

Base de Dados II

CTeSP – Técnico de Programação de Sistemas de Informação

Departamento de Informática

Sistema de Gestão de um Instituto de Ensino Superior



Figura 1 - Oracle Database.

Prova Realizada por: Gabriel Teixeira, João Moço e Tiago Jorge

Data: 02/01/2026

Índice

Índice	2
Índice de Figuras	3
Introdução.....	10
Estrutura do Projeto	11
Contexto do Negócio Modelado.....	11
Modelo Físico	11
Entidades ao Pormenor	13
Relacionamentos do Modelo Físico:	37
<i>Data Definition Language – DDL</i>	39
Estrutura do DDL.....	39
Inserção de Dados.....	85
Conclusão – Primeira Fase	86
Fase 2 - Sistema de Gestão de um Instituto de Ensino Superior.....	87
Mudanças de Paradigma	88
Mudanças no Modelo Físico	89
Entidades ao Pormenor - Mudanças	91
Relações do Modelo Físico - Mudanças.....	101
<i>Data Definition Language – DDL</i>	102
Inserção de Dados.....	123
Operações PL/SQL	124
Ordem de Operações Necessária para o Sistema Funcionar	124
Operações PL/SQL do Sistema.....	126
Cálculo do Espaço Físico que a Base de Dados Ocupa em Produção	172
Parâmetros de Estimativa do Espaço Ocupado	172
Parâmetros a Calcular	172
Conclusão – Segunda Fase	174
FIM.....	175

Índice de Figuras

Figura 1 - Oracle Database.....	1
Figura 2 - Modelo Físico.....	12
Figura 3 - Tabela "log".....	14
Figura 4 - Tabela "acao_log".....	14
Figura 5 - Tabela "entidade_log".....	15
Figura 6 - Tabela "tipo_contacto".....	15
Figura 7 - Tabela "tipo_departamento".....	15
Figura 8 - Tabela "estado_inscricao".....	16
Figura 9 - Tabela "estado_matricula".....	16
Figura 10 - Tabela "curso".....	17
Figura 11 - Tabela "tipo_curso".....	17
Figura 12 - Tabela "unidade_curricular".....	17
Figura 13 - Tabela "uc_curso".....	18
Figura 14 - Tabela "uc_departamento".....	18
Figura 15 - Tabela "uc_docente".....	18
Figura 16 - Tabela "turma".....	19
Figura 17 - Tabela "truno".....	19
Figura 18 - Tabela "turno_aulas".....	19
Figura 19 - Tabela "aula".....	20
Figura 20 - Tabela "tipo_aula".....	20
Figura 21 - Tabela "colaborador".....	21
Figura 22 - Tabela "docente".....	21
Figura 23 - Tabela "nao_docente".....	21
Figura 24 - Tabela "cargo".....	22
Figura 25 - Tabela "funcao_docente".....	22
Figura 26 - Tabela "departamento".....	23
Figura 27 - Tabela "gabinete_docente".....	23
Figura 28 - Tabela "horario_colaborador".....	23
Figura 29 - Tabela "estudante".....	24
Figura 30 - Tabela "contacto_estudante".....	24
Figura 31 - Tabela "matricula".....	25
Figura 32 - Tabela "inscricao".....	25
Figura 33 - Tabela "presenca".....	26
Figura 34 - Tabela "nota".....	26
Figura 35 - Tabela "estado_inscricao".....	27
Figura 36 - Tabela "estado_matricula".....	27
Figura 37 - Tabela "propina".....	28
Figura 38 - Tabela "parcela_propina".....	28

Instituto Politécnico de Coimbra
Instituto Superior de Engenharia de Coimbra

Figura 39 - Tabela "avaliacao".	29
Figura 40 - Tabela "tipo_avaliacao".	29
Figura 41 - Tabela "entrega".	30
Figura 42 - Tabela "estudante_entrega".	30
Figura 43 - Tabela "ficheiro_entrega".	30
Figura 44 - Tabela "recurso".	31
Figura 45 - Tabela "ficheiro_recurso".	31
Figura 46 - Tabela "polo".	32
Figura 47 - Tabela "edificio".	32
Figura 48 - Tabela "andar".	33
Figura 49 - Tabela "espaco".	33
Figura 50 - Tabela "tipo_espaco".	34
Figura 51 - Tabela "curso_area".	34
Figura 52 - Tabela "docente_aula".	35
Figura 53 - Tabela "turma_docente".	35
Figura 54 - Tabela "gabinete_docente".	35
Figura 55 - Tabela "estudante_entrega".	36
Figura 56 - Tabela "horario_colaborador".	36
Figura 57 - Tabela "uc_curso".	37
Figura 58 - Tabela "area".	37
Figura 59 - Drop nas Tabelas.	40
De referir ainda, que devido à extensão do script DDL (cerca de linhas), a Figura 60, representa uma pequena parte das tabelas. O restante script acompanha o relatório aquando da sua submissão.....	40
Figura 61 - Create e Alter Tabela "acao_log".	41
Figura 62 - Create e Alter da Tabela "andar".	42
Figura 63 - Create e Alter da Tabela "area".	42
Figura 64 - Create e Alter da Tabela "aula".	43
Figura 65 - Create e Alter da Tabela "avaliacao".	43
Figura 66 - Create e Alter da Tabela "cargo".	44
Figura 67 - Create e Alter da Tabela "colaborador".	44
Figura 68 - Create da Tabela "contacto_colaborador".	45
Figura 69 - Create e Alter da Tabela "contacto_estudante".	45
Figura 70 - Create e Alter da Tabela "curso".	46
Figura 71 - Create e Alter da Tabela "curso_area".	46
Figura 72 - Create e Alter da Tabela "departamento".	47
Figura 73 - Create e Alter da Tabela "docente".	47
Figura 74 - Create e Alter da Tabela "docente_aula".	48
Figura 75 - Create e Alter da Tabela "edificio".	48
Figura 76 - Create e Alter da Tabela "entidade_log".	49

Instituto Politécnico de Coimbra
Instituto Superior de Engenharia de Coimbra

Figura 77 - Create e Alter da Tabela "entrega".....	49
Figura 78 - Create e Alter da Tabela "espaco".....	50
Figura 79 - Create e Alter da Tabela "estado_inscricao".....	50
Figura 80 - Create e Alter da Tabela "estado_matricula".....	51
Figura 81 - Create e Alter da Tabela "estudante".....	51
Figura 82 - Create e Alter da Tabela "estudante_entrega".....	52
Figura 83 - Create e Alter da Tabela "ficheiro_entrega".....	52
Figura 84 - Create e Alter da Tabela "ficheiro_recurso".....	53
Figura 85 - Create e Alter da Tabela "funcao_docente".....	53
Figura 86 - Create e Alter da Tabela "gabinete_docente".....	54
Figura 87 - Create e Alter da Tabela "horario".....	54
Figura 88 - Create e Alter da Tabela "horario_colaborador".....	55
Figura 89 - Create e Alter da Tabela "inscricao".....	55
Figura 90 - Create Alter da Tabela "log".....	56
Figura 91 - Create e Alter da Tabela "matricula".....	56
Figura 92 - Create e Alter da Tabela "nao_docente".....	57
Figura 93 - Create e Alter da Tabela "nota".....	57
Figura 94 - Create e Alter da Tabela "parcela_propina".....	58
Figura 95 - Create e Alter da Tabela "polo".....	58
Figura 96 - Create e Alter da Tabela "presenca".....	59
Figura 97 - Create e Alter da Tabela "propina".....	59
Figura 98 - Create e Alter da Tabela "recurso".....	60
Figura 99 - Create e Alter da Tabela "tipo_aula".....	60
Figura 100 - Create e Alter da Tabela "tipo_avaliacao".....	61
Figura 101 - Create e Alter da Tabela "tipo_contacto".....	61
Figura 102 - Create e Alter da Tabela "tipo_curso".....	62
Figura 103 - Create e Alter da Tabela "tipo_departamento".....	62
Figura 104 - Create e Alter da Tabela "tipo_espaco".....	63
Figura 105 - Create e Alter da Tabela "turma".....	63
Figura 106 - Create e Alter da Tabela "turma_docente".....	64
Figura 107 - Create e Alter da Tabela "turno".....	64
Figura 108 - Create e Alter da Tabela "turno_aulas".....	65
Figura 109 - Create e Alter da Tabela "uc_curso".....	65
Figura 110 - Create e Alter da Tabela "uc_departamento".....	66
Figura 111 - Create e Alter da Tabela "uc_docente".....	66
Figura 112 - Create e Alter da Tabela "unidade_curricular".....	66
Figura 113 - Create e Alter da Tabela "utilizador".....	67
Figura 114 - Alter da Tabela "andar".....	67
Figura 115 - Alter da Tabela "aula".....	68
Figura 116 - Alter da Tabela "avaliacao".....	69

Instituto Politécnico de Coimbra
Instituto Superior de Engenharia de Coimbra

Figura 117 - Alter da Tabela "colaborador".	69
Figura 118 - Alter da Tabela "contacto_estudante".....	70
Figura 119 - Alter da Tabela "curso_area".	70
Figura 120 - Alter da Tabela "curso".	70
Figura 121 - Alter da Tabela "departamento".	71
Figura 122 - Alter da Tabela "docente_aula".	71
Figura 123 - Alter da Tabela "docente".	72
Figura 124 - Alter da Tabela "edificio".....	72
Figura 125 - Alter da Tabela "entrega".	73
Figura 126 - Alter da Tabela "espaco".....	73
Figura 127 - Alter da Tabela "estudante_entrega"	73
Figura 128 - Alter da Tabela "estudante".....	74
Figura 129 - Alter das Tabelas "ficheiro_entrega" e "ficheiro_recurso" .	74
Figura 130 - Alter da Tabela "gabinete_docente".	75
Figura 131 - Alter da Tabela "horario_colaborador".	75
Figura 132 - Alter da Tabela "horario".	76
Figura 133 - Alter da Tabela "matricula".	76
Figura 134 - Alter da Tabela "inscricao".	77
Figura 135 - Alter da Tabela "log".	78
Figura 136 - Outro Alter da Tabela "matricula".	78
Figura 137 - Alter da Tabela "nao_docente".	79
Figura 138 - Alter da Tabela "nota".	79
Figura 139 - Alter da Tabela "parcela_propina".	80
Figura 140 - Alter da Tabela "presenca".....	80
Figura 141 - Alter da Tabela Propina.	80
Figura 142 - Alter da Tabela "recurso".....	81
Figura 143 - Alter da Tabela "turma_docente".	81
Figura 144 - Outro Alter da Tabela "espaco".	82
Figura 145 - Alter da Tabela "turma".....	82
Figura 146 - Alter da Tabela "uc_curso".....	83
Figura 147 - Alter da Tabela "uc_departamento".	83
Figura 148 - Alter da Tabela "uc_docente".	84
Figura 149 - Modelo Físico.	89
Figura 150 - Entidade "curso".	91
Figura 151 - Entidade "tipo_curso".....	91
Figura 152 - Entidade "unidade_curricular".	92
Figura 153 - Entidade "uc_curso".....	92
Figura 154 - Entidade "turma".....	92
Figura 155 - Entidade "aula".....	93
Figura 156 - Entidade "tipo_aula".....	93



