Diagrama

O objetivo do Diagrama de dados é descrever a estrutura das classes, com seus atributos e relacionamentos (Figura 1). A criação e edição das classes do diagrama podem ser feitas a partir de recursos visuais, como *drag-and-drop*, permitindo em seguida, o uso de ferramentas para gerar de forma automática as camada de persistência e páginas CRUDs.

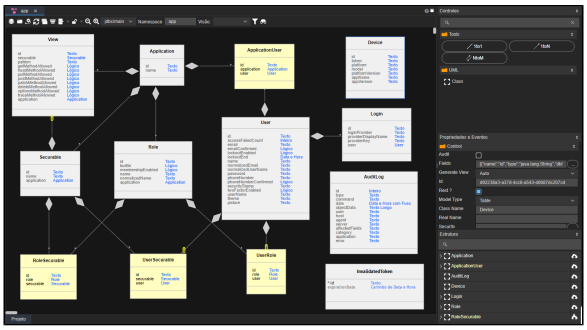

Figura 1 - *Layout* do diagrama

Acesso ao Editor

No Cronapp, os diagramas de dados ficam no diretório **Diagramas de Dados** (Localização: `Diagramas de dados/`) e, por padrão, todos os projetos possuem o diagrama "app", porém, outros podem ser criados no mesmo diretório. Ao expandir o arquivo do diagrama na árvore de arquivo (destaque 1 da figura 1.1), tanto as classes quanto os atributos dessas classes serão exibidos. Ao clicar duas vezes sobre uma classe ou atributo, a janela de configuração da classe ou atributo é exibida dentro do Editor do Diagrama (figuras 4.1 e 4.2).

O menu de contexto "Ação" (destaque 2 da figura 1.1) das classes e seus atributos permitem gerar páginas CRUDs com o Assistente de CRUD ou gerar um relatório (veja mais detalhes no tópico "Criar relatórios" da documentação Relatório).

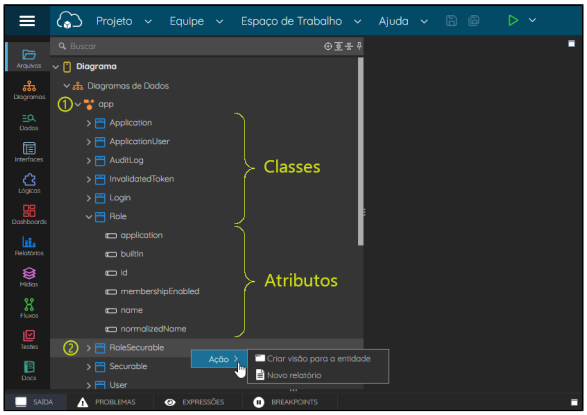

Figura 1.1 - Hierarquia de um arquivo de diagrama de dados

Classes

As classes que são geradas junto aos projetos Cronapp contemplam as funcionalidades de permissão de segurança, *log* de auditoria, sistemas de multi aplicações e registros dos dispositivos móveis. Para mais detalhes, acesse os links da coluna funcionalidades da tabela abaixo.

Nome da Entidade	Descrição	Funcionalidades

Nesta Página

Conteúdo complementar

Assista sobre o tema no Cronapp Academy

Caso seja seu primeiro acesso ao Cronapp Academy, crie antes uma conta gratuita e matricule-se no curso abaixo.

- Aula: Diagrama de dados

User	Usuários.	Permissão de Segurança
Role	Grupos (função).	
Login	*Logins* de provedores externos para um usuário.	
Securable	Permissionáveis.	
View	Páginas com restrições de acesso.	
RoleSecurable	Associa grupos e permissionáveis.	
UserSecurable	Associa usuários e permissionáveis.	
UserRole	Associa usuários e grupos (função).	
Application	Utilizado em sistemas com multi aplicações.	Multi Aplicações
ApplicationUser	Associa usuários e aplicações.	
AuditLog	Registra informações de eventos ocorridos no sistema.	Log de Auditoria
Device	Registra dispositivos que acessaram o sistema mobile.	Dispositivos mobile
InvalidatedToken	Registra as informações dos tokens de usuários após efetuar *logout* da aplicação.	Invalidação de tokens

Classe Device

Tabela responsável por armazenar informações dos dispositivos móveis que acessaram o sistema mobile.

Coluna do Banco	Tipo	Função
id	Texto	Chave primária.
token	Texto	Token do dispositivo.
platform	Texto	Plataforma do dispositivo.
model	Texto	Modelo do dispositivo.
platformVersion	Texto	Versão da plataforma.
appName	Texto	Nome da aplicação que acessou a base de dados.
appVersion	Texto	Versão da aplicação que acessou a base de dados.

Classe AuditLog

Tabela responsável por armazenar informações de eventos ocorridos no sistema, gerando um histórico dessas alterações. Para mais detalhes, consulte a documentação Log de Auditoria.

Coluna do Banco	Tipo	Função
id	Inteiro	Identificador numérico do registro.
type	Texto	Recurso que gerou aceso. Ex: app.entity.Entity, blocky.FolhaPagamento
command	Texto	Comando utilizado. Ex: UPDATE, DELETE.
date	Data e Hora	Data e hora em que ocorreu o evento.
objectData	Texto Longo	Objeto acessado.
user	Texto	Chave estrangeira da tabela User.
host	Texto	Endereço que gerou o acesso.

agent	Texto	Browser utilizado.
server	Texto	Endereço do servidor.
affectedFields	Texto	Campos modificados.
category	Texto	Recurso que gerou o registro. Ex: Authorization, Blockly ou DataSource
application	Texto	Chave estrangeira da tabela Application.
error	Texto	Log de erro, caso ocorra.

Classe InvalidatedToken

Tabela responsável por armazenar informações dos tokens de usuários que realizaram *logout* do sistema. Para mais detalhes, consulte a documentação Invalidação de tokens.

Coluna do Banco	Tipo	Função
id	Texto	Identificador do token do usuário.
expirationDate	Carimbo de Data e Hora	Data de expiração do token.

Menu superior

Nesse menu (Figura 2) se encontra todas as propriedades e ajustes do diagrama em si, como gerar a persistência, trocar de banco, filtros das classes, visões, etc.

