos introduz a um problema: a Impedância Objeto Relacional, que, segundo Bauer e King (2007, p. 9), é explicitada quando analisamos o SQL:



Se considerarmos SQL e bancos de dados relacionais novamente, conseguimos observar, finalmente, a disparidade entre os dois paradigmas. Operações SQL como projeções e junções sempre resultam em uma representação tabular do resultado dos dados. (Isso é conhecido como fechamento transitivo, o resultado das operações sobre relações é sempre uma relação).

Isso é um pouco diferente da rede de objetos interconectados utilizada para executar a lógica de negócios na aplicação Java. Esses são modelos fundamentalmente diferentes, não apenas modos diferentes de visualizar o mesmo modelo.

Assim, as consultas ao banco de dados serão feitas em HQL e serão traduzidas em tempo real por SQL pelo próprio framework. Além disso, ele gerencia todas as transações, não sendo necessário abrir conexão ou iniciar transações explicitamente. Também temos a vantagem da diminuição de código repetido para manipular dados no banco, pois, caso quiséssemos fazer a conexão diretamente com o banco utilizando JDBC, os métodos precisariam ser refeitos por classe.

Um dos diferenciais do Hibernate é que o mapeamento das classes para que se tornem banco não precisará ser pensado individualmente, ou seja, apenas por criarmos as classes e seu mapeamento inicial, já teremos o banco organizado de forma que todos os seus relacionamentos se conheçam. O código escrito na aplicação, com a implementação do Hibernate, fará o trabalho da criação do banco nos bastidores.

A fim de que as consultas dos dados aconteçam com sucesso, o banco precisa ser criado com orientação objeto relacional, e os objetos que virão as informações persistidas precisarão ser mapeados apenas uma vez. Essa abordagem dá ao desenvolvedor a liberdade de pensar apenas o código, assim como todas as vantagens da orientação a objetos.

O Hibernate resolverá problemas para aplicações grandes que tenham os dados utilizados por diferentes classes e com relacionamentos complexos, não se observando ganho prático do seu uso em aplicações de menor escopo. Outro ponto positivo é a flexibilidade para mudança nas colunas e nos relacionamentos, visto que basta mudar o código para que elas sejam refletidas na aplicação.

Vale lembrar ainda que a maneira generalista pela qual frameworks podem ter de lidar com as consultas pode trazer uma queda de performance, mas algumas estratégias podem ajudar na otimização deles. Nesse sentido, não é interessante que as consultas precisem ser ajustadas em HQL caso a caso, pois uma das vantagens do framework se perderia, mas há casos em que são necessárias algumas configurações extras para não se perder performance na aplicação.

VIDEOAULA: MAPEAMENTO OBJETO-RELACIONAL (ORM): CONCEITOS DO ORM; ENTENDIMENTO DA CORRESPONDÊNCIA ENTRE AS ENTIDADES DE CÓDIGO E BANCO E SUAS RELAÇÕES; EXEMPLO DE CRIAÇÃO DO BANCO A PARTIR DO ORM

Nesse vídeo mostramos a relação entre o mapeamento ORM e o paradigma da impedância de dados, deixando clara a forma como o ORM consegue resolver o problema. Além disso, mostramos as relações básicas entre SQL e HQL e como o código das consultas será abstraído quando feito pelo Hibernate.

Videoaula: Mapeamento objeto-relacional (ORM): conceitos do ORM; entendimento da correspondência entre as entidades de código e banco e suas relações; exemplo de criação do banco a partir do ORM

Para visualizar o objeto, acesse seu material digital.

ESTUDO DE CASO

A empresa onde você trabalha já usa o Hibernate e está analisando a possibilidade de mudar o banco de dados de MySQL para Postgree, mas ainda não sabe se será fácil ajustar a aplicação para que funcione com esse novo banco. Quais mudanças no Hibernate você proporia para a gestão do seu projeto?

RESOLUÇÃO DO ESTUDO DE CASO

No arquivo de configuração do Hibernate, será necessário trocar o *Dialect*, o driver da conexão e o endereço para acesso ao banco em si. Essas novas propriedades são as seguintes:

Dialect:

org.hibernate.dialect.PostgreSQLDialect

Driver:

connection.driver_class - org.postgresql.Driver

URL de exemplo:

jdbc:postgresql://localhost/banco_de_dados

Resolução do Estudo de Caso

Para visualizar o objeto, acesse seu material digital.

6º Saiba mais

Seguir o princípio ACID é imprescindível para garantir que as informações do banco de dados sejam confiáveis. Veja mais no link a seguir:

<https://pt.wikipedia.org/wiki/ACID>.

CONSULTAS EM FRAMEWORK HIBERNATE

Nesta aula veremos o que é o *Hibernate Query Language* (HQL) e em que patamar ele se coloca em relação ao SQL. Será que ele faz tudo o que o SQL (Structured Query Language) faz?

42 minutos

INTRODUÇÃO

Nesta aula veremos o que é o *Hibernate Query Language* (HQL) e em que patamar ele se coloca em relação ao SQL. Será que ele faz tudo o que o SQL (Structured Query Language) faz? As consultas que o Hibernate faz são todas traduzidas em tempo real para SQL antes de saírem da aplicação pelo JDBC (Java Database Connectivity).

A diferença crucial entre SQL e HQL é que este trabalha com a persistência de objetos (classes e propriedades das classes) e aquele faz as consultas nas tabelas e colunas. O Hibernate, através do HQL, propicia uma abstração tão alta que as consultas criadas pelo framework em HQL acontecem em SQL de forma transparente para o desenvolvedor.

VISÃO GERAL DO HQL: A LINGUAGEM DE CONSULTA DO HIBERNATE E SEU PARALELO COM O USO DO SQL PADRÃO

A linguagem HQL tem uma sintaxe muito parecida com a do SQL, o que facilita a aprendizagem do primeiro para quem já conhece o último. Apesar disso, as diferenças vão ficando evidentes assim que nos aprofundamos mais no assunto.

No HQL as consultas não diferem entre letras maiúsculas ou minúsculas, exceto no nome das classes e propriedades das classes, portanto org.hibernate.eg.KROTON não é igual a org.hibernate.eg.Kroton, mas InSert é igual a INSERT, por exemplo.

A consulta mais simples possível em HQL é o *from*, para a qual não precisamos qualificar o nome da classe. Diante disso, vamos utilizar uma classe fictícia, para fins de exemplo, chamada teste.Carro.

Podemos consultá-la como:

```
"from teste.Carro"
```

Ou apenas como:

```
"from Carro"
```

Isso pode acontecer porque o *auto-import* é padrão nas consultas. Caso seja necessário se referir ao objeto "Carro" em uma outra parte da consulta, é necessário o uso de *alias*:

```
"from Carro as carro"
```

Vale lembrar que a palavra-chave "as" não é obrigatória, mas deixá-la no código não traz prejuízo e, ao contrário, ajuda na legibilidade da consulta. Além disso, a boa prática para nomes de alias é que se siga o padrão de criação para variáveis locais, como: novoCarro.