Instituto Politécnico de Coimbra
Instituto Superior de Engenharia de Coimbra

Figura 157 - Entidade "uc_docente".	93
Figura 158 - Entidade "docente".	94
Figura 159 - Entidade "estudante".	94
Figura 160 - Entidade "matricula".	95
Figura 161 - Entidade "inscricao".	95
Figura 162 - Entidade "presenca".	96
Figura 163 - Entidade "avaliacao".	96
Figura 164 - Entidade "tipo_avaliacao".	97
Figura 165 - Entidade "nota".	97
Figura 166 - Entidade "entrega".	97
Figura 167 - Entidade "estudante_entrega".	98
Figura 168 - Entidade "ficheiro_entrega".	98
Figura 169 - Entidade "recurso".	98
Figura 170 - Entidade "ficheiro_recurso".	99
Figura 171 - Entidade "parcela_propina".	99
Figura 172 - Entidade "sala".	100
Figura 173 - Entidade "log".	100
Figura 174 - DROP de Tabelas.	102
Figura 175 - CREATE e ALTER da Tabela "aula".	103
Figura 176 - CREATE e ALTER da Tabela "avaliacao".	103
Figura 177 - CREATE e ALTER da Tabela "curso".	104
Figura 178 - CREATE e ALTER da Tabela "docente".	105
Figura 179 - CREATE e ALTER da Tabela "entrega".	105
Figura 180 - CREATE e ALTER da Tabela "estudante".	106
Figura 181 - CREATE e ALTER da Tabela "estudante_entrega".	106
Figura 182 - CREATE e ALTER da Tabela "ficheiro_entrega".	107
Figura 183 - CREATE e ALTER da Tabela "ficheiro_recurso".	107
Figura 184 - CREATE e ALTER da Tabela "inscricao".	108
Figura 185 - CREATE e ALTER da Tabela "log".	108
Figura 186 - CREATE e ALTER da Tabela "matricula".	109
Figura 187 - CREATE e ALTER da Tabela "nota".	109
Figura 188 - CREATE e ALTER da Tabela "parcela_propina".	110
Figura 189 - CREATE e ALTER da Tabela "presenca".	110
Figura 190 - CREATE e ALTER da Tabela "recurso".	111
Figura 191 - CREATE e ALTER da Tabela "sala".	111
Figura 192 - CREATE e ALTER da Tabela "tipo_aula".	112
Figura 193 - CREATE e ALTER da Tabela "tipo_avaliacao".	112
Figura 194 - CREATE e ALTER da Tabela "tipo_curso".	113
Figura 195 - CREATE e ALTER da Tabela "turma".	113
Figura 196 - CREATE e ALTER da Tabela "uc_curso".	114



Instituto Politécnico de Coimbra
Instituto Superior de Engenharia de Coimbra

Figura 197 - CREATE e ALTER da Tabela "uc_docente".	114
Figura 198 - CREATE e ALTER da Tabela "unidade_curricular".....	115
Figura 199 - ALTER da Tabela "aula".....	115
Figura 200 - ALTER da Tabela "avaliacao".....	116
Figura 201 - ALTER das Tabelas "curso" e "entrega".	117
Figura 202 - ALTER da Tabela "estudante_entrega".	117
Figura 203 - ALTER Tabelas "ficheiro_entrega" e "ficheiro_recurso".....	118
Figura 204 - ALTER da Tabela "matricula".....	118
Figura 205 - ALTER da Tabela "inscricao".....	119
Figura 206 - ALTER da Tabela "nota".....	119
Figura 207 - ALTER da Tabela "parcela_propina".....	120
Figura 208 - ALTER da Tabela "presenca".....	120
Figura 209 - ALTER da Tabela "recurso".....	121
Figura 210 - ALTER da Tabela "turma".....	121
Figura 211 - ALTER da Tabela "uc_curso".....	122
Figura 212 - ALTER da Tabela "uc_docente".....	122
Figura 213 - Sequenciação.....	126
Figura 214 - TRIGGERS Auto_Incremento, Parte 1.....	127
Figura 215 - TRIGGERS Auto-Incremento, Parte 2.	128
Figura 216 - PKG_CONSTANTES, Parte 1.....	129
Figura 217 - PKG_CONSTANTES, Parte 2.....	130
Figura 218 - PACKAGE LOG.....	131
Figura 219 - PACKAGE "PKG_GESTAO_DADOS".....	132
Figura 220 - PACKAGE "PKG_VALIDACAO", Parte 1.....	133
Figura 221 - PACKAGE "PKG_VALIDACAO", Parte 2.....	134
Figura 222 - PACKAGE "PKG_BUFFER_MATRICULA", Parte 1.	135
Figura 223 - PACKAGE "PKG_BUFFER_NOTA", Parte 2.....	136
Figura 224 - PACKAGE "PKG_BUFFER_INSCRICAO".....	137
Figura 225 - TRIGGERS de Auditoria "aula" e "avaliacao".....	138
Figura 226 - TRIGGER Auditoria "curso" e "docente".....	139
Figura 227 - TRIGGERS Auditoria "entrega" e "estudante".....	140
Figura 228 - TRIGGERS Audit. "estudante_entrega" e "ficheiro_entrega".....	141
Figura 229 - TRIGGERS Audit. "ficheiro_recurso", "inscricao" e "matricula".....	142
Figura 230 - TRIGGERS Audit. "nota", "parcela_propina" e "presenca".....	143
Figura 231 - TRIGGERS Audit. "recurso", "sala" e "tipo_aula".....	144
Figura 232 - TRIGGERS Audit. "tipo_avaliacao", "curso" e "turma".....	145
Figura 233 - TRIGGERS Audit. "uc_curso" e "uc_docente".....	146
Figura 234 - TRIGGERS Audit. "unidade_curricular" e "ficheiro_recurso".....	147
Figura 235 - PACKAGES Tesouraria, Parte 1.....	148
Figura 236 - PACKAGES Tesouraria, Parte 2.....	149

Instituto Politécnico de Coimbra
Instituto Superior de Engenharia de Coimbra

Figura 237 - TRIGGERS Valid. e Cálculo de Notas.	151
Figura 238 - TRIGGERS Gestão de Presenças.	152
Figura 239 - TRIGGERS Regras de Avaliação, Parte 1.	153
Figura 240 - TRIGGERS Regras de Avaliação, Parte 2.	154
Figura 241 - TRIGGER Valid. de Dados dos Estudantes.....	155
Figura 242 - TRIGGER Valid. de Dados dos Docentes.	156
Figura 243 - TRIGGER Validação de Horários.	157
Figura 244 - TRIGGER Valid. Inscrição em U. Curricular.....	158
Figura 245 - TRIGGER Atualização Automática da Média Geral.	159
Figura 246 - TRIGGERS Valid. de Matrícula e Gerar Propinas pós Matrícula.....	160
Figura 247 - TRIGGERS Validação Salas e Cursos.	161
Figura 248 - TRIGGERS Valid. de Fich. de Entrega e Fich. de Recurso.	162
Figura 249 - TRIGGERS Valid. de Parc. de Propinas e Presenças.	163
Figura 250 - TRIGGER Imutabilidade de Logs.	164
Figura 251 - TRIGGERS Valid. de Recursos, Tipos de Aula e Tipos Avaliação.	164
Figura 252 - TRIGGERS Valid. do Tipo de Curso, U. Curricular e Turma.	165
Figura 253 - TRIGGERS Valid. de UC_Curso e UC_Docente.....	166
Figura 254 - FUNCTION de Assiduidade.	167
Figura 255 - VIEWS Pautas por Turma e Assiduidade.....	168
Figura 256 - VIEWS Perfil Acad., Ocupação de Sala/Dia e Horário dos Docentes.	169
.....	170
Figura 257 - VIEWS Rel. de Dívidas, Análise Financeira Curso e Calendário de Aval.	170
Figura 258 - VIEWS Monitorização Entregas em Grupo e Estatística Ocupação Cursos.	171

Introdução

O presente documento refere-se à primeira fase do trabalho final da disciplina, que vem sendo realizado ao longo das últimas semanas, no âmbito da unidade curricular de Bases de Dados II, lecionada pelo professor Luís Travassos.

O mesmo visa abordar de forma abrangente, mas também dar continuidade, aos conceitos lecionados nas disciplinas de Bases de Dados I e II.

Para a primeira fase do projeto, isto inclui:

- Diagrama Entidade-Relacionamento (ER) do esquema proposto;
- Script SQL com os comandos *CREATE TABLE* e *ALTER TABLE*;
- Script SQL com os comandos *INSERT* para popular as tabelas;
- Breve descrição do contexto do negócio modelado;

Para a segunda fase do projeto, isto inclui:

- Inserção de dados;
- Desenvolvimento de operações PL/SQL;
- Cálculo do tamanho da base de dados;

Todos os tópicos referidos acima, serão abordados no decorrer deste documento.

Nota: O presente relatório é uma continuação do relatório desenvolvido para a primeira fase do projeto, a segunda fase do presente projeto, pode ser encontrada a partir da página 86 (página onde a primeira fase do projeto termina). Esta decisão surge devido à consciência de que o projeto é bastante extenso e ter ambas as fases no mesmo documento torna tudo mais fácil em termos de acessibilidade/usabilidade de quem lê.

Estrutura do Projeto

O projeto foi desenvolvido com recurso às ferramentas: *Data Modeler* que foi usada para modelar e produzir o diagrama físico, bem como o DDL que por sua vez foi exportado para o *SQL Developer*, para fazer a parte da inserção de dados no projeto.

Esta última etapa (inserção de dados), pôde contar com a ajuda do *Mockaroo*, que é uma ferramenta online que permite gerar dados falsos (simulados) de forma altamente personalizada.

Contexto do Negócio Modelado

O nosso grupo escolheu o tema da universidade porque este tipo de sistema representa um cenário realista, complexo e altamente relevante no contexto atual. Universidades lidam diariamente com grandes volumes de dados relacionados a estudantes, professores, cursos, disciplinas, matrículas, avaliações, etc.

Ademais, é um tema com o qual todos temos familiaridade, o que facilita a compreensão dos requisitos e a definição das entidades e atributos necessários. Também consideramos que esta escolha proporciona oportunidades futuras de inclusão de outros conceitos aprendido na disciplina de Base de Dados II, como os triggers, funções, cursores e procedimentos.

Por fim, acreditamos que é uma excelente forma de desenvolver competências que poderão ser aplicadas em diversos contextos profissionais, dado que a gestão eficiente dos dados é uma necessidade presente em várias áreas.

Modelo Físico

Após perceber qual era a tarefa imposta para realizar na primeira fase, e de terem sido identificadas, tanto as entidades, como os atributos de cada entidade e ainda os relacionamentos que as entidades teriam entre elas, chegava a hora de traduzir todos esses dados/informação para o *Data Modeler*.

O *Data Modeler*, serve para criar, visualizar, modificar e gerir modelos de dados, desde a sua fase mais conceitual até ao modelo físico. Por isso foi produzido o modelo físico, que no fundo acaba por ser uma representação mais detalhada e refinada do diagrama de entidade-relacionamento, o seu objetivo foi representar

as entidades como tabelas, os atributos como colunas, e os relacionamentos como chaves estrangeiras.

De referir ainda que, este modelo segue os princípios da normalização (ou seja, elimina redundâncias e melhora a integridade dos dados).