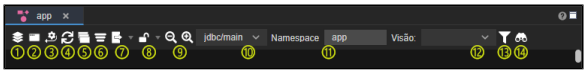

Figura 2 - Menu superior do diagrama de classes

1. **Gerar camada de persistência:** responsável por gerar as camadas **entity** e **dao** (ver mais detalhes no tópico "Java" em Estrutura de arquivos).
2. **Assistente de view para o diagrama:** cria os formulários CRUD na camada **view** nas classes do diagrama.
3. **Gerar diagrama a partir do banco de dados:** faz engenharia reversa a partir de um banco de dados cadastrado no projeto. Essa opção só fica disponível ao criar um segundo diagrama no projeto.
4. **Sincronizar Views**: realiza engenharia reversa nas tabelas virtuais do banco de dados.
5. **Organizar elementos do diagrama:** organiza toda a visualização das classes do diagrama.
6. **Detectar relacionamentos:** sinaliza as classes geradas em relacionamentos **NtoM.**
7. **Exportar diagrama em:** exporta o diagrama em dois formatos diferentes:
    - SVG
    - PDF
8. **Bloquear/Desbloquear Diagrama:** impede alterações no diagrama.
9. **Zoom:** ferramenta para aumentar ou diminuir zoom da área do diagrama.
10. **Banco de dados:** selecionar o banco de dados que será trabalhado, já que algumas aplicações podem ter mais de um banco.
11. **Namespace:** nome do espaço de trabalho
12. **Visão:** escolher a visualização do diagrama a partir da visão selecionada, como por exemplo visão "produto", mostrará apenas as classes, relacionamentos que tem interação direta com produto.
13. **Filtro por visão:** filtrar as classes e relacionamentos que serão mostrados na visão, cria novas visões.
14. **Buscar**: buscar controle, ferramenta.

Menu lateral

Nesse menu se encontram as ferramentas das classes, assim como seus aspectos (Figura 3).

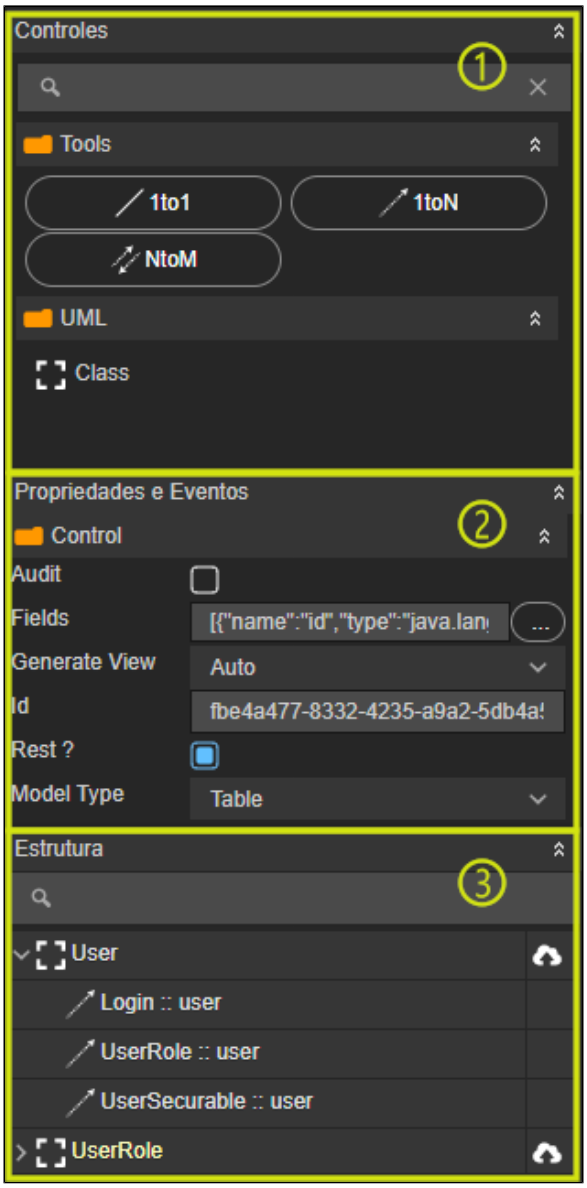

Figura 3 - Menu lateral

1. **Controle**: onde as ferramentas de relacionamentos e criação de novas classes se encontram.
    - **Tools:** ferramentas de relacionamento, para adicionar o relacionamento entre classes selecione o relacionamento, em seguida as classes que se relacionarão:
        - **1to1:** cada uma das duas entidades envolvidas referência obrigatoriamente apenas uma unidade da outra.
        - **1toN:** uma das entidades envolvidas pode referenciar várias unidades da outra, porém, do outro lado cada uma das várias unidades referenciadas só pode estar ligada uma unidade da outra entidade.
        - **NtoM:** neste tipo de relacionamento cada entidade, de ambos os lados, pode referenciar múltiplas unidades da outra.
    - **Class:** adiciona uma classe ao diagrama, basta arrastá-la para área do diagrama.
2. **Propriedades e eventos**: ao selecionar uma classe do diagrama ou ligação entre duas classes, os campos do **control** abrirá algumas propriedades que podem ser alteradas, assim como eventos.
3. **Estrutura**: exibe as classes existentes e seus relacionamentos com outras classes. Clique em uma classe para centralizá-la no diagrama, útil em grandes diagramas.

Estrutura da Classe

Ao criar uma classe você pode adicionar um nome e atribuir atributos. A manipulação da classe apresenta suas funcionalidades (Figura 4.1), como adição, remoção e edição de atributos, auditória em *log*; além disso, ela também apresenta as principais configurações de um atributo, como tipo, nome na coluna do banco, nome no formulário, se permite nulo ou não etc.

É possível adicionar atributos de armazenamento em quatro locais diferentes: Banco de Dados (não recomendado, confira o Manual de Boas Práticas), Dropbox, S3 (Amazon) ou Serviços de Cloud.