As consultas podem retornar mais de um objeto por vez:

```
"from Carro, Avião"
```

E também temos as associações presentes no SQL, como *inner join*, *left join*, *right outer join*:

```

1 "from Carro as carro
2 join carro.marca as marca
3 left join carro.modelo as modelo"

```

Além disso, as funções de agregação, bem comuns no SQL, também podem ser feitas no HQL. Essas funções vão retornar um valor respectivo à seleção que foi feita, como a média ou a quantidade total de ocorrências. Veja como ficaria, por exemplo, um *count*:

```

1 "select count(carro)
2 from Carro as carro"

```

Além do *count*, outras funções de agregação disponíveis no HQL são: avg, sum, min, max. As palavras "*distinct*" e "*all*" podem ser combinadas nas agregações para filtrar dados repetidos ou grupos completos. Uma consulta com *distinct* no HQL seria muito parecida com a do SQL:

```

1 "select distinct carro.modelo from Carro carro"
2 "select count(distinct carro.modelo), count(carro) from Carro carro"

```

SQL:

```

1 "select distinct modelo from Carro"
2 "select count(*) distinct modelo from Carro"

```

A cláusula *where* permite limitar o escopo da consulta para uma tabela específica, assim como no SQL. Segue exemplo:

```
"from Carro carro where carro.modelo.ano is not null"
```

Pela natureza das consultas do HQL, podemos acabar fazendo mais *inner joins* do que gostaríamos. Isso acontece pois o HQL busca objetos, os quais podem estar espalhados dentro de outros objetos compostos. Um *inner join* é uma consulta que une duas tabelas diferentes. Nesse sentido, veja um exemplo que, ao ser traduzido em SQL, terá *inner join* e outro que não terá:

```

1 "from Carro as carro where carro.id = 50
2 from Carro as carro where Carro.modelo.id = 70"

```

A primeira consulta necessitará de um *inner join*, e a segunda, com o caminho completo, não, fato que a deixa mais eficiente.

O HQL tem dois tipos de união, a implícita e a explícita. A forma implícita usa apenas um ponto ("."), que pode aparecer em qualquer cláusula, e a explícita usa a palavra "união". A implícita sempre gerará declarações SQL que têm uniões inteiras. Além disso, as consultas em HQL podem retornar diferentes propriedades ou objetos do tipo object[] ou List.

VIDEOAULA: VISÃO GERAL DO HQL: A LINGUAGEM DE CONSULTA DO HIBERNATE E SEU PARALELO COM O USO DO SQL PADRÃO

No vídeo, falamos sobre as funcionalidades básicas do HQL e como são os paralelos com o SQL em alguns pontos. Conceituamos o que é uma consulta em um banco de dados; fazemos uma comparação básica sobre a performance de chamadas HQL, com cache ou não, *versus* chamadas puras em SQL; e apresentamos uma comparação do custo-benefício entre elas.

Videoaula: Visão geral do HQL: a linguagem de consulta do Hibernate e seu paralelo com o uso do SQL padrão

Para visualizar o objeto, acesse seu material digital.

CONTROLE DE TRANSAÇÕES DO HQL UTILIZANDO TRANSAÇÕES E FILTRAGEM DE DADOS

Uma transação é uma consulta no banco de dados que tem início, meio e fim e que pode fazer várias operações de uma vez. É mandatório que, se uma das consultas da transação falhar, ela não deve continuar e o seu *rollback* deve ser efetuado. Esse é um dos princípios ACID chamado atomicidade, o qual fica a cargo do Hibernate no nosso caso.

Outro conceito ACID, a consistência, afirma que as transações devem ser isoladasumas das outras para que seja possível ter consistência de dados no banco. No Hibernate, tratamos transações como unidades de trabalho e, para que as propriedades ACID se mantenham e sejam preservadas, são necessárias algumas operações.

Veja a seguir quais são elas e seus significados:

- *Begin-transaction*: inicia a execução da transação.
- *Commit-transaction*: significa que tudo pode ser gravado no banco.
- *Rollback*: deve desfazer as alterações já feitas quando alguma transação não for bem-sucedida.
- *End-transaction*: delimita o fim da transação e é o gatilho para iniciar o *commit* ou fazer o *rollback*.
- *Undo*: desfaz uma transação.
- *Redo*: refaz uma transação.

Exemplo de uma transação:

```

1 begin transaction
2 update preco set novoPreco = 45
3 where Preco = 35
4 if @@ERROR <> 0
5 rollback
6 else
7 commit
8 end transaction

```

A transação foi feita de modo que, na primeira impossibilidade de atualizar o preço de R\$35 para R\$45, todos os campos que já haviam sido atualizados foram revertidos. Isso é o que garante todos os princípios ACID.

Controle de concorrência otimista

Para manter a concorrência das transações sob controle em cenários com volumes muito altos de transações, podemos utilizar a técnica do versionamento. Uma transação pode ter um identificador único, como um ID, ou um *timestamp* para evitar conflitos nas atualizações e impedir as perdas no meio do caminho.

Essa propriedade deve ser mapeada na classe com enquanto o Hibernate cuida de validar se os dados mudaram para incrementar ou não o identificador.

Em uma aplicação com poucas transações concorrentes, basear-se no último *commit* realizado é a estratégia padrão, pois não é necessário utilizar o versionamento.

Os filtros funcionam como uma *view* de banco de dados, limitando um conjunto de dados de maneira muito parecida da função *where*. No Hibernate, esse filtro pode decidir quais dados entram e quais saem da seleção em tempo real, porém eles não vêm habilitados por padrão, e devem ser declarados e explicitamente acoplados a uma classe que se deseja filtrar. Na *session*, os métodos que implementam os filtros são: `enableFilter("nomeFiltro")`, `getEnabledFilter("nomeFiltro")` e `disableFilter("nomeFiltro")`.

No código HQL da consulta, habilite o filtro antes das *queries* que buscam os dados que se deseja filtrar e, caso você deseje usar um filtro padrão para todas as consultas sem filtro declarado, basta definir isso dentro de "`<filter-def >/filter-def>`".

VIDEOAULA: CONTROLE DE TRANSAÇÕES DO HQL UTILIZANDO TRANSAÇÕES E FILTRAGEM DE DADOS

Nesse vídeo conceituamos o que é uma transação com o intuito de explicarmos que o Hibernate faz o controle automático dela. Além disso, mostramos um simples exemplo de transação, a qual é comentada linha a linha, e uma explicação sobre controle de concorrência otimista com uso de versionamento ou *timestamps* e controle de concorrência pessimista.

Vídeoaula: Controle de transações do HQL utilizando transações e filtragem de dados

Para visualizar o objeto, acesse seu material digital.

SQL NATIVO E PERFORMANCE

Consultas oriundas de ORMs serão traduzidas e feitas em SQL, em tempo real, inclusive as feitas em HQL. Essa carga de processamento em tempo de execução não quer dizer que o SQL seja naturalmente mais rápido que HQL. Nesse sentido, a melhor alternativa, então, seria melhorar a performance do Hibernate, na medida do possível, sem apelar para consultas SQL, pois o Hibernate possui os dois tipos de caches, que podem acelerar bastante uma consulta, já que as consultas feitas direto em SQL não contam com esse tratamento.

Carregar apenas a coluna que se deseja pode ser irrelevante em termos de performance, pois o gargalo estaria na busca pela coluna e não na transferência dos dados para a aplicação. Logo, um problema de performance seria acessar várias vezes a mesma tabela para trazer colunas diferentes, o que faria a aplicação refazer processos para atualizar essa nova informação.

Ao carregar uma classe com o básico `"session.load(nomeDaClasse, idDaClasse)"`, caso ela ainda não esteja no cache, será gerado um *"from"* em SQL, como `"select * from nomeDaTabela"`, com velocidade equiparável ao *"from"* do HQL. Assim, o ganho de performance explícito diria respeito aos termos uma classe já carregada em cache.

O custo-benefício das decisões de manter tudo em HQL, de refiná-lo ou de usar SQL deve ser bem analisado, pois o Hibernate apresenta benefícios como atualização automática das entidades e *dirty checking*, que faz a verificação quanto à necessidade de algum objeto ainda precisar ser atualizado sem alterar em nada os que já o foram. Essa validação, que é automática, substituiria várias consultas de escrita caso a consulta fosse feita em SQL puro.

Agora, como podemos verificar, na aplicação, se a performance está satisfatória? Ou até mesmo prever possíveis problemas antes mesmo de eles começarem? Quando configuramos o Hibernate com a propriedade `"show_sql"` para `"true"` ele mostrará o SQL que foi gerado em todas as consultas. Dessa forma, pode-se "debugar" as consultas e ter certeza de que elas são as certas para você, isso ao comparar o que você tem com o que gostaria de ter.