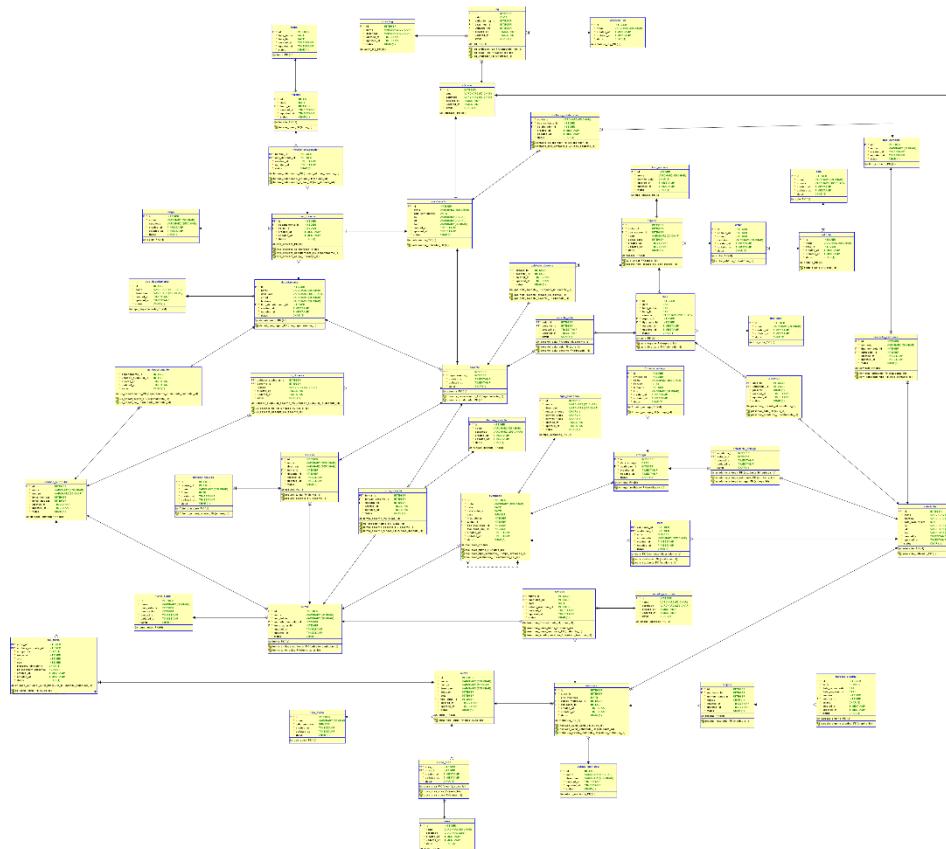


Figura 2 - Modelo Físico.

Este modelo físico comporta entidades separadas por nove tipos de entidades, sendo elas:

1. **Entidades de Utilidade Geral:** Isto é, tabelas genéricas de apoio à infraestrutura e de apoio ao sistema da universidade;
2. **Entidades Académicas Principais:** Estas representam os conceitos centrais do ensino, ou seja, tem uma estrutura mais pedagógica e curricular;
3. **Entidades de Pessoal:** Dividas entre docentes e não docentes, basicamente representam a gestão de recurso humanos da universidade;

4. **Entidades de Estudantes:** Mais focadas em dados pessoais e percurso académico;
5. **Entidades Financeiras:** Relacionadas a propinas e pagamentos, comportam a gestão financeira;
6. **Entidades de Avaliação e Recursos Didáticos:** Neste caso, controlam as avaliações e materiais de apoio;
7. **Entidades Físicas/Infraestrutura:** Isto é, relacionadas com espaços e edifícios, no fundo a infraestrutura física;
8. **Entidades de Associação/Relação:** Servem mais para ligar várias entidades principais (tabelas de junção), citadas acima;
9. **Entidades de Meta dados/Áreas:** Tendem a falar mais da área científica do curso;

Resumindo por tipo, existem sete entidades relativas à utilidade geral (sistema, tipos, logs), existem nove entidades académicas, oito entidades de pessoal, sete entidades de estudantes, duas entidades com foco financeiro da universidade, seis entidades sobre avaliações e recursos didáticos, cinco entidades físicas/infraestrutura, oito entidades de associação/relação e por fim uma única entidade relativa a área científica do curso. Totalizando entidades, incluindo normalização.

Legenda Modelo Físico

A título de elucidação e antes de fechar em definitivo este capítulo relativo ao modelo físico, é necessário fazer a legenda de outros elementos contidos nesse preciso modelo.

- **Asteriscos Antes de Atributos** – Significa que são atributos obrigatórios no quesito de serem preenchidos;
- **P** – A sigla P é relativa a *Primary Key*;
- **F** – A letra F, indica uma *Foreign Key*;
- **N:** – Relacionamento um para muitos;

Entidades ao Pormenor

Este tópico visa explicar ao pormenor cada uma das cinquenta e três entidades (agrupando-as por género naturalmente), isto porque o modelo físico é enorme.

Entidades de Utilidade Geral

Começando por entidades de utilidade geral, como já referido acima, existem sete e apesar de parecer complexo, é fácil subdividir.

Tabela “log”:

Esta tabela, é uma tabela de utilidade geral, existe para suprir a necessidade de registar e armazenar *logs*.

log	
P *	id INTEGER
	data CLOB
F *	entidade_log_id INTEGER
F *	acao_log_id INTEGER
F *	utilizador_id INTEGER
	created_at TIMESTAMP
	updated_at TIMESTAMP
	status CHAR (1)
log_PK (id)	
log_entidade_log_FK (entidade_log_id)	
log_acao_log_FK (acao_log_id)	
log_utilizador_FK (utilizador_id)	

Figura 3 - Tabela "log".

Tabela “acao_log” :

É uma tabela de utilidade geral, isto é, ela acaba por guardar ações registadas no *log* (como por exemplo, a criação, atualização e remoção de dados).

acao_log	
P *	id INTEGER
	nome VARCHAR2 (20 CHAR)
	descricao VARCHAR2 (20 CHAR)
	created_at TIMESTAMP
	updated_at TIMESTAMP
	status CHAR (1)
acao_log_PK (id)	

Figura 4 - Tabela "acao_log".

Tabela “entidade_log”:

Relativamente a esta tabela, é uma tabela de utilidade geral, como tal, surgiu a necessidade de haverem outras entidades que podiam gerar *logs*, daí a criação da mesma.

entidade_log	
P	* id
*	integer
*	nome
*	VARCHAR2 (50 CHAR)
*	created_at
*	TIMESTAMP
*	updated_at
*	TIMESTAMP
*	status
CHAR (1)	
» entidade_log_PK (id)	

Figura 5 - Tabela "entidade_log".

Tabela “tipo_contacto”:

Esta tabela, já não tem como finalidade guardar logs, mas sim contactos, daí ser na mesma uma tabela de utilidade geral no sistema da universidade.

Ela guarda contactos, tipo e-mail, telefone, etc.

tipo_contacto	
P	* id
*	integer
*	nome
*	VARCHAR2 (50 CHAR)
*	created_at
*	TIMESTAMP
*	updated_at
*	TIMESTAMP
*	status
CHAR (1)	
» tipo_contacto_PK (id)	

Figura 6 - Tabela "tipo_contacto".

Tabela “tipo_departamento”:

Esta tabela, serve para guardar o tipo ou categoria de um departamento da universidade e como tal, mantém na categoria de entidades de utilidade geral no sistema.

tipo_departamento	
P	* id
*	integer
*	nome
*	VARCHAR2 (30 CHAR)
*	descricao
*	VARCHAR2 (255 CHAR)
*	created_at
*	TIMESTAMP
*	updated_at
*	TIMESTAMP
*	status
CHAR (1)	
» tipo_departamento_PK (id)	

Figura 7 - Tabela "tipo_departamento".

Tabela “estado_inscricao”:

Esta tabela, mantém-se na mesma categoria que as anteriores, o seu propósito é justamente armazenar o estado de inscrição (se está ativa, se está anulada, etc).

estado_inscricao	
P	* id
	INTEGER
	* nome
	VARCHAR2 (50 CHAR)
	* descricao
	VARCHAR2 (255 CHAR)
	* created_at
	TIMESTAMP
	* updated_at
	TIMESTAMP
	* status
	CHAR (1)
> estado_inscricao_PK (id)	

Figura 8 - Tabela "estado_inscricao".

Tabela “estado_matricula”:

Já esta tabela, tem o mesmo propósito da anterior, mas armazena o estado da matrícula (se a matrícula está ativa, se está anulada, se está suspensa, concluída, etc).

estado_matricula	
P	* id
	INTEGER
	* nome
	VARCHAR2 (50 CHAR)
	descricao
	VARCHAR2 (255 CHAR)
	* created_at
	TIMESTAMP
	* updated_at
	TIMESTAMP
	* status
	CHAR (1)
> estado_matricula_PK (id)	

Figura 9 - Tabela "estado_matricula".

Entidades Académicas Principais:

Neste tópico, fala-se das entidades académicas, que como já referido são uma grande parte do sistema, contando com nove entidades.

Tabela “curso”:

A tabela “curso”, vai armazenar todos os cursos daquela unidade de ensino superior.

curso	
P	* id INTEGER
	* nome VARCHAR2 (100 CHAR)
	* codigo VARCHAR2 (20 CHAR)
	* descricao VARCHAR2 (255 CHAR)
	* duracao INTEGER
	* ects INTEGER
F	* tipo_curso_id INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
» curso_PK (id)	
» curso_tipo_curso_FK (tipo_curso_id)	

Figura 10 - Tabela "curso"

Tabela “tipo_curso”:

Esta tabela é na verdade uma sub-tabela da anterior (curso), que permite armazenar não o curso, mas sim o tipo de curso (CTeSP, licenciatura, mestrado, etc...).

tipo_curso	
P	* id INTEGER
	* nome VARCHAR2 (50 CHAR)
	* valor_propinas NUMBER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
» tipo_curso_PK (id)	

Figura 11 - Tabela "tipo_curso".

Tabela “unidade_curricular”:

Esta tabela diz respeito precisamente à unidade curricular/disciplina.

unidade_curricular	
P	* id INTEGER
	* nome VARCHAR2 (100 CHAR)
	* codigo VARCHAR2 (20 CHAR)
	* horas_teoricas INTEGER
	* horas_praticas INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
» unidade_curricular_PK (id)	

Figura 12 - Tabela "unidade_curricular".

Tabela “uc_curso”:

Esta tabela é na verdade uma associação entre a tabela “curso” e a tabela “unidade_curricular”.

uc_curso	
PF*	curso_id
PF*	unidade_cunicular_id
*	obrigatoria
*	semestre
*	ano
*	ects
*	presenca_obrigatoria
*	percentagem_presenca
*	created_at
*	updated_at
*	status
unidade_cunicular_curso_PK(curso_id, unidade_cunicular_id)	
uc_curso_curso_FK(curso_id)	

Figura 13 - Tabela “uc_curso”.

Tabela “uc_departamento”:

Esta tabela tem o mesmo propósito da anterior, mas desta vez é uma associação entre a tabela “unidade_curricular” e a tabela “departamento”.

uc_departamento	
PF*	departamento_id
PF*	unidade_cunicular_id
*	created_at
*	updated_at
*	status
uc_departamento_PK(departamento_id, unidade_cunicular_id)	
uc_depart_depart_FK(departamento_id)	
uc_depart_uc_FK(unidade_cunicular_id)	

Figura 14 - Tabela “uc_departamento”.