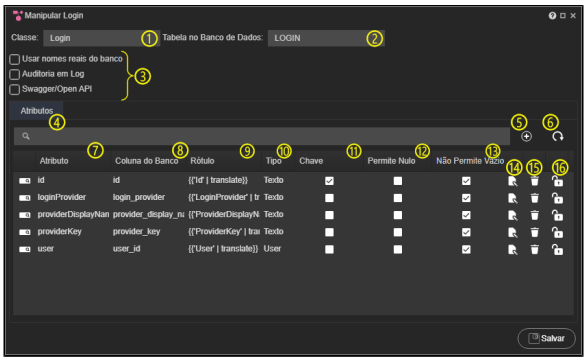

Figura 4.1 - Manipulação de classe

1. **Classe**: nome que você escolherá para a classe.
2. **Database Table:** nome que você escolherá para a tabela no banco de dados. Este campo só precisa ser usado se o nome da tabela tiver que ser diferente do nome da classe, caso contrário pode deixar em branco.
3. **Caixas de configuração:**
    - **Usar nomes reais do banco:** ao gerar a camada de persistência das classes criadas do Diagrama, o Cronapp sempre tenta converter os nomes das classes para o padrão UperCamelCase, atributos para o padrão lowerCamelCase e retirar caracteres como *underline* "_", com o objetivo de melhorar a legibilidade do código gerado. Porém, ao obter as classes a partir da engenharia reversa, pode ser necessário manter a mesma grafia das tabelas e colunas do banco de dados para permitir, por exemplo, reutilizar consultas SQL. Exemplo: `atributo_1, NomeDoAtributo, LIVROS.`
    - **Auditoria em Log:** permite a criação de log para essa tabela, veja mais informações na documentação.
    - **Swagger/Open API**: habilitar o recurso no Swagger nas entidades do diagrama de dados, veja mais informações na documentação.
4. **Pesquisar**: pesquisa o atributo pelo ID.
5. **Adicionar**: adiciona um novo atributo na classe.
6. **Atualizar**: recarrega os campos dos atributos exibidos na grade.
7. **Atributo:** campo para definir o nome do atributo na classe java. Quando esse é definido, automaticamente a **coluna do banco** e o **rótulo** recebem o mesmo nome.
8. **Coluna do Banco:** nome do campo no banco de dados.
9. **Rótulo:** rótulo do *front-end*. Nome da coluna que será exibida no CRUD gerado pelo Assistente de View para o Diagrama.
10. **Tipo:** caixa de seleção para escolher o tipo do atributo, se ela será uma string, um inteiro, um date.
11. **Chave**: define o campo como chave primaria da tabela, as chaves nunca se repetem na mesma tabela e, desta forma, podem ser usadas como um índice de referência para criar relacionamentos com as demais tabelas do banco de dados.
12. **Permite nulo:** define se o campo permitirá valores nulos. Um campo nulo é um valor ausente ou indefinido em um banco de dados. Isso significa que não há nenhum valor atribuído ao campo.
13. **Não Permite Vazio**: define se o campo permitirá valores vazios. Um campo vazio é um campo que contém uma string sem caracteres. Essa funcionalidade aplica-se somente aos tipos de dados Texto e Texto Longo.
14. **Editar**: abre mais propriedades referentes ao atributo.
15. **Excluir**: apaga o atributo selecionado.
16. **Permissão:** ao clicar no ícone de um atributo você poderá alterar as permissões de CRUD e filtro para determinados perfis.

Ao clicar no botão de editar de um atributo, a janela de edição irá apresentar todas as configurações possíveis para o mesmo (Figura 4.2).

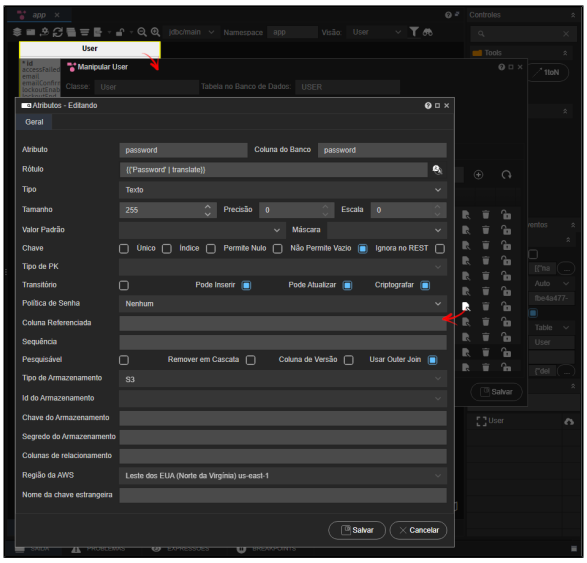

Figura 4.2 - Edição do atributo

- **Atributo:** campo para definir o nome do atributo na classe Java. Quando esse é definido, automaticamente a **coluna do banco** e o **rótulo** recebem o mesmo nome.
- **Coluna do Banco:** nome da coluna na tabela do banco de dados. (veja aqui como criar relacionamentos entre classes com chaves compostas).
- **Rótulo:** rótulo do *front-end*. Nome da coluna que será exibida no CRUD gerado pelo **Assistent e de View para o Diagrama**.
- **Tipo:** caixa de seleção para escolher o tipo do atributo, por exemplo, *string*, inteiro, *date*, *array de byte* e outros.
- **Tamanho:** tamanho do campo no banco de dados. Funciona de acordo com o tipo do campo adotado. Em campos do tipo *string* (*varchar*), essa coluna define a quantidade máxima de caracteres.

  Para deixar o atributo sem limites de caracteres, configure-o com o valor **0**.

- **Precisão:** quantidade total de algarismos aceitos em um campo do tipo numérico fracionado, contando a parte inteira e a parte fracionada.
- **Escala:** quantidade de algarismos que será configurada para a parte fracionada no número.