Encontrar e solucionar os problemas antes de eles sequer acontecerem é o cenário ideal, mas muito difícil de conseguir. Para auxiliar nessa tarefa, o Hibernate nos oferece algumas ferramentas de estatísticas e *logs*. Com isso, você perceberá que muitos dos problemas não acontecem nos testes durante o desenvolvimento, porém basta a aplicação subir para as luzes de emergência acenderem. A escala da aplicação e o número de consultas concorrentes podem impactar a aplicação.

Para ter mais informações sobre o que está acontecendo, ative o módulo de estatísticas configurando as propriedades `"hibernate.generate_statistics"` para `true`, e `"org.hibernate.stat"` para DEBUG. Essas propriedades lhe mostrarão, ao fim de cada *session*, o número de conexões abertas, o tempo que cada uma delas levou, a quantidade de *statements* e de colunas retornadas.

VIDEOAULA: SQL NATIVO E PERFORMANCE

Nesses vídeos demonstramos quais são as vantagens do HQL em relação ao SQL quanto à performance e mostramos os comandos que podem ser utilizados para acompanhamento de performance e monitoria da qualidade das consultas em geral. Falamos sobre custo-benefício entre usar apenas HQL ou usar SQL e conceitos básicos de performance em banco de dados.

Vídeoaula: SQL nativo e performance

Para visualizar o objeto, acesse seu material digital.

ESTUDO DE CASO

O tempo das consultas criadas em HQL pode se tornar um problema e até mesmo atrapalhar a aplicação mesmo que tudo esteja configurado corretamente. Sabendo de algumas estratégias que podem ser aplicadas para melhorar a performance do Hibernate, como a performance da consulta padrão poderia ser melhorada?

RESOLUÇÃO DO ESTUDO DE CASO

A melhor escolha entre *Lazy loading* e *Eager loading* é uma dessas estratégias que pode melhorar a performance da aplicação. No nosso caso, faz sentido carregá-las de uma vez com o *Eager Loading*, ou seja, o padrão da aplicação.

O *Lazy Loading* não carregará todas as características do objeto de uma vez assim que ele for mencionado no código, pois ele vai buscando as informações aos poucos conforme são requisitadas. Isso diminui o tempo inicial de carregamento desse objeto na consulta.

Já o *Eager Loading* carrega todo o objeto de uma vez na memória assim que ele é mencionado na primeira vez, gerando um tempo maior de carregamento inicial, mas uma disponibilidade imediata dos dados desse objeto em seguida. Cabe ao desenvolvedor da aplicação, então, escolher a abordagem que melhor se encaixa à realidade da aplicação que está sendo criada.

Resolução do Estudo de Caso

Para visualizar o objeto, acesse seu material digital.

↳ Saiba mais

Conhecer SQL é importante para entender o que será gerado pelo HQL. Na página a seguir, temos uma série de dicas e comandos SQL simples e avançados para estudo: <https://www.sqltutorial.org/sql-cheat-sheet/>.

Aula 3

CONCEITOS DE FRAMEWORK MOBILE

Um framework de software pode ser considerado uma plataforma que facilita a construção de aplicativos. Toda essa estrutura de linguagem de programação tem como característica permitir a construção de softwares robustos, eficientes e versáteis (FERRAZ, 2018).

43 minutos

INTRODUÇÃO

Um framework de software pode ser considerado uma plataforma que facilita a construção de aplicativos. Toda essa estrutura de linguagem de programação tem como característica permitir a construção de softwares robustos, eficientes e versáteis (FERRAZ, 2018). Utilizando essas plataformas, o desenvolvedor se preocupará somente com as funcionalidades de alto nível dos aplicativos, as voltadas para a usabilidade do

software, pois as de baixo nível já estão embutidas nessas ferramentas (SILVA, 2016). Como existem inúmeras dessas plataformas, uma para cada diferente linguagem de programação, aqui serão abordados importantes frameworks para as linguagens Android (ADT), React e Swift. Vamos aprender e fazer bom uso dessas facilidades do desenvolvimento de software!

FRAMEWORKS UTILIZADOS EM ANDROID (ADT), REACT E SWIFT

O *Android Development ToolKit* (ADT) permite que se adicionem, dentro da IDE (*Integrated Development Environment*) Eclipse, outras ferramentas de desenvolvimento, as quais trazem agilidade e facilidade na criação da sua aplicação Android (DEITEL; DEITEL; WALD, 2016). Como exemplo dessas ferramentas pode-se citar:

- *DDMS (Dalvik Debug Monitor Server)*: permite realizar capturas de telas e gerenciamento de pontos de interrupções e *threads*.
- *New Project Wizard*: facilita a criação e configuração de um projeto de aplicativos Android.
- Editor de código Android: permite a criação fácil de código XML.

Além dessas ferramentas que agilizam a criação e o gerenciamento dos aplicativos, ou *apps (applications)* Android, existem frameworks que possibilitam a criação de programas com mais funcionalidades para os usuários (DEITEL; DEITEL; DEITEL, 2015). Dentre eles, estão:

- ***Ormelite***: facilita o desenvolvimento de aplicações utilizando o banco de dados local *SQLite* do Android, sem a necessidade de escrita de código SQL (*Structured Query Language*).
- ***Retrofit***: permite desenvolver aplicações utilizando serviços *REST (Representational State Transfer)* de forma segura e sem complicações.
- ***Universal Image Loader***: acelera o carregamento de imagens utilizando *cache* de memória em disco.
- ***NativeScript***: permite a criação de código em HTML (*HyperText Markup Language*), CSS (*Cascading Style Sheets*) e *JavaScript*.
- ***Ionic***: permite a criação de aplicativos Android híbridos, integrando HTML5, CSS3 e *JavaScript Angular*.

O React é uma biblioteca *JavaScript open-source*, ou de código aberto, que possibilita a produção de *interfaces* de usuários com alta usabilidade tanto para sistemas *mobile* quanto para sistemas web. O principal objetivo do React é produzir *interfaces* com a melhor renderização possível. Para auxiliar na construção dessas interfaces, o React permite utilizar diversos frameworks (SILVA, 2016). A seguir são apresentados alguns deles.

- ***Redux***: pode ser considerado um framework de solução de gerenciamento de status de aplicativos *JavaScript*.
- ***React Admin***: utilizado para criar aplicativos de administração B2B (*Business-to-business*) e *REST APIs (Applications Protocol Interface)*.
- ***Ant Design***: conjunto de componentes React para o design de UI (*Universal Image*) de classe corporativa e implementação para interfaces de usuário.
- ***React-Motion***: framework de animação que utiliza configurações da biblioteca *Spring* para descrever as animações.
- ***Styled Components***: ferramenta CSS que permite criar componentes reutilizáveis, os quais ajudarão a melhorar a aparência do seu aplicativo.

A Swift é uma linguagem relativamente nova desenvolvida pela Apple e utilizada no desenvolvimento de aplicativos para os Mac OS, macOS, iPad OS, iOS, watch OS, Linux e tv OS. A linguagem apresenta como características ser compilada, multiuso e multiparadigma, além de ser voltada para a expressividade, o desempenho e a segurança. Como as demais linguagens de desenvolvimento *mobile*, a Swift permite a utilização dos frameworks para facilitar o desenvolvimento dos aplicativos (SILVA, 2016). A seguir são apresentados alguns desses frameworks.

- ***Quick***: é um framework de teste de software.
- ***Vapor***: permite o desenvolvimento de projetos web, API e baseados em nuvem.
- ***Kitura***: framework web e *HTTP server*.
- ***Zip***: permite criar aplicativos para realizar a compactação e a descompactação de arquivos.