Tabela “uc_docente”:

Mais uma tabela de associação, mas desta vez entre “unidade curricular” e a tabela “docente”.

uc_docente	
PF*	unidade_cunicular_id
PF*	docente_id
	funcao
*	created_at
*	updated_at
*	status
unidade_cunicular_docente_PK(unidade_cunicular_id, docente_id)	
uc_docente_ue_FK(unidade_cunicular_id)	
uc_docente_docente_FK(docente_id)	

Figura 15 - Tabela “uc_docente”.

Tabela “turma”:

A tabela turma, permite guardar dados relativos à turma, relativamente a determinada unidade curricular.

turma	
P *	id INTEGER
*	nome VARCHAR2 (20 CHAR)
*	ano_letivo VARCHAR2 (10 CHAR)
F *	unidade_curricular_id INTEGER
F *	turmo_aulas_id INTEGER
*	created_at TIMESTAMP
*	updated_at TIMESTAMP
*	status CHAR (1)
► turma_PK (id)	
► turma_unidade_curricular_FK (unidade_curricular_id)	
► turma_turmo_aulas_FK (turmo_aulas_id)	

Figura 16 - Tabela "turma".

Tabela “turno”:

Por sua vez, a tabela “turno”, diz respeito ao tipo de horário que ocorrem as aulas (por exemplo, pós laboral, horário integral, etc).

turno	
P *	id INTEGER
*	hora_inicio DATE
*	hora_fim DATE
*	created_at TIMESTAMP
*	updated_at TIMESTAMP
*	status CHAR (1)
► turno_PK (id)	

Figura 17 - Tabela "turno".

Tabela “turno_aulas”:

Embora com um nome muito idêntico à tabela anterior, esta tabela trata de lidar com dados relativos ao grupo de aulas por turno.

turno_aulas	
P *	id INTEGER
*	nome VARCHAR2 (50 CHAR)
*	hora_entrada INTEGER
*	hora_saida INTEGER
*	created_at TIMESTAMP
*	updated_at TIMESTAMP
*	status CHAR (1)
► turno_aulas_PK (id)	

Figura 18 - Tabela "turno_aulas".

Tabela “aula”:

Mais uma das entidades principais deste sistema de gestão de universidade, esta tabela visa armazenar dados relativos à aula em específico.

aula	
P	* id
	INTEGER
*	data
	DATE
*	hora_inicio
	DATE
*	hora_fim
	DATE
	sumario
	VARCHAR2 (255 CHAR)
F	* espaco_id
	INTEGER
F	* tipo_aula_id
	INTEGER
	created_at
	TIMESTAMP
	updated_at
	TIMESTAMP
	status
	CHAR (1)
⇒ aula_PK (id)	
⇒ aula_espaco_FK (espaco_id)	
⇒ aula_tipo_aula_FK (tipo_aula_id)	

Figura 19 - Tabela "aula".

Tabela “tipo_aula”:

Mais uma sub-tabela, mas desta vez da tabela “aula”. Nesta tabela guardam-se dados relativos a se a aula é teórica, prática, laboratorial, etc...

tipo_aula	
P	* id
	INTEGER
*	nome
	VARCHAR2 (50 CHAR)
*	created_at
	TIMESTAMP
*	updated_at
	TIMESTAMP
	status
	CHAR (1)
⇒ tipo_aula_PK (id)	

Figura 20 - Tabela "tipo_aula".

Entidades de Pessoal

Continuando a pormenorizar as diversas entidades desde sistema de gestão de uma universidade, chegou-se às entidades relacionadas com pessoal da universidade.

Tabela “colaborador”:

Esta tabela, é fácil de descodificar, isto porque como o próprio nome diz, a mesma armazena dados relativos a pessoas que colaboram com a instituição (desde professores, funcionários, etc...).

colaborador	
PF*	id
*	nome
*	data_contratacao
cc	DATE
nif	VARCHAR2 (8 CHAR)
iban	VARCHAR2 (9 CHAR)
*	created_at
*	updated_at
*	status
	CHAR (1)
	colaborador_PK (id)
	colaborador_utilizador_FK (id)

Figura 21 - Tabela "colaborador".

Tabela “docente”:

Acima armazenavam-se dados relativos aos colaboradores e agora nesta tabela armazena-se dados relativos aos docentes.

docente	
PF*	id
F	departamento_id
*	created_at
*	updated_at
*	status
	CHAR (1)
	docente_PK (id)
	docente_departamento_FK (departamento_id)
	docente_colaborador_FK (id)

Figura 22 - Tabela "docente".

Tabela “não_docente”:

Esta tabela diz respeito mais propriamente a pessoal colaborador, mas não docente.

nao_docente	
PF*	id
F	departamento_id
F	cargo_id
*	created_at
*	updated_at
*	status
	CHAR (1)
	nao_docente_PK (id)
	nao_docente_colaborador_FK (id)
	nao_docente_departamento_FK (departamento_id)
	nao_docente_cargo_FK (cargo_id)

Figura 23 - Tabela "nao_docente".



Tabela “cargo”:

Neste caso a tabela “cargo” armazena dados relativos ao cargo do pessoal colaborador, mas não docente (funcionário, administrador, presidente, etc...).

cargo	
P	• id
P	• nome
	VARCHAR2 (50 CHAR)
	descrição
	VARCHAR2 (255 CHAR)
	• created_at
	• updated_at
	• status
	CHAR (1)
cargo_PK (id)	

Figura 24 - Tabela "cargo".

Tabela “funcao_docente”:

Se anteriormente se falou de cargos para pessoal não docente, esta tabela vai guardar funções/cargo de docentes (se é professor assistente, se é catedrático, etc...).

funcao_docente	
P	• id
P	• nome
	VARCHAR2 (50 CHAR)
	descrição
	VARCHAR2 (255 CHAR)
	• created_at
	• updated_at
	• status
	CHAR (1)
funcao_docente_PK (id)	

Figura 25 - Tabela "funcao_docente".

Tabela “departamento”:

Esta entidade também se pode classificar com uma entidade principal, isto porque a mesma guardar dados relativos ao departamento académico.

departamento	
P	* id INTEGER
	* nome VARCHAR2 (30 CHAR)
	descricao VARCHAR2 (100 CHAR)
	email VARCHAR2 (50 CHAR)
	telefone VARCHAR2 (20 CHAR)
F	* tipo_departamento_id INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
↳ departamento_PK (id)	
↳ depart_tipo_depart_FK (tipo_departamento_id)	

Figura 26 - Tabela "departamento".

Tabela “gabinete_docente”:

Esta tabela, serve para associar a tabela “docente” e o seu gabinete (tabela “espaco”).

gabinete_docente	
PF	* espaco_id INTEGER
PF	* docente_id INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
↳ gabinete_docente_PK (espaco_id, docente_id)	
↳ gabinete_docente_espaco_FK (espaco_id)	
↳ gabinete_docente_docente_FK (docente_id)	

Figura 27 - Tabela "gabinete_docente".

Tabela “horário_colaborador”:

Neste documento já se sabe da existência de uma entidade relativa aos colaboradores, então esta tabela serve para ligar a tabela “horario” e a tabela “nao_docente”.

horario_colaborador	
PF	* horario_id INTEGER
PF	* nao_docente_id INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
↳ horario_colaborador_PK (horario_id, nao_docente_id)	
↳ horario_colaborador_horario_FK (horario_id)	
↳ horario_colaborador_n_doc_FK (nao_docente_id)	

Figura 28 - Tabela "horario_colaborador".

Entidades de Estudantes

Neste tópico falar-se-á de entidades relativas a estudantes da universidade retratada ao longo deste documento.

Tabela “estudante”:

Esta entidade é considerada uma entidade principal deste sistema, por isso mesmo esta entidade armazena os dados pessoais do estudante.

estudante	
P	* id
F	* nome
	morada
F	* data_nascimento
	cc
	nif
	iban
F	* created_at
F	* updated_at
	status
estudante_PK (id)	
estudante_utilizador_FK (id)	

Figura 29 - Tabela "estudante".

Tabela “contacto_estudante”:

No caso desta tabela, ela nada mais faz do que armazenar os contactos dos estudantes.

contacto_estudante	
P	* id
F	* contacto
F	* tipo_contacto_id
F	* estudante_id
	* created_at
	* updated_at
	status
contacto_PK (id)	
contacto_estudante_FK (estudante_id)	
cont_estudante_tipo_FK (tipo_contacto_id)	

Figura 30 - Tabela "contacto_estudante".

Tabela “matricula”:

Esta entidade como é natural, irá reter dados que dizem respeito às matrículas dos diversos estudantes num curso.

matricula	
P	* id
F	* curso_id
	ano_inscricao
F	* estado_matricula_id
F	* estudante_id
	* created_at
	* updated_at
	* status
matricula_PK (id)	
inscricao_curso_curso_FK (curso_id)	
inscricao_curso_estudante_FK (estudante_id)	
matricula_estado_matricula_FK (estado_matricula_id)	

Figura 31 - Tabela "matricula".

Tabela “inscricao”:

Esta tabela, também é de fácil percepção, uma vez que ira guardar os dados de inscrições em unidades curriculares ou turmas.

inscricao	
PF	* turma_id
PF	* matricula_id
	* data
F	* estado_inscricao_id
	* created_at
	* updated_at
	* status
inscricao_PK (matricula_id, turma_id)	
inscricao_turma_turma_FK (turma_id)	
inscricao_turma_matricula_FK (matricula_id)	
inscricao_estado_inscricao_FK (estado_inscricao_id)	

Figura 32 - Tabela "inscricao".

Tabela “presença”:

Esta entidade armazena precisamente o registo de presenças em aulas.

presença	
PF*	aula_id
PF*	estudante_id
*	presente
*	created_at
*	updated_at
*	status
► presencia_PK (aula_id, estudante_id)	
► presencia_aula_FK (aula_id)	
► presencia_estudante_FK (estudante_id)	

Figura 33 - Tabela "presença".

Tabela “nota”:

Esta tabela diz respeito às notas dos alunos e como tal, armazena as mesmas, bem como possíveis comentários de avaliação.

nota	
PF*	estudante_id
PF*	avaliacao_id
*	nota
*	comentario
*	VARCHAR2 (255 CHAR)
*	created_at
*	updated_at
*	status
► nota_PK (avaliacao_id, estudante_id)	
► nota_avaliacao_FK (avaliacao_id)	
► nota_estudante_FK (estudante_id)	

Figura 34 - Tabela "nota".

Tabela “estado_inscricao”:

Acima foi reportada uma entidade denominada por “inscricao”, como tal esta tabela irá guardar o estado da mesma.

estado_inscricao	
P	* id
	INTEGER
	* nome
	VARCHAR2 (50 CHAR)
	* descricao
	VARCHAR2 (255 CHAR)
	* created_at
	TIMESTAMP
	* updated_at
	TIMESTAMP
	* status
	CHAR (1)
» estado_inscricao_PK (id)	

Figura 35 - Tabela "estado_inscricao".