  As colunas ***Precisão e Escala*** somente são habilitadas à escrita quando o tipo do atributo escolhido na coluna ***Tipo*** for correspondente a um tipo numérico fracionado. Ex. *Double*, *Long* e tc.
- **Valor padrão:** permite selecionar uma das opções de valor padrão ("Nova GUID", "Data Atual" ou "Data Atual (Milissegundos)"). Esse campo também aceita expressões Java.
- **Máscara:** campo será configurado para ter a entrada rotulada com a expressão de máscara definida. Existem algumas máscaras padrões ou você poderá adicionar uma específica. **Ex:** uma máscara de telefone celular (99) 9 9999-9999.
- **Chave**: define o campo como chave primaria da tabela, as chaves nunca se repetem na mesma tabela e, desta forma, podem ser usadas como um índice de referência para criar relacionamentos com as demais tabelas do banco de dados.
- **Único:** marcando essa opção, o valor do campo será único na tabela.
- **Índice**: aplica uma referência associada a um atributo, é utilizada para fins de otimização de desempenho em consultas com filtro de igualdade, permitindo uma localização mais rápida de um registro quando efetuada uma consulta.
- **Permite Nulo:** define se o campo permitirá valores nulos. Um campo nulo é um valor ausente ou indefinido em um banco de dados. Isso significa que não há nenhum valor atribuído ao campo.
- **Não Permite Vazio**: define se o campo permitirá valores vazios. Um campo vazio é um campo que contém uma string sem caracteres. Essa funcionalidade aplica-se somente aos tipos de dados Texto e Texto Longo.
- **Ignorar no REST:** define se o campo poderá ser ignorado, não preenchido.
- **Tipo de PK:** tipo do campo da chave primária. Contém opções de tratamento da chave primária. Podem ser selecionados os valores:
  - **Nenhum:** não configura nenhum tratamento especial para o campo de chave primária.
  - **Auto Incremental:** define a chave primária como um campo numérico sequencial e auto incremental que é controlado pelo banco de dados. O tipo do campo deverá ser selecionado como inteiro.
    Exemplos.: 1, 2, 3.

    - **Nova GUID:** define um conjunto de caracteres no padrão *Universally Unique Identifier* (UUID) gerados pelo servidor da aplicação, seu tipo deverá ser texto (java.lang.String (varchar)).
      Exemplo: "`1d4e5d7b-dca8-4a1d-90fd-72648cf5dc8d`".
- **Transitório:** normalmente usados em atributos que passam por constantes modificações durante a aplicação. Atributos transitórios não geram campos na tabela, pois não participam da persistência e seus valores nunca são armazenados no banco de dados. Podem ser utilizados para realizar cache de atributos derivados de outros atributos, sendo possível configurar o seu retorno de forma *high code* na classe `.java` correspondente. Nesse caso, talvez seja preciso habilitar a "Compatibilidade do Cronapp com a versão 1.0", veja como habilitar no tópico "Origem dos Dados" na documentação da Fonte de Dados.
  No entanto, esse recurso era usado em versões antigas do Cronapp, por isso é recomendável utilizar campos calculados diretamente na fonte de dados, em vez de um atributo transitório. Verifique o tópico "Campos calculados" em Fonte de Dados.
- **Pode Inserir:** define que o atributo pode ser inserido.
- **Pode Atualizar:** define que o atributo como editável.
- **Criptografar:** define que o atributo será criptografado ao ser salvo em banco. O método de criptografia usado é o bcrypt, que gera um hash Blowfsh para o valor informado. Normalmente esse recurso é utilizado em campos de senha.
- **Política de senha**: oferece dois tipos de políticas ao cadastrar senhas, são eles: **Nenhum** e **Co mpleto** Acesse o tópico "Autenticação e Segurança" da documentação Configurações do projeto para entender mais.
    - **Nenhum**: permite que o usuário cadastre uma senha sem obedecer qualquer tipo de condição prévia, como por exemplo: não delimita o número de caracteres, o usuário pode cadastrar a senha sem nenhum caractere especial ou letra maiúscula.
    - **Completo**: as configurações de senha feitas nas configurações do projeto serão aplicadas ao campo. Por padrão, esse campo obedece algumas condições, são elas: a senha deve ter 6 ou mais caracteres, deve conter 1 ou mais caracteres especiais e deve conter 1 ou mais letras maiúsculas. o limite máximo de caracteres permitido é 16. Ao selecionar este campo, será criado um atributo <“`nomeAtributo_history`”> na classe java e uma coluna, com o mesmo nome do atributo, na tabela relacionada a Classe. Objetivo é impedir que o usuário altere a sua senha para as duas últimas senhas utilizadas anteriormente.
- **Coluna Referenciada:** usado para informar manualmente o nome da coluna de chave primária referenciada por esta coluna de chave estrangeira. Esse campo não possui mais função em projetos novos, ainda é exibido apenas para manter compatibilidade em projetos antigos (Cronapp v1.0).
- **Sequência**: esse campo espera receber o identificador de uma sequência criada diretamente no banco de dados. Em banco de dados, "sequence" são objetos criados afim de controlarmos os valores das chaves primárias. Veja mais detalhes sobre sequências na documentação do seu banco de dados, exemplo: Oracle e SQL Server.
- **Pesquisável**: configura automaticamente o atributo como pesquisável na aplicação (campo Pesquisar da tela de CRUD).
- **Remover em Cascata**: remove dados em cascata vinculados àquele campo. Útil em relacionamentos 1 to N. **Ex:** Entidades **Pai** e **Filho**, apagando um registro Pai, todos os registros Filhos relacionados serão excluídos automaticamente. Esse campo deve ser marcado no atributo da chave estrangeira da entidade Filho.
- **Coluna de Versão:** versiona o objeto da entidade a partir das modificações que forem feitas em seus atributos.
- **Usar o Outer Join:** essa propriedade permite gerar consultas e filtros entre entidades no formato *outer join*, ao invés de *inner join*. Dessa forma, é possível realizar uma consulta em uma entidade de relacionamento M para N mesmo que não exista relacionamento de chave estrangeira em um dos relacionamentos "pai". Para funcionar, essa propriedade deve ser marcada nas caixas de seleção dos atributos de chave estrangeira e propriedade Filtros e Parâmetros da Fonte de dados deve ser configurada.
- **Tipo de Armazenamento**: selecione um dos locais de armazenamento na nuvem (Dropbox, S3 ou Serviços de Cloud). Esse campo ficará habilitado ao selecionar os Tipos "imagem no Cloud" ou "Arquivo no Cloud".
- **Id do Armazenamento**: utilizado apenas para Selecionar a URL do Serviço de Cloud. Esse campo ficará habilitado ao selecionar os Tipos "imagem no Cloud" ou "Arquivo no Cloud".
- **Chave do Armazenamento**: chave disponibilizada por uma das opções de armazenamento. Esse campo ficará habilitado ao selecionar os Tipos "imagem no Cloud" ou "Arquivo no Cloud".
- **Segredo do Armazenamento**: segredo disponibilizado por uma das opções de armazenamento. Esse campo ficará habilitado ao selecionar os Tipos "imagem no Cloud" ou "Arquivo no Cloud".
- **Colunas de relacionamento:** permite informar os campos físicos de ligação entre a tabela de origem e a tabela de destino, é possível informar múltiplos campos, separados por vírgula. Ex: `fk_pessoa_id=id, fk_cpf=cpf`. Caso o campo fique vazio, o sistema assumirá que a ligação é feita com a chave primária.
- **Região da AWS:** permite selecionar a região onde está a conta do Amazon S3 (AWS)**.** Esse campo ficará habilitado ao selecionar os Tipos "imagem no Cloud" ou "Arquivo no Cloud".
- **Nome da chave estrangeira**: Usado em atributos de FK (Foreign Key), ele altera o campo "Constraint_name" da respectiva coluna na tabela do Banco de dados.