Neste contexto, foram apresentados alguns dos frameworks mais utilizados nas linguagens Android ADT, React e Swift. Com isso, é preciso dizer que existem diversos outros frameworks que podem ser aplicados para a facilitar a construção de aplicativos móveis.

VIDEOAULA: FRAMEWORKS UTILIZADOS EM ANDROID (ADT), REACT E SWIFT

O vídeo apresentará informações sobre as linguagens Android ADT, React e Swift considerando os frameworks utilizados em cada uma delas. No texto, foram apresentados cinco frameworks e suas aplicações para cada uma das linguagens; no vídeo, pretende-se abordar diferentes frameworks, dando uma breve explicação da sua utilização e aplicação.

Videoaula: Frameworks utilizados em Android (ADT), React e Swift

Para visualizar o objeto, acesse seu material digital.

CONCEITO DE ACTIVITY, CICLO DE VIDA, ENTRADA E SAÍDA, EVENTOS

A utilização de aplicativos *mobile*, os chamados *apps*, é totalmente diferente quando comparados com os aplicativos para computadores (INTRODUÇÃO..., 2019). Os usuários desses *apps* iniciam a sua utilização de forma aleatória, fazendo com que um aplicativo possa ser inicializado de inúmeras formas. Um exemplo disso é a utilização dos e-mails, que são inicializados de forma diferentes se chamados pelos seus *apps* de origem ou por um *app* de alguma rede social. Assim, para facilitar a iteração entre a execução das atividades de seus *apps*, a linguagem Android disponibiliza a classe *activity*.

Quando um *app A* chama outro *app B*, o que realmente acontece, é que o *app A* solicita a execução de uma atividade ou tela específica do *app B* e não de todo o *app*. Assim, as atividades ou telas são consideradas o ponto de entrada da iteração dos usuários com os aplicativos.

Os *apps* como um todo são compostos por diversas telas, o que faz que cada um possibilite a execução de diversas atividades. Nesse sentido, a primeira tela que o *app* exibe para o usuário, é denominada de atividade principal do *app*. Assim, cada atividade do *app* pode iniciar diversas outras para realizar alguma ação diferente. Considerando o exemplo do e-mail, a primeira tela do *app* de e-mail apresenta a caixa de entrada. A tela da caixa de entrada permite que se tenha acesso a diversas outras atividades ou telas do e-mail.

O Android fornece alguns recursos para facilitar o gerenciamento das atividades, sendo um deles o filtro *Intent*, que permite a uma solicitação de atividade ser realizada tanto de forma explícita (pelo próprio aplicativo, por exemplo, abrir o e-mail diretamente no Gmail) quanto implícita (abrir o e-mail por outro aplicativo). Outro recurso que o Android disponibiliza para o tratamento das atividades é realização de um bom gerenciamento do ciclo de vida dessas atividades. Lembrando que, enquanto uma atividade estiver sendo executada, ela possui um ciclo de transições de estados. Para tratar dessas transições, o Android utiliza diversas funções *callbacks*, ou funções que são passadas como parâmetros para outras funções. A seguir são apresentadas as *callbacks* disponibilizadas pelo Android e suas respectivas funções.

- ***onCreate()*:** utilizada para criar uma atividade. Essa função deve ser utilizada com a função *setContentView()*, a qual define o layout da interface do usuário da atividade.
- ***onStart()*:** após a *onCreate()* finalizar a criação da atividade, ela passa para o estado criado e a função *onStart()* deve ser chamada para que a atividade possa ter iteração com o usuário.
- ***onResume()*:** antes de se iniciar a iteração da atividade com o usuário, a callback *onResume()* entra em ação e as principais funcionalidades da aplicação devem ser implementadas nessa função.
- ***onPause()*:** logo após a *onResume()* a callback *onPause()* inicia a execução. Essa função é sempre chamada quando se inicia um estado de pausa ou quando o usuário realizou uma atividade de voltar ou recarregar a tela, por exemplo.
- ***onStop()*:** sempre se executa essa callback após a *onPause()*, quando a atividade não está mais visível para o usuário.
- ***onRestart()*:** caso o usuário volte a interagir com a atividade parada, a próxima callback é a *onRestart()*.
- ***onDestroy()*:** caso o usuário não volte a interagir com a atividade parada, a próxima callback é a *onDestroy()*. A execução da *onDestroy()* garante que todos os recursos utilizados pela atividade são liberados pelo sistema.

O ciclo de vida das atividades utiliza as *callbacks* para realizar todas as ações e mudanças de estados das atividades. Assim, o bom entendimento do funcionamento e da aplicação dessas funções ajudará no bom funcionamento da aplicação.

VIDEOAULA: CONCEITO DE ACTIVITY, CICLO DE VIDA, ENTRADA E SAÍDA, EVENTOS

No vídeo da aula, será apresentado um fluxograma com o ciclo de vida das atividades no Android como uma explicação complementar ao material descrito no texto. O fluxograma conterá os estados das atividades e como todas as *callbacks* descritas na aula são utilizadas para a realização da transição desses estados.

Videoaula: Conceito de *activity*, ciclo de vida, entrada e saída, eventos

Para visualizar o objeto, acesse seu material digital.

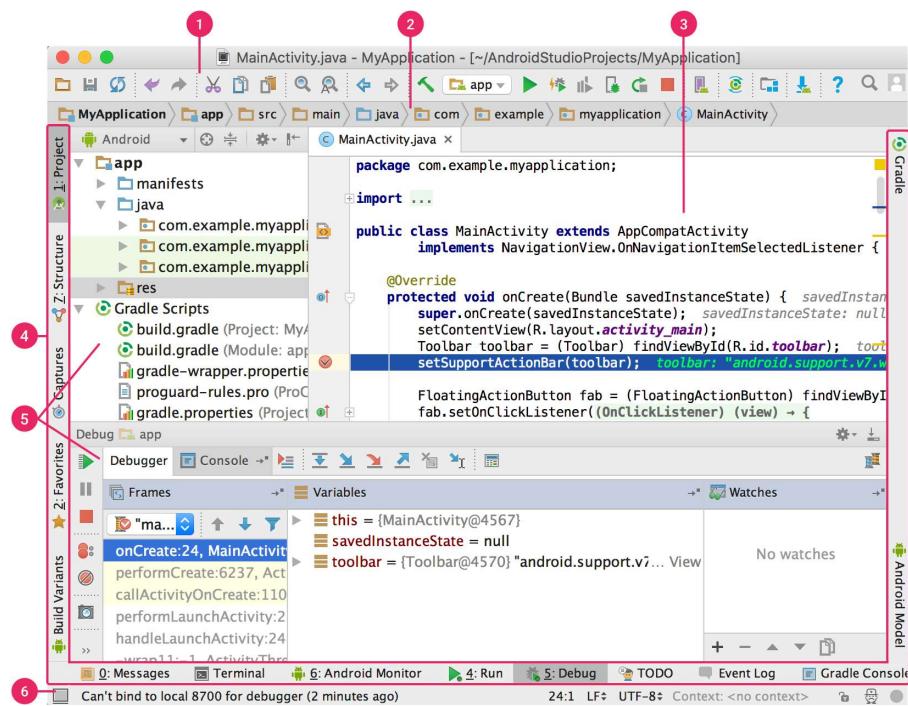
UTILIZAÇÃO DO ANDROID STUDIO

O Android Studio, considerado uma das principais ferramentas para a construção de *apps* (DEVELOPERS, 2021), é um IDE utilizado para o desenvolvimento de aplicações do ambiente Android. A ferramenta tem como base a IDE *IntelliJ IDEA*, por isso possui todas as características do *IntelliJ* e também novas funcionalidades como:

- Sistema de compilação flexível.
- Ambiente unificado, o que possibilita a utilização de todas as versões de plataformas Android.
- Possibilidade de envio de alteração de recursos ao app sem sua inicialização ou interrupção.
- Integração com ferramentas de versionamento de código como o *GitHub*.
- Frameworks de testes.
- Detecção de problemas de usabilidade, compatibilidade entre versões e desempenho.
- Compatibilidade com a linguagem C++.
- Compatibilidade com Google Cloud Platform.