Tabela “estado_matricula”:

À semelhança da tabela anterior, neste documento já foi falado de uma entidade denominada de “matricula”, como tal esta entidade armazena o estado dessa mesma matrícula.

estado_matricula	
P	* id
	INTEGER
	* nome
	VARCHAR2 (50 CHAR)
	* descricao
	VARCHAR2 (255 CHAR)
	* created_at
	TIMESTAMP
	* updated_at
	TIMESTAMP
	* status
	CHAR (1)
» estado_matricula_PK (id)	

Figura 36 - Tabela "estado_matricula".

Entidades Financeiras:

Prosseguindo na desmistificação das diversas entidades, chegou-se ao ponto de se falar das entidades financeiras.

Tabela “propina”:

Mais um caso de uma entidade principal deste sistema. Esta tabela armazena precisamente a propina associada à matrícula.

propina	
P	* id
F	* matricula_id
	* numero_parcelas
	* estado
	* created_at
	* updated_at
	* status
	INTEGER
	INTEGER
	INTEGER
	CHAR (1)
	TIMESTAMP
	TIMESTAMP
	CHAR (1)
	propina_PK (id)
	propina_matricula_FK (matricula_id)

Figura 37 - Tabela "propina".

Tabela “parcela_propina”:

Esta tabela representa na verdade uma sub-tabela da tabela “propina”. Posto isto, o seu propósito é armazenar os dados relativos às parcelas de pagamento de propinas.

parcela_propina	
P	* id
	* valor
	* data_vencimento
	data_pagamento
	* numero
F	* propina_id
	* estado
	* created_at
	* updated_at
	* status
	INTEGER
	NUMBER
	DATE
	DATE
	INTEGER
	INTEGER
	CHAR (1)
	TIMESTAMP
	TIMESTAMP
	CHAR (1)
	parcela_propina_PK (id)
	parcela_propina_propina_FK (propina_id)

Figura 38 - Tabela "parcela_propina".

Entidades de Avaliação e Recursos Didáticos:

Quase a acabar, mas ainda importa falar de sete entidades relacionadas ao tópico no título descrito.

Tabela “avaliacao”:

Mais um caso de uma entidade principal, esta tabela armazenará os dados relativos às avaliações dos alunos.

avaliacao	
P	* id INTEGER
	* titulo VARCHAR2 (100 CHAR)
	* data DATE
	* data_entrega DATE
	* peso NUMBER
	* max_alunos INTEGER
F	* turma_id INTEGER
F	* tipo_avaliacao_id INTEGER
F	avaliacao_pai_id INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
avaliacao_PK (id)	
avaliacao_turma_FK (turma_id)	
avaliacao_tipo_avaliacao_FK (tipo_avaliacao_id)	
avaliacao_avaliacao_FK (avaliacao_pai_id)	

Figura 39 - Tabela "avaliacao".

Tabela “tipo_avaliacao”:

Mais uma sub-tabela, desta vez é uma sub-tabela da entidade “avaliacao” e como tal irá guardar dados relativos ao tipo de avaliação, isto é, se é exame, frequência, trabalhos, etc...

tipo_avaliacao	
P	* id INTEGER
	* nome VARCHAR2 (50 CHAR)
	descricao VARCHAR2 (255 CHAR)
	* requer_entrega CHAR (1)
	* permite_grupo CHAR (1)
	* permite_filhos CHAR (1)
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
tipo_avaliacao_PK (id)	

Figura 40 - Tabela "tipo_avaliacao".

Tabela “entrega”:

Esta tabela é uma tabela de associação e relativa à entidade “avaliacao”, como tal guarda dados relativos às entregas para avaliações.

entrega	
P	* id INTEGER
	* data_entrega DATE
F	* avaliacao_id INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
↳ entrega_PK (id)	
↳ entrega_avaliacao_FK (avaliacao_id)	

Figura 41 - Tabela "entrega".

Tabela “estudante_entrega”:

Falando da tabela “estudante_entrega”, tal como o nome revela, faz a associação entre a entidade “estudante” e “entrega”.

estudante_entrega	
PF*	estudante_id INTEGER
PF*	entrega_id INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
↳ estudante_entrega_PK (estudante_id, entrega_id)	
↳ estudante_entrega_estudante_FK (estudante_id)	
↳ estudante_entrega_entrega_FK (entrega_id)	

Figura 42 - Tabela "estudante_entrega".

Tabela “ficheiro_entrega”:

Para complementar ainda mais a parte de entrega de materiais para avaliação, nasce a necessidade de criar esta tabela.

ficheiro_entrega	
P	* id INTEGER
F	* entrega_id INTEGER
	* nome VARCHAR2 (100 CHAR)
	* ficheiro BLOB
	* tamanho INTEGER
	* tipo VARCHAR2 (20 CHAR)
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
↳ ficheiro_entrega_PK (id)	
↳ ficheiro_entrega_FK (entrega_id)	

Figura 43 - Tabela "ficheiro_entrega".

Tabela “recurso”:

Para os alunos que infelizmente têm azar e têm de ir a recurso, surge esta tabela que visa armazenar precisamente os dados do recurso.

recurso	
P	* id INTEGER
	* nome VARCHAR2 (100 CHAR)
	* descricao VARCHAR2 (255 CHAR)
F	* turma_id INTEGER
F	* docente_id INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
↳ recurso_PK (id)	
↳ recurso_turma_FK (turma_id)	
↳ recurso_docente_FK (docente_id)	

Figura 44 - Tabela "recurso".

Tabela “ficheiro_recurso”:

Uma vez mais, houve necessidade de complementar a entidade recurso e como tal surgiu a tabela “ficheiro_recurso”, que guarda os ficheiros de recurso, como PDF’s por exemplo.

ficheiro_recurso	
P	* id INTEGER
F	* recurso_id INTEGER
	* nome VARCHAR2 (100 CHAR)
	* ficheiro BLOB
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
↳ ficheiro_recurso_PK (id)	
↳ ficheiro_recurso_recurso_FK (recurso_id)	

Figura 45 - Tabela "ficheiro_recurso".

Entidades Físicas/Infraestrutura:

Faltando apenas três tópicos para terminar este capítulo de observação ao pormenor das diferentes entidades do sistema de gestão da universidade, falar-se-á agora de entidades relativas à infraestrutura da universidade.

Tabela “polo”:

Começando pela entidade principal deste tópico, é razoavelmente entendível que esta entidade irá reter todos os dados relativos ao edifício, como quem o polo/campus universitário.

polo	
P	* id
	INTEGER
	* nome
	VARCHAR2 (50 CHAR)
	* morada
	VARCHAR2 (100 CHAR)
	* created_at
	TIMESTAMP
	* updated_at
	TIMESTAMP
	* status
	CHAR (1)
► polo_PK (id)	

Figura 46 - Tabela "polo".

Tabela “edificio”:

Se anteriormente se guardaram dados na entidade “polo”, na entidade “edificio” adensa-se ainda mais o fator de armazenamento de dados do edifício (instalações da universidade”.

edificio	
P	* id
	INTEGER
	* nome
	VARCHAR2 (50 CHAR)
F	* polo_id
	INTEGER
	* created_at
	TIMESTAMP
	* updated_at
	TIMESTAMP
	* status
	CHAR (1)
► edificio_PK (id)	
► edificio_polo_FK (polo_id)	

Figura 47 - Tabela "edificio".

Tabela “andar”:

Já deu para entender que existe uma entidade “polo” e uma entidade “edifício”, mas os dados relativos ao andar do edifício também são armazenados.

andar	
P	• id
F	• edificio_id
	• numero
	• descricao
	• created_at
	• updated_at
	• status
↳ andar_PK (id)	
↳ andar_edificio_FK (edificio_id)	

Figura 48 - Tabela "andar".

Tabela “espaco”:

Na tabela “espaco” é possível guardar dados relativos ao espaço físico, isto inclui por exemplo, salas, laboratórios, gabinetes de professor/atendimento.

espaco	
P	• id
F	• andar_id
F	• tipo_espaco_id
	• nome
	• capacidade
	• created_at
	• updated_at
	• status
↳ espaco_PK (id)	
↳ sala_andar_FK (andar_id)	
↳ espaco_tipo_espaco_FK (tipo_espaco_id)	

Figura 49 - Tabela "espaco".

Tabela “tipo_espaco”:

Por último e ainda relativamente a entidades que visam guardar dados da estrutura física, há uma tabela que guarda o tipo de espaço, que é uma sub-tabela da tabela “espaco” e vai essencialmente guardar a disponibilidade das salas para permitir aulas ou não.

tipo_espaco	
P	* id
	INTEGER
	* nome
	VARCHAR2 (50 CHAR)
	* permite_aula
	CHAR (1)
	* created_at
	TIMESTAMP
	* updated_at
	TIMESTAMP
	* status
	CHAR (1)
► tipo_espaco_PK (id)	

Figura 50 - Tabela "tipo_espaco".

Entidades de Associação e Relação:

Como já referido anteriormente no presente documento, existem entidades mais focadas em guardar certas relações entre entidades principais e é precisamente isso que este tópico visa descrever.

Relembrando que algumas destas tabelas, já foram faladas anteriormente, mas por uma questão de fácil percepção e explicação, é preferível descrevê-las.

Tabela “curso_area”:

Trata-se de uma tabela que serve para guardar dados que são fruto da relação da tabela “curso” e da tabela “área” (esta última, ainda não foi descrita).

curso_area	
PF	* area_id
	INTEGER
PF	* curso_id
	INTEGER
	* created_at
	TIMESTAMP
	* updated_at
	TIMESTAMP
	* status
	CHAR (1)
► curso_area_PK (area_id, curso_id)	
► curso_area_area_FK (area_id)	
► curso_area_curso_FK (curso_id)	

Figura 51 - Tabela "curso_area".

Tabela “docente_aula”:

Espelhando a tabela anterior, esta tabela surgiu da necessidade de guardar dados entre a relação da entidade “docente” e a entidade “aula”.

docente_aula	
PF*	aula_id
PF*	docente_id
▪	created_at
▪	updated_at
▪	status
docente_aula_PK (aula_id, docente_id)	
docente_aula_aula_FK (aula_id)	
docente_aula_docente_FK (docente_id)	

Figura 52 - Tabela "docente_aula".

Tabela “turma_docente”:

Seguindo o mesmo pensamento do tópico anterior, esta tabela descreve a relação entre a entidade principal “turma” e a entidade principal “docente”.

turma_docente	
PF*	turma_id
F	funcao_docente_id
F	docente_id
▪	created_at
▪	updated_at
▪	status
turma_docente_PK (turma_id)	
uc_docente_turma_FK (turma_id)	
turma_docente_docente_FK (docente_id)	
turma_docente_funcao_FK (funcao_docente_id)	

Figura 53 - Tabela "turma_docente".

Tabela “gabinete_docente”:

Continuando, mais uma tabela para guardar dados furtuitos das entidades “docente” e “espaco”, no caso cada professor tem o seu gabinete.

gabinete_docente	
PF*	espaco_id
PF*	docente_id
▪	created_at
▪	updated_at
▪	status
gabinete_docente_PK (espaco_id, docente_id)	
gabinete_docente_espaco_FK (espaco_id)	
gabinete_docente_docente_FK (docente_id)	

Figura 54 - Tabela "gabinete_docente".