Acesse o tópico "Upload de arquivos" em Arquivo para mais detalhes sobre como configurar os tipos de atributos **Arquivo** ou **Imagem**.

Chave composta

Usamos chaves compostas quando precisamos de dois campos para compor um identificador único para um registro. No exemplo abaixo criamos uma classe cuja chave primária é constituída de 2 campos. A classe "Tipo" contém dois atributos de chave primária, **id_tipo** e **ano_tipo**, e faz uma relação 1 para N com a classe "Cadastro", que possui o atributo de chave estrangeira **tipo**.

Inicialmente, após criar o relacionamento 1 (Tipo) para N (Cadastro) entre as classes, na janela **Manipular Cadastro** da classe Cadastro, o nome do atributo de chave estrangeira estará com o nome da classe relacionada (**tipo**) e na coluna "Coluna do Banco" estará como **fk_tipo**, para transformar essa chave estrangeira em uma chave estrangeira composta, é preciso adicionar as duas chaves primárias da primeira classe, separadas por ";" (ponto e vírgula), à coluna "Coluna do Banco" (destaque 1 da figura 4.3).

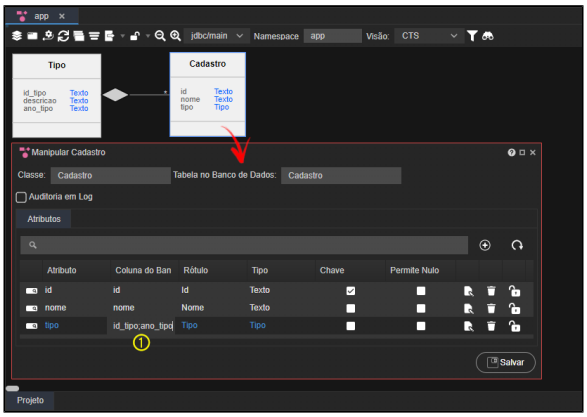

Figura 4.3 - Adição da chave composta

Importante

No diagrama, o relacionamento é feito entre as classes, e não entre os campos (atributos) como ocorre no banco de dados. Assim, no caso de chaves compostas é necessário informar manualmente os nomes das chaves no atributo de relação.

Tipos

A tabela abaixo apresenta a relação entre os tipos disponíveis no Cronapp e o que é gerado nas classes Java e no banco de dados.

Tipo Cronapp	Classe Java	Correspondente no Banco **	Descrição
Texto (string)	String	**Firebird, H2, MySQL, Oracle, Postgres, SQL Server**: VARCHAR.	Armazena um conjunto de caracteres.
Texto Longo (Long text)	String	**Firebird**: BLOB SUB_TYPE TEXT. **H2, MySQL, Postgres, SQL Server:** TEXT **Oracle**: CLOB.	Armazena um conjunto de caracteres com tamanho máximo de 4GB.
UUID	String	**Firebird, H2, MySQL, Oracle, Postgres, SQL Server:** VARCHAR. **PostgreSQL**: UUID (tipo nativo PostgreSQL).	Utilizado para tratar o tipo UUID do banco PostgreSQL durante a engenharia reversa. Ao criar uma tabela pelo Cronapp (camada de persistência) utilizando o tipo UUID, o tipo gerado para o campo em todos os bancos de dados será sempre Varchar. Além disso, ao selecionar essa opção, a propriedade **Tipo da PK** é automaticamente configurada com a opção "Nova GUID".
Lógico (Boolean)	Boolean	**Firebird:** SMALLINT.	Permite que o atributo armazene somente um de dois estados: true ou false.

		H2: BOOLEAN **MySQL:** TINYINT. **Oracle:** NUMBER. **Postgres:** BOOL. **SQL Server:** BIT.	
Caracter (Character)	Character	**Firebird:** VARCHAR. **H2, Oracle, MySQL, SQL Server:** CHAR. **Postgres**: BPCHAR.	Armazena somente um caractere alfanumérico.
Numérico (Numeric)	Double	**Firebird**: DOUBLE PRECISION. **H2, MySQL:** DOUBLE. **Oracle:** NUMBER. **Postgres, SQL Server**: FLOAT.	Armazena tanto números inteiros quanto fracionários no padrão da IEEE 754.
Inteiro (Integer)	Integer	**Firebird, H2, SQL Server**: INTEGER. **MySQL**: INT. **Oracle:** NUMBER. **Postgres:** INT4.	Armazena números inteiros entre -2,147,483,648 à 2,147,483,647.
Inteiro Longo (Long Integer)	Long	**Firebird, SQL Server:** NUMERIC. **H2, MySQL:** BIGINT. **Oracle:** NUMBER. **Postgres:** INT8.	Armazena números inteiros entre - 2^{63} à 2^{63} - 1.
Decimal Grande (Big Decimal)	BigDecimal	**Firebird, H2, Postgres, SQL Server**: NUMERIC. **MySQL:** DECIMAL. **Oracle:** NUMBER.	Armazena tanto números inteiros quanto fracionários.
Inteiro Grande (Big Integer)	BigInteger	**Firebird, H2, SQL Server**: NUMERIC. **MySQL:** BIGINT. **Oracle:** NUMBER. **Postgres**: INT8.	Armazena números inteiros que são maiores que Integer ou Long.
Inteiro Curto (Short)	Short	**Firebird, H2, MySQL, SQL Server**: SMALLINT. **Oracle:** NUMBER. **Postgres**: INT2.	Armazena números inteiros entre -32,768 à 32,767.
Byte	Byte	**Firebird, H2, SQL Server**: SMALLINT. **MySQL:** TINYINT. **Oracle:** NUMBER. **Postgres:** INT2.	Armazena números inteiros entre -32,768 à 32,767.
Data (Date)	Date	**Firebird, H2, MySQL, Oracle, Postgres**: DATE. **SQL Server**: DATETIME.	Armazena data (data, mês e ano).
Data e	Date		Armazena tanto a data quanto a hora.