O IDE do Android Studio apresenta uma interface de usuário que contém diversas funcionalidades que facilitam o desenvolvimento de apps, como apresenta a Figura 1.

Figura 1 | Interface do usuário do IDE Android Studio



Fonte: Developers (2021, [s. p.]).

A seguir são apresentadas as descrições de cada item numerado na figura de interface com o usuário.

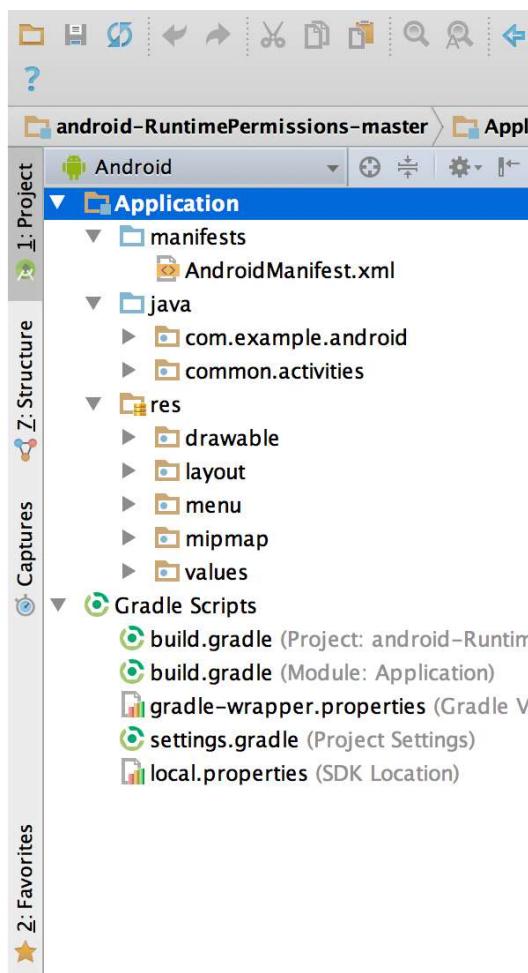
1. **Barra de ferramentas:** permite executar o app e inicializar as diversas ferramentas do Android.
2. **Barra de navegação:** apresenta uma visão compacta do projeto, facilitando a sua navegação.
3. **Janela do editor:** local onde o código do programa será criado e modificado.
4. **Barra de janela de ferramentas:** apresenta menus que permitem expandir e encolher as janelas de cada ferramenta de desenvolvimento.
5. **Janela de ferramentas:** apresenta as ferramentas de gerenciamento de projetos, pesquisa e controle de versões, entre outras.
6. **Barra de status:** exibe todas as mensagens do ambiente de desenvolvimento e status do projeto.

O Android Studio possui uma estrutura de projeto composto por um ou mais módulos, os quais incluem os arquivos de código-fonte e de recursos do sistema. A ferramenta apresenta os seguintes tipos de módulos:

- **Módulos de recurso do app:** recursos modularizados, podendo utilizar recursos do Google Play.
- **Módulo de biblioteca:** gera um contêiner para o código reutilizável, sendo disponibilizado em dois tipos: módulo biblioteca Android, que possui todos os tipos de arquivos compatíveis com o Android, e módulo biblioteca Java, que possui todos os tipos de arquivos compatíveis com Java.
- **Módulo do Google App Engine:** gera um contêiner para o código de back-end do Google Cloud.

O projeto e seus módulos podem ser visualizados no Android Studio, na janela de visualização de projetos, como apresentado na Figura 2.

Figura 2 | Janela de visualização do projeto de desenvolvimento do Android



Fonte: Developers (2021, [s. p.]).

Na janela apresentada na Figura 2, observa-se que todos os itens de criação do projeto estão disponibilizados no nível superior **Gradle Scripts**. A pasta **manifest** contém o arquivo *AndroidManifest.xml* com configurações de criação do projeto. Na pasta **Java**, encontram-se os arquivos de código fonte **Java** e arquivos de testes. Na pasta **res**, referente aos recursos extras do projeto, encontram-se os arquivos de layout XML e imagens, por exemplo.

VIDEOAULA: UTILIZAÇÃO DO ANDROID STUDIO

No vídeo da aula, será apresentado, resumidamente, onde fazer o download e como instalar o Android Studio no computador. Também será demonstrado um passo a passo de como criar um projeto na IDE, de forma rápida e simples. Assim, considera-se que este material possa ser um complemento ao texto apresentado sobre o assunto.

Videoaula: Utilização do Android Studio

Para visualizar o objeto, acesse seu material digital.

ESTUDO DE CASO

A empresa MobileSoft, na qual você é Web Mobile Developer Pleno, está com uma nova demanda de serviço para o desenvolvimento de um aplicativo para celulares Android. O aplicativo em questão terá algumas funcionalidades específicas na tela inicial, como compactar/descompactar arquivos, apresentar imagens e animações, realizar transferência de informações e armazenamento em banco de dados. Com base nas informações apresentadas para o projeto, descreva quais frameworks (dentre aqueles apresentados em aula ou outros) você como Web Mobile Developer Pleno, empregaria no desenvolvimento do aplicativo. Também apresente um modelo de ciclo de vida da atividade de compactar arquivo.

RESOLUÇÃO DO ESTUDO DE CASO

Para iniciar a solução, vamos realizar algumas considerações:

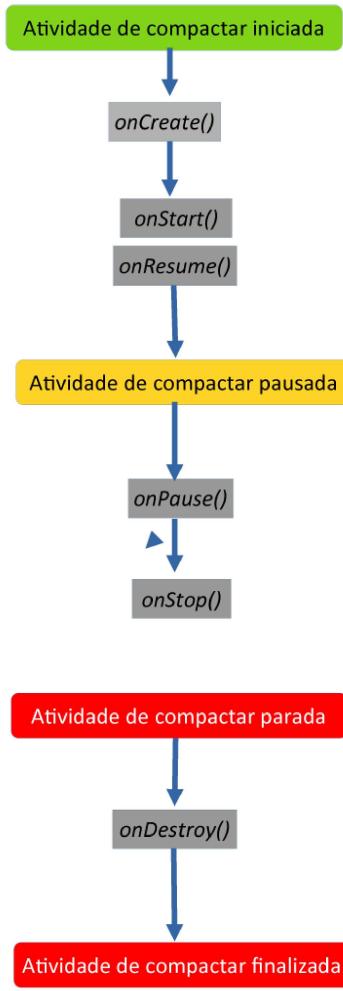
O aplicativo deve:

- Compactar/descompactar arquivos.
- Apresentar imagens e animações.
- Transferir de informações.
- Armazenar informações em banco de dados.

Algumas sugestões de frameworks:

- **Ormelite**: para criação de banco de dados.
- **Universal Image Loader**: auxilia no carregamento de imagens.
- **React Admin**: API Rest para transferência de informações.
- **React-Motion**: framework de animação.
- **Zip**: para compactar e descompactar arquivos.

Um modelo de ciclo de atividades para a atividade de compactar arquivos.



Fonte: Lorem ipsum dolor sit amet.

Tente fazer um ciclo de vida para as outras atividades solicitadas para o aplicativo.

Resolução do Estudo de Caso

Para visualizar o objeto, acesse seu material digital.

6º Saiba mais

O bom gerenciamento das atividades dos aplicativos se deve ao bom conhecimento de como funciona o ciclo de vida das atividades. Assim, no link a seguir, apresenta-se, com mais detalhes, o funcionamento desse ciclo de vida. <https://developer.android.com/guide/components/activities/activity-lifecycle?hl=pt-br>.