Tabela “estudante_entrega”:

Esta tabela é o resultado da relação entre as entidades principais “estudante” e “entrega”.

estudante_entrega	
PF*	estudante_id INTEGER
PF*	entrega_id INTEGER
*	created_at TIMESTAMP
*	updated_at TIMESTAMP
*	status CHAR (1)
↳ estudante_entrega_PK (estudante_id, entrega_id)	
↳ estudante_entrega_estudante_FK (estudante_id)	
↳ estudante_entrega_entrega_FK (entrega_id)	

Figura 55 - Tabela "estudante_entrega".

Tabela “horário_colaborador”:

Uma associação entre a entidade “horario” e a entidade “não_docente”, no caso um não docente é um colaborador da universidade.

horario_colaborador	
PF*	horario_id INTEGER
PF*	não_docente_id INTEGER
*	created_at TIMESTAMP
*	updated_at TIMESTAMP
*	status CHAR (1)
↳ horario_colaborador_PK (horario_id, não_docente_id)	
↳ horario_colaborador_horario_FK (horario_id)	
↳ horario_colaborador_n_doc_FK (não_docente_id)	

Figura 56 - Tabela "horario_colaborador".

Tabela “uc_curso”:

Por último, em relação a estas tabelas de associação entre entidades principais, existe a tabela “uc_curso”, que na mais é do que o resultado da relação da tabela “curso” e “unidade_curricular”.

uc_curso	
PF* curso_id	INTEGER
PF* unidade_curricular_id	INTEGER
* obrigatoria	CHAR (1)
* semestre	INTEGER
* ano	INTEGER
* ects	INTEGER
* presenca_obrigatoria	CHAR (1)
presenca_percentagem_presenca	NUMBER
* created_at	TIMESTAMP
* updated_at	TIMESTAMP
* status	CHAR (1)
► unidade_curricular_curso_PK (curso_id, unidade_curricular_id)	
► uc_curso_curso_FK (curso_id)	

Figura 57 - Tabela "uc_curso".

Entidades de Meta dados/Áreas:

Contando com apenas uma tabela, surgiu a necessidade haver uma tabela que guardasse dados das áreas científicas presentes na universidade.

Foi então que nasceu a tabela “área”.

Tabela “area”:

Esta tabela guarda dados relativos à área científica (por exemplo, matemática, informática, etc...).

area	
P * id	INTEGER
* nome	VARCHAR2 (50 CHAR)
descricao	VARCHAR2 (255)
* created_at	TIMESTAMP
* updated_at	TIMESTAMP
* status	CHAR (1)
► area_PK (id)	

Figura 58 - Tabela "area".

Relacionamentos do Modelo Físico:

Ainda relativamente ao modelo físico, sabemos que é um modelo gigante e complexo, outrora não fosse a tentativa de representação real de um sistema de gestão de uma universidade.

Como tal, abaixo, segue uma tabela descritiva de todas as relações presentes no modelo, bem como as entidades que se relacionam e a cardinalidade das mesmas.

Entidade	Cardinalidade	Entidade Relacionada
andar	N:	edificio
edificio	N:	polo
espaco	N:	andar
espaco	N:	tipo_espaco
aula	N:	espaco
aula	N:	tipo_aula
docente_aula	N:	docente
docente_aula	N:	aula
avaliacao	N:	turma
avaliacao	N:	tipo_avaliacao
avaliacao	N: (não obrigatório)	avaliacao (auto-relacionamento)
entrega	N:	avaliacao
estudante_entrega	N:	entrega
estudante_entrega	N:	estudante
ficheiro_entrega	N:	entrega
recurso	N:	turma
recurso	N:	docente
ficheiro_recurso	N:	recurso
docente	N:	departamento
nao_docente	N:	departamento
nao_docente	N:	cargo
gabinete_docente	N:	docente
gabinete_docente	N:	espaco
horario	N:	turno
horario_colaborador	N:	horario
horario_colaborador	N:	nao_docente
inscricao	N:	turma
inscricao	N:	matricula
inscricao	N:	estado_inscricao
matricula	N:	curso
matricula	N:	estado_matricula
matricula	N:	estudante
propina	N:	matricula
parcela_propina	N:	propina
turma	N:	unidade_curricular
turma	N:	turno_aulas
turma_docente	N:	turma
turma_docente	N:	docente
turma_docente	N:	funcao_docente
curso_area	N:	curso
curso_area	N:	area

uc_curso	N:	curso
uc_curso	N:	unidade_curricular
contacto_estudante	N:	estudante
contacto_estudante	N:	tipo_contacto
contacto_colaborador	N:	colaborador
contacto_colaborador	N:	tipo_contacto
nota	N:	avaliacao
nota	N:	Estudante
log	N:	acao_log
log	N: (não obrigatório)	entidade_log
log	N:	utilizador

Data Definition Language – DDL

Ora o DDL é um subconjunto da linguagem SQL, é usado para definir e gerir a estrutura dos objetos da base de dados, objetos esses que podem ser, tabelas, índices, vistas ou até mesmo esquemas.

Existem vários comandos DDL, mas os principais usados neste projeto são, o *CREATE* que tal como indica o nome, cria tabelas, o *ALTER* que também foi usado neste projeto e tal como nome diz, altera/modifica a estrutura de objetos já existentes, o *DROP* que remove objetos da base de dados e que também foi usado.

Estrutura do DDL

Diz-se por aí que dar *DROP* nas tabelas antes de fazer qualquer tipo de operação é uma boa prática comum, isto porque evita duplicados, caso certo objeto já exista, além disso garante um controlo melhor sobre a estrutura, isto porque o utilizador sabe exatamente a estrutura que foi criada e não depende do estado anterior da base de dados. É também útil em ambientes de desenvolvimento/teste, onde o *script* é executado várias vezes e se pode dar *reset* ao estado.

Como não podia deixar de ser, começou-se por fazer *DROP* a todas as tabelas.

```
8   DROP TABLE acao_log CASCADE CONSTRAINTS
9   ;
10
11  DROP TABLE andar CASCADE CONSTRAINTS
12  ;
13
14  DROP TABLE area CASCADE CONSTRAINTS
15  ;
16
17  DROP TABLE aula CASCADE CONSTRAINTS
18  ;
19
20  DROP TABLE avaliacao CASCADE CONSTRAINTS
21  ;
22
23  DROP TABLE cargo CASCADE CONSTRAINTS
24  ;
25
26  DROP TABLE colaborador CASCADE CONSTRAINTS
27  ;
```

Figura 59 - Drop nas Tabelas.

De referir ainda, que devido à extensão do script DDL (cerca de linhas), a Figura 60, representa uma pequena parte das tabelas. O restante script acompanha o relatório aquando da sua submissão.

Logo após o *DROP*, foi possível arrancar para a criação da primeira tabela, que neste caso foi a tabela “acao_log”, é importante relembrar que o DDL quando está a ser desenvolvido, normalmente começa-se a criar tabelas pela ordem de eventos e dependências. No caso, não foi isso que aconteceu, isto porque preferimos uma abordagem diferente (ordem alfabética) para não nos perdemos.

```
CREATE TABLE acao_log
(
    id      INTEGER NOT NULL ,
    nome    VARCHAR2 (20 CHAR) NOT NULL ,
    descricao VARCHAR2 (20 CHAR) NOT NULL ,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status   CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE acao_log
ADD CONSTRAINT acao_log_PK PRIMARY KEY ( id ) ;
```

Figura 61 - Create e Alter Tabela "acao_log".

Por fim, foi feito o *ALTER*, que por meio de uma restrição (*CONSTRAINT*), define a chave primária da tabela como sendo o campo “id”. Isto garante que esse campo contenha valores únicos e não nulos, assegurando a identificação exclusiva de cada registo.

De modo geral todas a tabelas presentes no nosso *script DDL*, foram criadas da mesma forma e com os seus respetivos atributos, alteradas/modificadas usando os mesmos comandos/cláusulas, pelo que se entende que mais explicações não serão propriamente necessárias.

```
CREATE TABLE andar
(
    id          INTEGER NOT NULL ,
    edificio_id INTEGER NOT NULL ,
    numero      INTEGER NOT NULL ,
    descricao   VARCHAR2 (50 CHAR) ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE andar
ADD CONSTRAINT andar_PK PRIMARY KEY ( id ) ;
```

Figura 62 - Create e Alter da Tabela "andar".

```
CREATE TABLE area
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    descricao   VARCHAR2 (255) ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE area
ADD CONSTRAINT area_PK PRIMARY KEY ( id ) ;
```

Figura 63 - Create e Alter da Tabela "area".

```
CREATE TABLE aula
(
    id          INTEGER NOT NULL ,
    data        DATE NOT NULL ,
    hora_inicio DATE NOT NULL ,
    hora_fim    DATE NOT NULL ,
    sumario     VARCHAR2 (255 CHAR) ,
    espaco_id   INTEGER NOT NULL ,
    tipo_aula_id INTEGER NOT NULL ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE aula
ADD CONSTRAINT aula_PK PRIMARY KEY ( id ) ;
```

Figura 64 - Create e Alter da Tabela "aula".

```
CREATE TABLE avaliacao
(
    id          INTEGER NOT NULL ,
    titulo      VARCHAR2 (100 CHAR) NOT NULL ,
    data        DATE NOT NULL ,
    data_entrega DATE NOT NULL ,
    peso         NUMBER ,
    max_alunos  INTEGER NOT NULL ,
    turma_id    INTEGER NOT NULL ,
    tipo_avaliacao_id INTEGER NOT NULL ,
    avaliacao_pai_id INTEGER ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE avaliacao
ADD CONSTRAINT avaliacao_PK PRIMARY KEY ( id ) ;
```

Figura 65 - Create e Alter da Tabela "avaliacao".

```
CREATE TABLE cargo
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    descricao   VARCHAR2 (255 CHAR) ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE cargo
ADD CONSTRAINT cargo_PK PRIMARY KEY ( id ) ;
```

Figura 66 - Create e Alter da Tabela "cargo".

```
CREATE TABLE colaborador
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (100 CHAR) NOT NULL ,
    data_contratacao DATE NOT NULL ,
    cc          VARCHAR2 (8 CHAR) ,
    nif         VARCHAR2 (9 CHAR) ,
    iban        VARCHAR2 (25 CHAR) ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE colaborador
ADD CONSTRAINT colaborador_PK PRIMARY KEY ( id ) ;
```

Figura 67 - Create e Alter da Tabela "colaborador".

```
CREATE TABLE contacto_colaborador
(
    contacto      VARCHAR2 (50 CHAR) NOT NULL ,
    tipo_contacto_id INTEGER NOT NULL ,
    colaborador_id INTEGER NOT NULL ,
    created_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;
```

Figura 68 - Create da Tabela "contacto_colaborador".

```
CREATE TABLE contacto_estudante
(
    id           INTEGER NOT NULL ,
    contacto     VARCHAR2 (50 CHAR) NOT NULL ,
    tipo_contacto_id INTEGER NOT NULL ,
    estudante_id INTEGER NOT NULL ,
    created_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE contacto_estudante
ADD CONSTRAINT contacto_PK PRIMARY KEY ( id ) ;
```