Hora (Date and Time)		**Firebird, H2, Oracle, Postgres**: TIMESTAMP. **MySQL**: DATETIME. **SQL Server**: DATETIME2.	
Carimbo de Data e Hora (Timestamp)	Date	**Firebird, H2, MySQL, Oracle, Postgres**: TIMESTAMP. **SQL Server**: DATETIME2.	Armazena tanto a data quanto a hora.
Hora (Time)	Date	**Firebird, H2, MySQL, Postgres:** TIME. **Oracle:** TIMESTAMP. **SQL Server**: DATETIME.	Armazena hora (hora, minuto e segundo).
Binário (Binary)	*	**Firebird:** BLOB SUB_TYPE BINARY. **H2:** LONGVARNIBARY. **MySQL:** LONGBLOB. **Oracle:** BLOB. **Postgres:** BYTEA. **SQL Server:** BINARY.	Armazena o dado em formato binário.
Arquivo no Banco (Database File)	*	**Firebird**: BLOB SUB_TYPE BINARY. **H2:** LONGVARNIBARY. **MySQL**: LONGBLOB. **Oracle**: BLOB. **Postgres**: BYTEA. **SQL Server**: VARBINARY.	Armazena arquivos no banco de dados usado em seu projeto no Cronapp.
Arquivo no Cloud (Cloud File)	*	**Firebird, H2, MySQL, Oracle, Postgres, SQL Server**: VARCHAR.	Armazena arquivos na nuvem (Dropbox, S3 Amazon ou Serviços de Cloud) e salva em banco a sua URI.
Imagem no Banco (Database Image)	*	**Firebird**: BLOB SUB_TYPE BINARY. **H2:** LONGVARNIBARY **MySQL**: LONGBLOB. **Oracle**: BLOB. **Postgres**: BYTEA. **SQL Server**: IMAGE.	Armazena imagens no banco de dados usado em seu projeto no Cronapp.
Imagem no Cloud (Cloud Image)	*	**Firebird, H2, MySQL, Oracle, Postgres, SQL Server**: VARCHAR.	Armazena imagens na nuvem (Dropbox, S3 Amazon ou Serviços de Cloud) e salva em banco a sua URI.

Versão (Row Version)	Long	**Firebird, H2, MySQL, Oracle, Postgres**: VARCHAR. **SQL Server**: TIMESTAMP.	Atributo usado para fazer controle de atualização.
XML	Byte	**Firebird, H2, MySQL, Oracle, Postgres**: VARCHAR. **SQL Server**: XML.	Armazena documentos ou fragmentos XML
Classe (Chave estrangeira / Foreign Key)	-	Foreign Key	Apresenta a(s) classe(s) existentes no diagrama para gerar a chave estrangeira dessas tabelas.

***** Não é uma classe Java, mas sim um dado primitivo. Byte[] - um array de byte.

****** Os tipos correspondentes dos bancos de dados podem variar a depender da versão do Banco de dados, do driver jdbc utilizado ou se o atributo é uma chave primária (PK), em caso de dúvida, recomendamos verificar a documentação do seu banco de dados.

Valor Padrão

Esse campo permite definir uma regra para alimentar o valor do atributo selecionado sempre que o objeto for criado. Além das opções de **chave única (GUID)**, **Data Atual** e **Data Atual (Milissegundos)**, esse campo também aceita expressões Java.

Por exemplo, ao inserir a expressão "`new Random().nextInt(26)`", sempre que o objeto for iniciado, o atributo selecionado receberá um valor aleatório entre "0" e "25". É importante ficar atento ao campo **Tipo**, pois no exemplo da expressão randômica, a expressão Java retornará um inteiro, logo, será necessário selecionar um tipo compatível com o retorno da expressão ou opção.

Opção	Tipos aceitos	O que faz
Nova GUID	• Texto (string)	Gera um valor único e universal (*Universally Unique Identifier* (UUID)).
Data e Hora	• Data • Data e Hora • Carimbo de Data e Hora • Hora	Obtém a data e hora atual e alimenta o atributo com base no Tipo selecionado.
Data e Hora (Milissegundos)	• Inteiro longo • Versão	Retorna um valor no formato Inteiro longo com a data e hora atual em milissegundos. Essa opção foi incluída para atender o tipo "versão" do banco de dados SQL Server, que requer um *Timestamp* no formato de Inteiro longo.

Usar o Outer Join

Por padrão, a IDE utiliza o método de junção *Inner Join* ao relacionar 2 classes (M - N), com isso, se uma das chaves estrangeiras estiver nula, ao pesquisar, ele não irá aparecer no filtro, que exibirá apenas os registros que contenha todas as colunas de chave estrangeira preenchidas. Portanto, para exibir todos os registros do filtro, mesmo que algum relacionamento esteja nulo (formato *Outer Join)*, é necessário configurar os atributos de relacionamento e o filtro da fonte de dados no formulário.

No exemplo abaixo vamos criar um relacionamento M para N (Cliente - Venda - Produto) (Figura 5.1) e usar o filtro da tela de CRUD **Vendas**.

Atributos da classe

Para usar o Outer Join, é necessário abrir a janela de edição do atributo de chave estrangeira da classe de relacionamento (Venda) e ativar a caixa **Usar Outer Join** (Figura 5.1). No exemplo abaixo, ativamos a opção **Usar Outer Join** nos atributos **Comprador** e **produtoVendido** da classe **Venda**. Após isso, ger e a camada de persistência e crie as views das 3 classes.