Aula 4

CONCEITOS DE FRAMEWORKS MOBILE EM BACK-END

Apesar de não aparecerem diretamente para os usuários, os frameworks back-end desempenham um papel muito importante para o bom funcionamento dos aplicativos móveis.

41 minutos

Os frameworks de desenvolvimento *mobile back-end* são utilizados para armazenar, processar e proteger os dados de um aplicativo *mobile*. Eles caracteristicamente rodam por trás do aplicativo *mobile*, ou seja, os usuários não percebem sua execução.

Apesar de não aparecerem diretamente para os usuários, os frameworks *back-end* desempenham um papel muito importante para o bom funcionamento dos aplicativos móveis. Considerando a importância da utilização desses frameworks, abordaremos alguns conceitos fundamentais relacionados a eles, como armazenamento, web services, banco de dados e arquivos. Também abordaremos algumas aplicações que unem a utilização dos conceitos de frameworks *back-end* e *front-end* no desenvolvimento de aplicativos *mobile*. Vamos estudar!

VISÃO GERAL DE BACK-END MOBILE: ARMAZENAMENTO DE DADOS, SQLITE, ARQUIVOS, WEB SERVICES

O conceito de desenvolvimento *back-end* ou *server side* (lado do servidor) *mobile* está relacionado com toda implementação de software que executa as regras de negócios; a organização das informações apresentadas ao usuário, ou seja, o armazenamento e processamento dos dados em bases de dados relacionais ou não; a segurança dessas informações e a integração desses softwares *back-end mobile* com os web services ou serviços da web (ROVAL *et al.*, 2018).

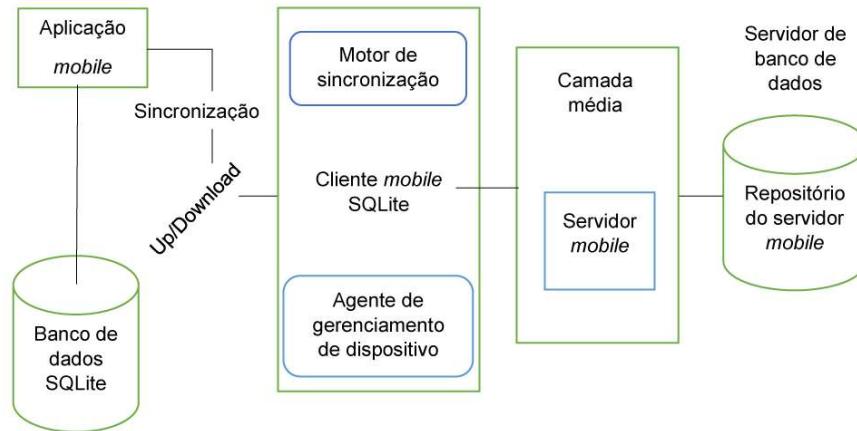
Esse modelo de desenvolvimento pode ser dividido da seguinte maneira: servidor, banco de dados (relacional ou não) e API (*Application Programming Interface*, em português, Interface de Programação de Aplicações).

A organização das informações nos sistemas *mobile* pode ser realizada por meio dos chamados servidores web, os quais podem estar disponíveis fisicamente ou em serviço de nuvem. Para realizar o tratamento dessas informações armazenadas nos servidores, uma das linguagens mais utilizada é a SQLite, que atua como um mini-SGBD (Sistema de Gerenciamento de Base de Dados) e possibilita o controle de inúmeros bancos de dados com diversas tabelas. Por ser um banco de dados compacto, o SQLite está disponível em diversas plataformas *mobile* de forma pública.

Um exemplo

A Figura 1 apresenta um exemplo da arquitetura do cliente de banco de dados SQLite com o cliente do SQLite *mobile* apresentando a sincronização do upload e do download das informações armazenadas em um servidor de banco dados como o Oracle.

Figura 1 | Arquitetura de como o banco de dados SQLite sincroniza as informações de upload/download do cliente *mobile* SQLite



Fonte: adaptado de Oracle (2010).

Vantagens

A integração das diversas aplicações *mobile* com os seus respectivos sistemas *back-end* é realizada através dos conhecidos serviços web ou web services, que são utilizados em procedimentos de transferência de dados por meio de protocolos de comunicação para diferentes plataformas de diferentes linguagens. A utilização dos web services apresenta variadas vantagens, como (DEITEL *et al.*, 2018):

- Integração de informação e sistemas: a comunicação entre os sistemas é bem simples, pois precisa somente de tecnologias como XML/JSON (*eXtensible Markup Language/JavaScript Object Notation*, em

português: Linguagem de Marcação Extensível/Notação de Objeto *JavaScript*) e protocolos HTTP (*Hypertext Transfer Protocol*, em português: Protocolo de Transferência de Hipertexto).

- Reutilização de código: o código desenvolvido para os web services pode ser utilizado por diferentes plataformas e com diferentes regras de negócios.
- Diminuição do tempo de desenvolvimento: possibilita a utilização de diversos frameworks, o que permite que o desenvolvedor não inicie do zero a sua aplicação.
- Mais segurança: não realiza comunicação direta com a base de dados, garantindo a segurança dos dados armazenados.
- Redução dos custos: não há necessidade de aplicações para integração de dados.

Os web services têm como característica ser uma solução prática, segura e com custo reduzido para as empresas realizarem a integração entre as suas diversas aplicações, apresentando uma solução para problemas de incompatibilidade entre essas aplicações.

VIDEOAULA: VISÃO GERAL DE BACK-END MOBILE: ARMAZENAMENTO DE DADOS, SQLITE, ARQUIVOS, WEB SERVICES

Serão apresentados exemplos de diversos frameworks de desenvolvimento de *back-end mobile* e serão abordados os frameworks de desenvolvimento de aplicação de banco dados *mobile*. Dessa forma, o aluno terá conhecimento tanto dos principais conceitos do desenvolvimento *back-end mobile* e de exemplos de frameworks que facilitam a implementação das aplicações.

Videoaula: Visão geral de *back-end mobile*: armazenamento de dados, SQLite, arquivos, web services

Para visualizar o objeto, acesse seu material digital.

APLICAÇÃO DO FRONT-END AO BACK-END USANDO SQLITE

Toda aplicação *mobile* precisa armazenar suas informações a fim de que tenha realmente alguma utilidade. Para a maioria desses aplicativos, não faria muito sentido se as informações fossem disponibilizadas de forma temporária, somente enquanto o usuário os estivesse utilizando e, depois disso, elas fossem apagadas. Com isso, para que se consiga realizar o armazenamento dessas informações passadas pelos usuários, precisa-se ter uma interligação entre as telas que os usuários utilizam, o desenvolvimento *front-end* do aplicativo e o desenvolvimento *back-end*, que tem como uma de suas funções o armazenamento dessas informações. Esse tipo de tarefa deve ser realizada com linguagens específicas de gerenciamento de bancos de dados, como a SQLite (ALVES, 2020).

Um exemplo

Para o desenvolvimento *mobile* utilizando o banco de dados SQLite, existem diversas APIs dentro de cada plataforma *mobile*, as quais facilitam essa integração.

Na plataforma Android, existe a API *android.database.sqlite*, que possui uma classe auxiliar denominada *SQLiteOpenHelper*, a qual gerencia a criação e o gerenciamento de versões do banco de dados para a aplicação (AKOWUAH; AHLAWAT; DU, 2018).

A *SQLiteOpenHelper* apresenta diversos métodos que facilitam essa interação com o banco de dados. A seguir temos os principais métodos dessa classe e uma breve explicação de suas funcionalidades.