Figura 69 - Create e Alter da Tabela "contacto_estudante".

```
CREATE TABLE curso
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (100 CHAR) NOT NULL ,
    codigo      VARCHAR2 (20 CHAR) NOT NULL ,
    descricao   VARCHAR2 (255 CHAR) NOT NULL ,
    duracao     INTEGER NOT NULL ,
    ects         INTEGER NOT NULL ,
    tipo_curso_id INTEGER NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE curso
ADD CONSTRAINT curso_PK PRIMARY KEY ( id ) ;
```

Figura 70 - Create e Alter da Tabela "curso".

```
CREATE TABLE curso_area
(
    area_id     INTEGER NOT NULL ,
    curso_id    INTEGER NOT NULL ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE curso_area
ADD CONSTRAINT curso_area_PK PRIMARY KEY ( area_id, curso_id ) ;
```

Figura 71 - Create e Alter da Tabela "curso_area".



```
CREATE TABLE departamento
(
    id             INTEGER NOT NULL ,
    nome           VARCHAR2 (30 CHAR) NOT NULL ,
    descricao      VARCHAR2 (100 CHAR) ,
    email          VARCHAR2 (50 CHAR) ,
    telefone       VARCHAR2 (20 CHAR) ,
    tipo_departamento_id INTEGER NOT NULL ,
    created_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status          CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE departamento
ADD CONSTRAINT departamento_PK PRIMARY KEY ( id ) ;
```

Figura 72 - Create e Alter da Tabela "departamento".

```
CREATE TABLE docente
(
    id             INTEGER NOT NULL ,
    departamento_id INTEGER NOT NULL ,
    created_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status          CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE docente
ADD CONSTRAINT docente_PK PRIMARY KEY ( id ) ;
```

Figura 73 - Create e Alter da Tabela "docente".

```
CREATE TABLE docente_aula
(
    aula_id      INTEGER NOT NULL ,
    docente_id   INTEGER NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE docente_aula
ADD CONSTRAINT docente_aula_PK PRIMARY KEY ( aula_id, docente_id ) ;
```

Figura 74 - Create e Alter da Tabela "docente_aula".

```
CREATE TABLE edificio
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    polo_id     INTEGER NOT NULL ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE edificio
ADD CONSTRAINT edificio_PK PRIMARY KEY ( id ) ;
```

Figura 75 - Create e Alter da Tabela "edificio".

```
CREATE TABLE entidade_log
(
    id      INTEGER NOT NULL ,
    nome    VARCHAR2 (50 CHAR) NOT NULL ,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status   CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE entidade_log
ADD CONSTRAINT entidade_log_PK PRIMARY KEY ( id ) ;
```

Figura 76 - Create e Alter da Tabela "entidade_log".

```
CREATE TABLE entrega
(
    id      INTEGER NOT NULL ,
    data_entrega DATE NOT NULL ,
    avaliacao_id INTEGER NOT NULL ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status     CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE entrega
ADD CONSTRAINT entrega_PK PRIMARY KEY ( id ) ;
```

Figura 77 - Create e Alter da Tabela "entrega".

```
CREATE TABLE espaco
(
    id          INTEGER NOT NULL ,
    andar_id    INTEGER NOT NULL ,
    tipo_espaco_id INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    capacidade  INTEGER NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE espaco
ADD CONSTRAINT espaco_PK PRIMARY KEY ( id ) ;
```

Figura 78 - Create e Alter da Tabela "espaco".

```
CREATE TABLE estado_inscricao
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    descricao   VARCHAR2 (255 CHAR) NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE estado_inscricao
ADD CONSTRAINT estado_inscricao_PK PRIMARY KEY ( id ) ;
```

Figura 79 - Create e Alter da Tabela "estado_inscricao".



```
CREATE TABLE estado_matricula
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    descricao   VARCHAR2 (255 CHAR) ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE estado_matricula
ADD CONSTRAINT estado_matricula_PK PRIMARY KEY ( id ) ;
```

Figura 80 - Create e Alter da Tabela "estado_matricula".

```
CREATE TABLE estudante
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    morada     VARCHAR2 (100 CHAR) ,
    data_nascimento DATE NOT NULL ,
    cc          VARCHAR2 (8) ,
    nif         VARCHAR2 (9) ,
    iban        VARCHAR2 (25 CHAR) ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE estudante
ADD CONSTRAINT estudante_PK PRIMARY KEY ( id ) ;
```

Figura 81 - Create e Alter da Tabela "estudante".

```
CREATE TABLE estudante_entrega
(
    estudante_id INTEGER NOT NULL ,
    entrega_id   INTEGER NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE estudante_entrega
ADD CONSTRAINT estudante_entrega_PK PRIMARY KEY ( estudante_id, entrega_id ) ;
```

Figura 82 - Create e Alter da Tabela "estudante_entrega".

```
CREATE TABLE ficheiro_entrega
(
    id          INTEGER NOT NULL ,
    entrega_id  INTEGER NOT NULL ,
    nome         VARCHAR2 (100 CHAR) NOT NULL ,
    ficheiro     BLOB NOT NULL ,
    tamanho      INTEGER NOT NULL ,
    tipo         VARCHAR2 (20 CHAR) NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE ficheiro_entrega
ADD CONSTRAINT ficheiro_entrega_PK PRIMARY KEY ( id ) ;
```

Figura 83 - Create e Alter da Tabela "ficheiro_entrega".

```
CREATE TABLE ficheiro_recurso
(
    id          INTEGER NOT NULL ,
    recurso_id INTEGER NOT NULL ,
    nome        VARCHAR2 (100 CHAR) NOT NULL ,
    ficheiro    BLOB NOT NULL ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE ficheiro_recurso
ADD CONSTRAINT ficheiro_recurso_PK PRIMARY KEY ( id ) ;
```

Figura 84 - Create e Alter da Tabela "ficheiro_recurso".

```
CREATE TABLE funcao_docente
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    descricao   VARCHAR2 (255 CHAR) ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE funcao_docente
ADD CONSTRAINT funcao_docente_PK PRIMARY KEY ( id ) ;
```

Figura 85 - Create e Alter da Tabela "funcao_docente".

```
CREATE TABLE gabinete_docente
(
    espaco_id INTEGER NOT NULL ,
    docente_id INTEGER NOT NULL ,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status     CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE gabinete_docente
ADD CONSTRAINT gabinete_docente_PK PRIMARY KEY ( espaco_id, docente_id ) ;
```

Figura 86 - Create e Alter da Tabela "gabinete_docente".

```
CREATE TABLE horario
(
    id         INTEGER NOT NULL ,
    data       DATE NOT NULL ,
    turno_id   INTEGER NOT NULL ,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status     CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE horario
ADD CONSTRAINT horario_PK PRIMARY KEY ( id ) ;
```

Figura 87 - Create e Alter da Tabela "horario".

```
CREATE TABLE horario_colaborador
(
    horario_id      INTEGER NOT NULL ,
    nao_docente_id INTEGER NOT NULL ,
    created_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status          CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE horario_colaborador
    ADD CONSTRAINT horario_colaborador_PK PRIMARY KEY ( horario_id, nao_docente_id ) ;
```

Figura 88 - Create e Alter da Tabela "horario_colaborador".

```
CREATE TABLE inscricao
(
    turma_id        INTEGER NOT NULL ,
    matricula_id    INTEGER NOT NULL ,
    data            DATE NOT NULL ,
    estado_inscricao_id INTEGER NOT NULL ,
    created_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status          CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE inscricao
    ADD CONSTRAINT inscricao_PK PRIMARY KEY ( matricula_id, turma_id ) ;
```

Figura 89 - Create e Alter da Tabela "inscricao".

```
CREATE TABLE log
(
    id          INTEGER NOT NULL ,
    data        CLOB ,
    entidade_log_id INTEGER ,
    acao_log_id   INTEGER NOT NULL ,
    utilizador_id  INTEGER NOT NULL ,
    created_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE log
ADD CONSTRAINT log_PK PRIMARY KEY ( id ) ;
```

Figura 90 - Create Alter da Tabela "log".

```
CREATE TABLE matricula
(
    id          INTEGER NOT NULL ,
    curso_id     INTEGER NOT NULL ,
    ano_inscricao DATE ,
    estado_matricula_id INTEGER NOT NULL ,
    estudante_id  INTEGER NOT NULL ,
    created_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE matricula
ADD CONSTRAINT matricula_PK PRIMARY KEY ( id ) ;
```

Figura 91 - Create e Alter da Tabela "matricula".

```
CREATE TABLE nao_docente
(
    id          INTEGER NOT NULL ,
    departamento_id INTEGER NOT NULL ,
    cargo_id     INTEGER NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE nao_docente
ADD CONSTRAINT nao_docente_PK PRIMARY KEY ( id ) ;
```

Figura 92 - Create e Alter da Tabela "nao_docente".

```
CREATE TABLE nota
(
    estudante_id INTEGER NOT NULL ,
    avaliacao_id INTEGER NOT NULL ,
    nota         NUMBER NOT NULL ,
    comentario   VARCHAR2 (255 CHAR) ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE nota
ADD CONSTRAINT nota_PK PRIMARY KEY ( avaliacao_id, estudante_id ) ;
```

Figura 93 - Create e Alter da Tabela "nota".

```
CREATE TABLE parcela_propina
(
    id          INTEGER NOT NULL ,
    valor        NUMBER NOT NULL ,
    data_vencimento DATE NOT NULL ,
    data_pagamento DATE ,
    numero       INTEGER NOT NULL ,
    propina_id   INTEGER NOT NULL ,
    estado        CHAR (1) NOT NULL ,
    created_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at    TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status         CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE parcela_propina
ADD CONSTRAINT parcela_propina_PK PRIMARY KEY ( id ) ;
```

Figura 94 - Create e Alter da Tabela "parcela_propina".

```
CREATE TABLE polo
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    morada      VARCHAR2 (100 CHAR) NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE polo
ADD CONSTRAINT polo_PK PRIMARY KEY ( id ) ;
```

Figura 95 - Create e Alter da Tabela "polo".

```
CREATE TABLE presenca
(
    aula_id      INTEGER NOT NULL ,
    estudante_id INTEGER NOT NULL ,
    presente     CHAR (1) NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE presenca
ADD CONSTRAINT presenca_PK PRIMARY KEY ( aula_id, estudante_id ) ;
```

Figura 96 - Create e Alter da Tabela "presenca".

```
CREATE TABLE propina
(
    id          INTEGER NOT NULL ,
    matricula_id INTEGER NOT NULL ,
    numero_parcelas INTEGER NOT NULL ,
    estado      CHAR (1) NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE propina
ADD CONSTRAINT propina_PK PRIMARY KEY ( id ) ;
```

Figura 97 - Create e Alter da Tabela "propina".

```
CREATE TABLE recurso
(
    id      INTEGER NOT NULL ,
    nome    VARCHAR2 (100 CHAR) NOT NULL ,
    descricao VARCHAR2 (255 CHAR) ,
    turma_id INTEGER NOT NULL ,
    docente_id INTEGER NOT NULL ,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status    CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE recurso
ADD CONSTRAINT recurso_PK PRIMARY KEY ( id ) ;
```

Figura 98 - Create e Alter da Tabela "recurso".

```
CREATE TABLE tipo_aula
(
    id      INTEGER NOT NULL ,
    nome    VARCHAR2 (50 CHAR) NOT NULL ,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status    CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE tipo_aula
ADD CONSTRAINT tipo_aula_PK PRIMARY KEY ( id ) ;
```