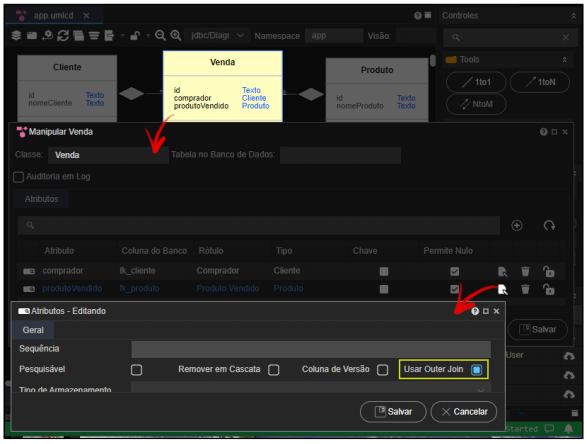

Figura 5.1 - Ativando a opção Usar Outer Join nos atributos de chave estrangeira

Filtros da view

Após isso, abra a view da classe de relacionamento (**Venda**), selecione a **Fonte de dados** principal e, em seguida, clique em "**...**" da propriedade **Filtros e Parâmetros**. Siga os passos da figura 5.2:

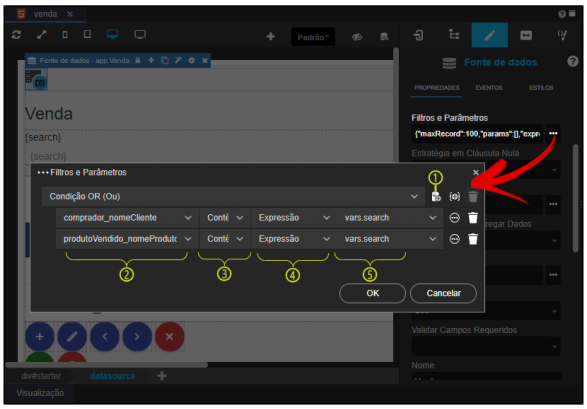

Figura 5.2 - Configurando a propriedade Filtros e Parâmetros da fonte de dados principal

1. Com a condição "OR" (Ou) selecionada no primeiro campo, clique no símbolo de página com **+** para adicionar uma condição;
2. Selecione os **atributos** da tabela relacionamento;
3. Escolha a opção **Contém**;
4. Selecione **Expressão**;
5. Por fim, defina o campo de pesquisa (**vars.search**) em todos os campos.

Resultado

Após executar o projeto, adicione alguns registros nas páginas **cliente** e **produto**. Em seguida, acesse a página **venda** para incluir registros de vendas, relacionando cliente e produtos. Por fim, digite algo no campo de pesquisa para exibir os registros de vendas, mesmo que um comprador não possua um produto relacionado (nulo) ou um produto relacionado não possua um comprador (nulo). Ao pesquisar por "ma" no exemplo da figura abaixo, o produto "Macarrão" foi exibido mesmo não possuindo um Comprador (cliente) relacionado.

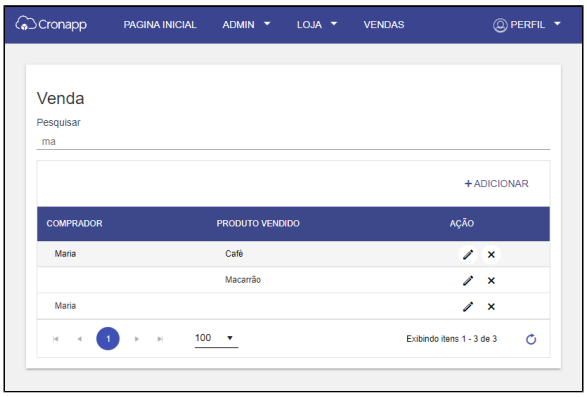

Figura 5.3 – Resultado do filtro com o *outer join*

Coluna de Versão

Essa propriedade permite definir um atributo da classe para trabalhar como versionador do registro, incrementando seu valor automaticamente a cada modificação do registro. Exemplo: Entidade **Empresa** possui os atributos **ID**, **nome** e **versao** (campo usado para versionar), sempre que o nome da empresa for alterado (Exemplo da figura abaixo, o valor "Cronapp" foi alterado para "Cronapp LowCode"), o atributo **versão** será **incrementado automaticamente**, informando quantas alterações foram feitas.

Requisitos para a coluna de versão:

1. Não pode ser **Chave.**
2. Deve possuir um dos seguintes tipos: **Inteiro, Inteiro Longo** ou **Carimbo de Data e Hora.**
3. Deve permitir **Inserção** e **atualização**.
4. Não pode ser **Nulo.**
5. Necessário informar um valor inicial para o campo configurado como coluna de versão.

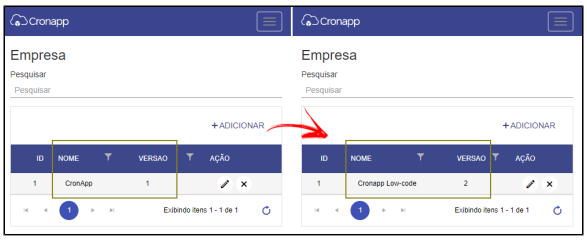

Figura 6 - Exemplo de incremento na coluna versionada após alterar algum campo no registro

Máscaras dos atributos

Abaixo estão listadas as máscaras pré-definidas e os tipos compatíveis da janela de atributos. O campo de máscara permite personalização. Para saber mais sobre máscaras, acesse a documentação: Formatação de máscaras na camada cliente.

Máscara	Tipos compatíveis
CPF	**texto** ou **texto longo**.
CNPJ	**texto** ou **texto longo**.
CEP	**texto** ou **texto longo**.
Telefone	**texto** ou **texto longo**.
Hora	**hora**, **data e hora** ou **carimbo de data e hora**.
Data	**data**, **data e hora** ou **carimbo de data e hora**.
Data e hora	**data**, **hora**, **data e hora** ou **carimbo de data e hora**.
Semana	**data**, **data e hora** ou **carimbo de data e hora**.