- *close()*: fecha o objeto (instância) criado para o banco de dados SQLite.
- *getDatabaseName()*: retorna o nome do banco de dados SQLite que está aberto.
- *getReadableDatabase()*: permite criar e/ou abrir um banco de dados SQLite.
- *getWritableDatabase()*: permite criar e/ou abrir um banco de dados SQLite que pode ser utilizado para a leitura e escrita de informações.
- *setLookasideConfig()*: permite configurar a alocação de memória para o banco de dados.
- *setOpenParams()*: permite configurar parâmetros específicos antes de iniciar o banco de dados.

Sintaxe e definição de alguns parâmetros

A seguir, temos a sintaxe de um dos construtores da classe *SQLiteOpenHelper* juntamente com a definição dos seus parâmetros.

Sintaxe: *public SQLiteOpenHelper(Context, String name, int version, SQLiteDatabase.OpenParams openParams)*

Definição dos parâmetros:

- *context*: especifica o caminho para o banco de dados. Precisa ser iniciado com *null* (nulo).
- *name*: especifica o nome do banco de dados. Precisa ser iniciado com *null*.
- *version*: define qual versão do banco de dados será utilizada.
- *openParams*: definir as *flags* que serão utilizadas na criação do banco de dados. Não pode ser *null*.

Exemplos de *flags*:

- *CREATE_IF_NECESSARY*: cria o banco de dados caso ele não exista.
- *ENABLE_WRITE_AHEAD_LOGGING*: habilita o *log* de registro de escrita antecipada para o banco de dados.
- *OPEN_READONLY*: abre o banco de dados somente para leitura.
- *OPEN_READWRITE*: abre o banco de dados para leitura e escrita.
- *NO_LOCALIZED_COLLATORS*: inicia o banco de dados sem ser definida a forma de ordenação das tabelas.

A definição das *flags* no momento de criação do banco de dados, em vez de realizar a chamada de um método específico para executar a mesma função, geralmente leva a um desempenho melhor para a execução do banco de dados pelo programa desenvolvido.

A classe *SQLiteOpenHelper()* possui outros construtores com diferentes parâmetros que facilitam a interação do desenvolvedor com o banco de dados.

VIDEOAULA: APLICAÇÃO DO FRONT-END AO BACK-END USANDO SQLITE

A classe *SQLiteOpenHelper()* apresenta diversos outros métodos para auxiliar na criação, atualização, remoção e consulta de informações no banco de dados. Esses métodos desempenham a função das *callbacks* para facilitar a utilização dos bancos de dados pelo desenvolvedor (SALVAR..., 2022). A seguir, apresentam-se essas *callbacks* com uma explicação das suas funcionalidades.

- *onConfigure()*: chamada quando se está realizando a configuração do banco de dados para que sejam habilitados os recursos de registro de escrita antecipada e de geração de chave estrangeira (*foreign key*).
- *onCreate()*: chamada quando o banco de dados é criado pela primeira vez.
- *onOpen()*: chamada quando o banco de dados está sendo aberto.
- *onDowngrade()*: chamada quando o banco de dados precisar passar por um *downgrade* (desatualização) para funcionar.
- *onUpgrade()*: chamada quando o banco de dados precisa passar por um *upgrade* (atualização) para funcionar.

Videoaula: Aplicação do *front-end* ao *back-end* usando *SQLite*

Para visualizar o objeto, acesse seu material digital.

APLICAÇÃO DO FRONT-END AO BACK-END USANDO WEB SERVICES

A *World Wide Consortium* (W3C) define *web services* como uma arquitetura de integração de diferentes sistemas que podem ou não utilizar a mesma plataforma de desenvolvimento. Essa arquitetura possibilita essa integração utilizando sempre uma rede, como a internet, e está sempre disponível para o uso.

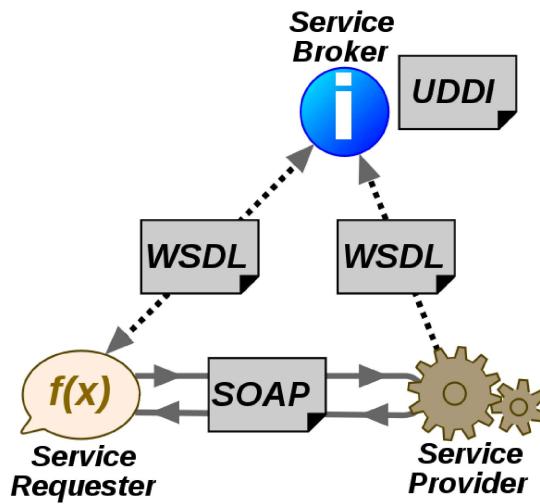
Os *web services* utilizam dois modelos de protocolos de comunicação: o *SOAP* (*Simple Object Access Protocol*) e o *REST* (*Representational State Transfer*).

Para que os *web services* possam estar disponíveis para a utilização, eles devem apresentar os seguintes componentes:

- A ferramenta *Universal Description, Discovery e Integration* (UDDI).
- *Web Services Description Language* (WSDL).
- Um dos protocolos SOAP ou REST (ROVAI *et al.*, 2018).

A Figura 2 apresenta uma ilustração de como os protocolos e componentes se comunicam para gerar os web services. O *Service Resquester* é o aplicativo que solicita o serviço; o *Service Provider* é o que fornece o serviço; e o *Service Broker* é aquele que fornece suporte, por exemplo, aos aplicativos de acesso aos serviços de banco de dados.

Figura 2 | Comunicação entre o protocolo SOAP e os componentes UDDI e WSDL.



Fonte: Wikimedia Commons.

Web services

Considerando o desenvolvimento *back-end* e *front-end*, web services são considerados programas *back-end* que podem facilitar a publicação, a distribuição e o processamento das informações na internet.

Os componentes web services são padronizados para qualquer plataforma de desenvolvimento *mobile*, mas esses serviços possuem características específicas para cada uma das plataformas. Para a Android, por exemplo, as principais características comportamentais dos web services são:

- Têm como base o XML: utilizam essa linguagem para representar e transportar dados pela rede (internet).
- São fracamente acoplados: os serviços de provedores e os serviços de clientes não estão vinculados um ou outro.
- Podem ser síncronos ou assíncronos: nos serviços síncronos, o cliente do serviço está vinculado à execução dos serviços e, nos serviços assíncronos, o cliente pode solicitar o serviço primeiro e depois executar outras funções.
- Têm suporte para chamadas de procedimentos remotos: permite que o cliente do serviço realize acesso a algumas funções, métodos e procedimentos de forma remota, ou seja, por uma rede.
- Apresentam suporte para troca de documentos: o XML representa as informações de forma genérica, permitindo a representação dos documentos de diversas formas.

Um exemplo

A construção de web services utilizando a plataforma de desenvolvimento *mobile* Android apresenta algumas APIs que auxiliam na construção dos web services.

Uma dessas APIs é a *Retrofit*, cuja principal função é realizar a integração entre a aplicação *mobile* e os serviços disponibilizados no *back-end*, que possuem como padrão de comunicação o protocolo *RESTful*.

Uma das principais vantagens de se utilizar a API *Retrofit* é sua codificação simples, que se baseia na utilização de anotações, na assinatura dos métodos, que são passadas de acordo com o método HTML a ser utilizado para realizar a requisição do serviço (MOZER *et al.*, 2014). Essas assinaturas são:

- @GET ("url"): solicita dados para os web services no endereço de web (url – Uniform Resource Locator).
- @POST("url"): especifica o caminho das informações para os web services do url.

- @PUT("url"): envia dados para os web services do url.
- @DELETE("url"): apaga dados dos web services do url.

A utilização dos web services para realizar a interligação dos serviços *back-ends* é de grande importância para um bom desempenho dos aplicativos *mobile*.

VIDEOAULA: APLICAÇÃO DO FRONT-END AO BACK-END USANDO WEB SERVICES

Para realizar requisições de serviços HTTP aos web services utilizando a plataforma de desenvolvimento *mobile*, existem outras duas classes nativas que realizam essa tarefa além da API *Retrofit*:

- *AsyncTask*: permite realizar as requisições HTTP.
- *HttpURLConnection*: permite realizar a conexão com um endereço de web – url.

Tanto a *Retrofit* quanto a *AsyncTask* possuem suas vantagens e desvantagens dependendo das funcionalidades de cada aplicação desenvolvida. Vale a pena se aprofundar em cada uma delas!

Videoaula: Aplicação do front-end ao back-end usando web services

Para visualizar o objeto, acesse seu material digital.

ESTUDO DE CASO

Você é o líder técnico da área de desenvolvimento mobile da empresa MobileApps, que tem como plataforma de desenvolvimento a linguagem Android Java.

Nesta semana, em uma reunião com um novo cliente, foi solicitado que sua equipe desenvolvesse um aplicativo mobile para processar informações que seriam acessadas de uma aplicação que disponibiliza dados financeiros e está disponível na web, e o resultado desse processamento teria que ser armazenado em uma base de dados criada especificamente para esse novo aplicativo mobile.

Como você é o líder técnico responsável pelo projeto, terá que detalhar os requisitos para o desenvolvimento do aplicativo para a sua equipe. Com isso, especifique quais plataformas, softwares e serviços você utilizaria para a concepção desse produto.

RESOLUÇÃO DO ESTUDO DE CASO

Para solucionar o problema proposto, vamos fazer algumas considerações sobre as funcionalidades do aplicativo mobile a ser desenvolvido:

- Ele terá que acessar informações disponíveis em outros serviços na web.
- Terá que processar essas informações e armazená-las em um banco de dados.
- Essas informações do banco de dados precisam ser apresentadas de alguma forma para o usuário final.

Com base nas considerações apresentadas, uma solução possível para descrever os requisitos de softwares para desenvolver o aplicativo seria:

- Desenvolver web services utilizando a plataforma Android, disponível na empresa.
- Criar um banco de dados SQLite, utilizando a classe *SQLiteOpenHelper* da Android.
- Criar uma tela de apresentação dos resultados para o usuário utilizando a programação *front-end* da linguagem Android.

Resolução do Estudo de Caso

Para visualizar o objeto, acesse seu material digital.

6º Saiba mais

Você pode se aprofundar um pouco mais nos conceitos e na utilização da classe *SQLiteOpenHelper()* em:

<https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper>.

Alguns exemplos de como utilizar as *callbacks* do SQLite podem ser encontrados em:

<https://developer.android.com/training/data-storage/sqlite>.

Informações mais detalhadas sobre as *flags* utilizadas por alguns parâmetros da classe *SQLiteOpenHelper()* podem ser encontradas em:

https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase#CREATE_IF_NECESSARY.

REFERÊNCIAS

4 minutos

Aula 1

ACID. In: WIKIPEDIA: a enclopédia livre. [S. I.]: Wikimedia Foundation, 2021. Disponível em:

<https://pt.wikipedia.org/wiki/ACID>. Acesso em: 5 fev. 2022.

BAUER, C.; KING, G. **Java Persistence com Hibernate**. Rio de Janeiro: Editora Ciência Moderna, 2007.

KING, G. et al. **Documentação de Referência Hibernate**: HIBERNATE – Persistência Relacional para Java Idiomático. [S. I.: s. n., s. d.]. Disponível em: https://docs.jboss.org/hibernate/orm/3.5/reference/pt-BR/pdf/hibernate_reference.pdf. Acesso em: 5 fev. 2022.

PLAIN Old Java Objects. In: WIKIPEDIA: a enclopédia livre. [S. I.]: Wikimedia Foundation, 2020. Disponível em:

https://pt.wikipedia.org/wiki/Plain_Old_Java_Objects. Acesso em: 5 fev. 2022.

Aula 2

BAUER, C.; KING, G. **Java Persistence com Hibernate**. Rio de Janeiro: Editora Ciência Moderna, 2007.

SQL Cheat Sheet. **SQL Tutorial**, [S. I.], c2021. Disponível em: <https://www.sqltutorial.org/sql-cheat-sheet/>. Acesso em: 5 fev. 2022.

Aula 3

DEITEL, H.; DEITEL, P.; DEITEL, A. **Android**. Porto Alegre: Grupo A, 2015. Disponível em:

<https://integrada.minhabiblioteca.com.br/#/books/9788582603482/>. Acesso em: 30 dez. 2021.

DEITEL, P.; DEITEL, H.; WALD, A. **Android 6 para programadores**. Porto Alegre: Grupo A, 2016. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788582604120/>. Acesso em: 30 dez. 2021.

DEVELOPERS. Conheça o Android Studio. **Developers**, [S. I.], 17 maio 2021. Documentação Android para desenvolvedores. Disponível em: <https://developer.android.com/studio/intro?hl=pt-br>. Acesso em: 29 dez. 2021.

ENTENDA o ciclo de vida da atividade. **Developers**, [S. I.], 23 jun. 2020. Disponível em: <https://developer.android.com/guide/components/activities/activity-lifecycle?hl=pt-br>. Acesso em: 6 fev. 2022.

Aula 4

ALVES, W. P. **Banco de Dados**: teoria e desenvolvimento. São Paulo: Editora Saraiva, 2020. Disponível em:

<https://integrada.minhabiblioteca.com.br/#/books/9788536533759/>. Acesso em: 9 jan. 2022.

AKOWUAH, F.; AHLAWAT, A.; DU, W. Protecting Sensitive Data in Android SQLite Databases Using TrustZone. In: INTERNATIONAL CONFERENCE ON SECURITY AND MANAGEMENT (SAM), 16., 2018, Las Vegas. **Proceedings** [...]. Athens: [s. n.], 2018. p. 227-233.

DEITEL, H.; DEITEL, P.; DEITEL, A. **Android**. Porto Alegre: Grupo A, 2015. Disponível em:
<https://integrada.minhabiblioteca.com.br/#/books/9788582603482/>. Acesso em: 8 jan. 2022.

DEITEL, P.; DEITEL, H.; WALD, A. **Android 6 para programadores**. Porto Alegre: Grupo A, 2016. Disponível em:
<https://integrada.minhabiblioteca.com.br/#/books/9788582604120/>. Acesso em: 8 jan. 2022.

MOZER, M. *et al.* **Sistemas Web**. Londrina: Educacional S. A., 2014. 192 p. Disponível em: <https://biblioteca-virtual-cms-serverless-prd.s3.us-east-1.amazonaws.com/ebook/788-sistemas-web.pdf>. Acesso em: 8 jan. 2022.

ORACLE. **Oracle Database Lite**: SQLite Mobile Client Guide. Redwood City: Oracle, 2010. Disponível em:
https://docs.oracle.com/cd/E12095_01/doc.10303/e16214/soverview.htm#BABHEJJL. Acesso em: 9 jan. 2022.

ROVAI, K. R. *et al.* **Tecnologias para web e para aplicativos móveis**. Londrina: Educacional S. A., 2018. 192 p. Disponível em: <https://biblioteca-virtual-cms-serverless-prd.s3.us-east-1.amazonaws.com/ebook/815-tecnologias-para-web-e-para-dispositivos-moveis.pdf>. Acesso em: 8 jan. 2022.

SALVAR dados usando o SQLite. **Developers**, [S. I.], 28 jan. 2022. Disponível em:
<https://developer.android.com/training/data-storage/sqlite>. Acesso em: 6 fev. 2022.

SQLITEDATABASE. **Developers**, [S. I.], 24 jan. 2021. Disponível em:
https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase#CREATE_IF_NECESSARY. Acesso em: 9 jan. 2022.

SQLITEOPENHELPER. **Developers**, [S. I.], 24 fev. 2021. Disponível em:
<https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper>. Acesso em: 9 jan. 2022.