Mês	**data**, **data e hora** ou **carimbo de data e hora**.
Moeda	**texto**, **texto longo**, **inteiro** ou **numérico**.
Inteiro	**texto**, **texto longo**, **inteiro** ou **numérico**.
Numérico	**texto**, **texto longo**, **inteiro** ou **numérico**.

Menu de contexto da classe

Ao clicar com o botão direito em cima da classe, um menu aparecerá com algumas opções.

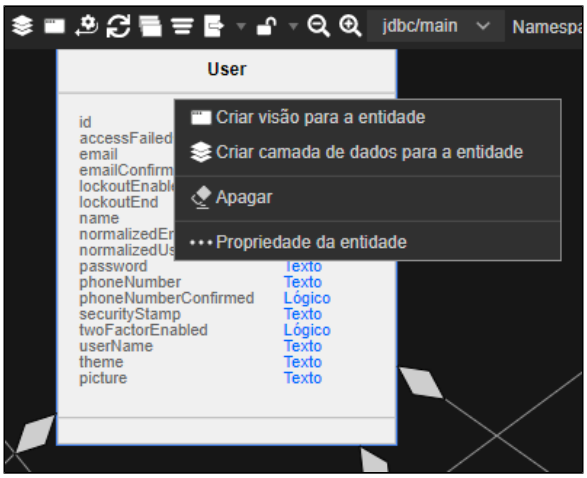

Figura 7 - Janela da classe

- **Gerar visão CRUD para a entidade:** somente gera o formulário da classe selecionada.
- **Criar camada de dados para a entidade:** somente gera a camada de persistência para a classe selecionada.
- **Apagar:** apaga a classe selecionada.
- **Propriedade da entidade:** abre a janela de manipulação da classe selecionada.

Exclusão do diagrama

Quando um diagrama é criado e é gerado sua camada de persistência, é criado um diretório (Namespace) com o nome do diagrama contendo as camadas **dao** e o **entity** (Endereço: `src/main/java/`)(1 da figura 8). o arquivo **persistence.xml** (Endereço: `src/main/java/META-INF/persistence.xml`) também é atualizado contendo o diagrama criado (2).

Para localizar o arquivo **persistence.xml** e os arquivo **entity** e **dao**, é necessário que o Modo Avançado esteja habilitado.

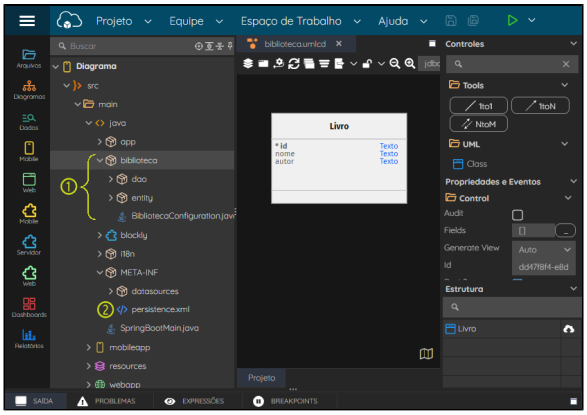

Figura 8 - Arquivos gerados após criar a camada de persistência

Após clicar para remover o diagrama, será exibido uma janela perguntando se também deseja excluir a camada de persistência junto ao diagrama, caso você clique em **OK**, o Namespace com os diretórios **dao** e **entity** serão excluídos e o arquivo **persistence.xml** será modificado.

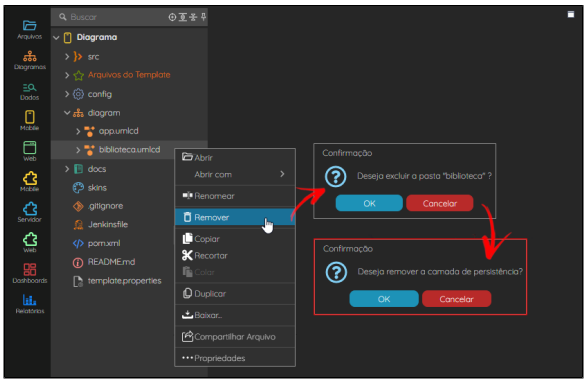

Figura 8.1 - Excluindo Diagrama e Camada de persistência

Após confirmar a ação de remover a camada de persistência (figura 8.1), o Namespace "`biblioteca`" será excluído (destaque 1 da figura 8) e o arquivo `persistence.xml` atualizado (destaque 2 da figura 8).

Organização dos relacionamentos

Relacionamento entre classes (destaque 1 da figura 9) podem ficar sobrepostos em cima de outras classes, caso o diagrama possua muitas classes, dificultando a visualização dos relacionamentos. Assim, é possível "quebrar" a linha para estruturar da maneira que preferir, como mostrado abaixo.

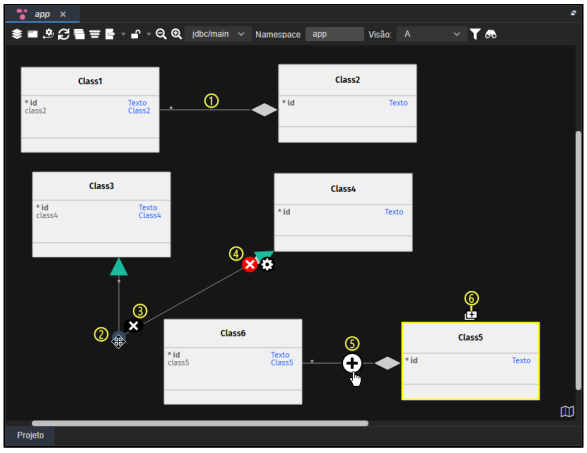

Figura 9 - Dobrando linhas do relacionamento

1. Relacionamento entre 2 classes são sempre exibidos com uma linha reta.
2. Para mudar o traçado, clique sobre a linha e puxe, gerando um círculo para posicionar a "quebra" onde preferir.
3. A cada ponto de quebra será exibido o ícone "X", ao clicar, a "quebra" será desfeita.
4. O ícone "X" vermelho permite excluir o relacionamento entre as classes.
5. Ao selecionar a ligação entre duas classes, também é possível clicar no círculo com o símbolo de "+" onde será exibido mais opções sobre a mesma.
6. O pequeno botão "+" acima da classe seleciona irá duplicar a classe.