Figura 99 - Create e Alter da Tabela "tipo_aula".

```
CREATE TABLE tipo_avaliacao
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    descricao   VARCHAR2 (255 CHAR) ,
    requer_entrega CHAR (1) NOT NULL ,
    permite_grupo CHAR (1) NOT NULL ,
    permite_filhos CHAR (1) NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE tipo_avaliacao
ADD CONSTRAINT tipo_avaliacao_PK PRIMARY KEY ( id ) ;
```

Figura 100 - Create e Alter da Tabela "tipo_avaliacao".

```
CREATE TABLE tipo_contacto
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE tipo_contacto
ADD CONSTRAINT tipo_contacto_PK PRIMARY KEY ( id ) ;
```

Figura 101 - Create e Alter da Tabela "tipo_contacto".

```
CREATE TABLE tipo_curso
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    valor_propinas NUMBER NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE tipo_curso
ADD CONSTRAINT tipo_curso_PK PRIMARY KEY ( id ) ;
```

Figura 102 - Create e Alter da Tabela "tipo_curso".

```
CREATE TABLE tipo_departamento
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (30 CHAR) NOT NULL ,
    descricao   VARCHAR2 (255 CHAR) ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE tipo_departamento
ADD CONSTRAINT tipo_departamento_PK PRIMARY KEY ( id ) ;
```

Figura 103 - Create e Alter da Tabela "tipo_departamento".

```
CREATE TABLE tipo_espaco
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    permite_aula CHAR (1) NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE tipo_espaco
ADD CONSTRAINT tipo_espaco_PK PRIMARY KEY ( id ) ;
```

Figura 104 - Create e Alter da Tabela "tipo_espaco".

```
CREATE TABLE turma
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (20 CHAR) NOT NULL ,
    ano_letivo   VARCHAR2 (10 CHAR) NOT NULL ,
    unidade_curricular_id INTEGER NOT NULL ,
    turno_aulas_id   INTEGER NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE turma
ADD CONSTRAINT turma_PK PRIMARY KEY ( id ) ;
```

Figura 105 - Create e Alter da Tabela "turma".

```
CREATE TABLE turma_docente
(
    turma_id      INTEGER NOT NULL ,
    funcao_docente_id INTEGER NOT NULL ,
    docente_id      INTEGER NOT NULL ,
    created_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status          CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE turma_docente
ADD CONSTRAINT turma_docente_PK PRIMARY KEY ( turma_id ) ;
```

Figura 106 - Create e Alter da Tabela "turma_docente".

```
CREATE TABLE turno
(
    id      INTEGER NOT NULL ,
    hora_inicio DATE NOT NULL ,
    hora_fim   DATE NOT NULL ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE turno
ADD CONSTRAINT turno_PK PRIMARY KEY ( id ) ;
```

Figura 107 - Create e Alter da Tabela "turno".

```
CREATE TABLE turno_aulas
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    hora_entrada INTEGER ,
    hora_saida   INTEGER ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE turno_aulas
ADD CONSTRAINT turno_aulas_PK PRIMARY KEY ( id ) ;
```

Figura 108 - Create e Alter da Tabela "turno_aulas".

```
CREATE TABLE uc_curso
(
    curso_id           INTEGER NOT NULL ,
    unidade_curricular_id INTEGER NOT NULL ,
    obrigatoria      CHAR (1) NOT NULL ,
    semestre          INTEGER NOT NULL ,
    ano               INTEGER NOT NULL ,
    ects              INTEGER NOT NULL ,
    presenca_obrigatoria CHAR (1) NOT NULL ,
    percentagem_presenca NUMBER ,
    created_at         TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at         TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status             CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE uc_curso
ADD CONSTRAINT unidade_curricular_curso_PK PRIMARY KEY ( curso_id, unidade_curricular_id ) ;
```

Figura 109 - Create e Alter da Tabela "uc_curso".

```

CREATE TABLE uc_departamento
(
    departamento_id      INTEGER NOT NULL ,
    unidade_curricular_id INTEGER NOT NULL ,
    created_at           TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at           TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status                CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE uc_departamento
ADD CONSTRAINT uc_departamento_PK PRIMARY KEY ( departamento_id, unidade_curricular_id ) ;

```

Figura 110 - Create e Alter da Tabela "uc_departamento".

```

CREATE TABLE uc_docente
(
    unidade_curricular_id INTEGER NOT NULL ,
    docente_id            INTEGER NOT NULL ,
    funcao                 VARCHAR2 (30 CHAR) ,
    created_at           TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at           TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status                CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE uc_docente
ADD CONSTRAINT unidade_curricular_docente_PK PRIMARY KEY ( unidade_curricular_id, docente_id ) ;

```

Figura 111 - Create e Alter da Tabela "uc_docente".

```

CREATE TABLE unidade_curricular
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (100 CHAR) NOT NULL ,
    codigo       VARCHAR2 (20 CHAR) NOT NULL ,
    horas_teoricas INTEGER NOT NULL ,
    horas_praticas INTEGER NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE unidade_curricular
ADD CONSTRAINT unidade_curricular_PK PRIMARY KEY ( id ) ;

```

Figura 112 - Create e Alter da Tabela "unidade_curricular".

```
CREATE TABLE utilizador
(
    id      INTEGER NOT NULL ,
    email   VARCHAR2 (50 CHAR) NOT NULL ,
    password VARCHAR2 (50 CHAR) NOT NULL ,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status   CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE utilizador
ADD CONSTRAINT utilizador_PK PRIMARY KEY ( id ) ;
```

Figura 113 - Create e Alter da Tabela "utilizador".

Como referido no início da explicação deste tópico, foi dito que de tabela em tabela o mecanismo de criação e alteração de tabelas, era exatamente o mesmo, definitivamente após estas figuras todas, de facto é possível comprovar isso mesmo.

Após a criação das tabelas e dos demais processos atrelados à criação das mesmas, agora é necessário continuar a fazer *ALTER TABLE* mas desta vez por causa das chaves estrangeiras.

```
ALTER TABLE andar
ADD CONSTRAINT andar_edificio_FK FOREIGN KEY
(
    edificio_id
)
REFERENCES edificio
(
    id
)
;
```

Figura 114 - Alter da Tabela "andar".

```
ALTER TABLE aula
    ADD CONSTRAINT aula_espaco_FK FOREIGN KEY
    (
        espaco_id
    )
    REFERENCES espaco
    (
        id
    )
;

ALTER TABLE aula
    ADD CONSTRAINT aula_tipo_aula_FK FOREIGN KEY
    (
        tipo_aula_id
    )
    REFERENCES tipo_aula
    (
        id
    )
;
```

Figura 115 - Alter da Tabela "aula".

Instituto Politécnico de Coimbra
Instituto Superior de Engenharia de Coimbra

```
ALTER TABLE avaliacao
    ADD CONSTRAINT avaliacao_avaliacao_FK FOREIGN KEY
    (
        avaliacao_pai_id
    )
    REFERENCES avaliacao
    (
        id
    )
;

ALTER TABLE avaliacao
    ADD CONSTRAINT avaliacao_tipo_avaliacao_FK FOREIGN KEY
    (
        tipo_avaliacao_id
    )
    REFERENCES tipo_avaliacao
    (
        id
    )
;

ALTER TABLE avaliacao
    ADD CONSTRAINT avaliacao_turma_FK FOREIGN KEY
    (
        turma_id
    )
    REFERENCES turma
    (
        id
    )
;
```

Figura 116 - Alter da Tabela "avaliacao".

```
ALTER TABLE colaborador
    ADD CONSTRAINT colaborador_utilizador_FK FOREIGN KEY
    (
        id
    )
    REFERENCES utilizador
    (
        id
    )
;
```

Figura 117 - Alter da Tabela "colaborador".

```
ALTER TABLE contacto_estudante
    ADD CONSTRAINT cont_estudante_tipo_FK FOREIGN KEY
    (
        tipo_contacto_id
    )
    REFERENCES tipo_contacto
    (
        id
    )
;
```

Figura 118 - Alter da Tabela "contacto_estudante".

```
ALTER TABLE curso_area
    ADD CONSTRAINT curso_area_area_FK FOREIGN KEY
    (
        area_id
    )
    REFERENCES area
    (
        id
    )
;

ALTER TABLE curso_area
    ADD CONSTRAINT curso_area_curso_FK FOREIGN KEY
    (
        curso_id
    )
    REFERENCES curso
    (
        id
    )
;
```

Figura 119 - Alter da Tabela "curso_area".

```
ALTER TABLE curso
    ADD CONSTRAINT curso_tipo_curso_FK FOREIGN KEY
    (
        tipo_curso_id
    )
    REFERENCES tipo_curso
    (
        id
    )
;
```

Figura 120 - Alter da Tabela "curso".

```
ALTER TABLE departamento
  ADD CONSTRAINT depart_tipo_depart_FK FOREIGN KEY
  (
    tipo_departamento_id
  )
  REFERENCES tipo_departamento
  (
    id
  )
;
```

Figura 121 - Alter da Tabela "departamento".

```
ALTER TABLE docente_aula
  ADD CONSTRAINT docente_aula_aula_FK FOREIGN KEY
  (
    aula_id
  )
  REFERENCES aula
  (
    id
  )
;

ALTER TABLE docente_aula
  ADD CONSTRAINT docente_aula_docente_FK FOREIGN KEY
  (
    docente_id
  )
  REFERENCES docente
  (
    id
  )
;
```

Figura 122 - Alter da Tabela "docente_aula".

```
ALTER TABLE docente
    ADD CONSTRAINT docente_colaborador_FK FOREIGN KEY
    (
        id
    )
    REFERENCES colaborador
    (
        id
    )
;

ALTER TABLE docente
    ADD CONSTRAINT docente_departamento_FK FOREIGN KEY
    (
        departamento_id
    )
    REFERENCES departamento
    (
        id
    )
;
```

Figura 123 - Alter da Tabela "docente".

```
ALTER TABLE edificio
    ADD CONSTRAINT edificio_polo_FK FOREIGN KEY
    (
        polo_id
    )
    REFERENCES polo
    (
        id
    )
;
```

Figura 124 - Alter da Tabela "edificio".

```
ALTER TABLE entrega
    ADD CONSTRAINT entrega_avaliacao_FK FOREIGN KEY
    (
        avaliacao_id
    )
    REFERENCES avaliacao
    (
        id
    )
;
;
```

Figura 125 - Alter da Tabela "entrega".

```
ALTER TABLE espaco
    ADD CONSTRAINT espaco_tipo_espaco_FK FOREIGN KEY
    (
        tipo_espaco_id
    )
    REFERENCES tipo_espaco
    (
        id
    )
;
;
```

Figura 126 - Alter da Tabela "espaco".

```
ALTER TABLE estudante_entrega
    ADD CONSTRAINT estudante_entrega_entrega_FK FOREIGN KEY
    (
        entrega_id
    )
    REFERENCES entrega
    (
        id
    )
;
ALTER TABLE estudante_entrega
    ADD CONSTRAINT estudante_entrega_estudante_FK FOREIGN KEY
    (
        estudante_id
    )
    REFERENCES estudante
    (
        id
    )
;
;
```

Figura 127 - Alter da Tabela "estudante_entrega".

```
ALTER TABLE estudante
  ADD CONSTRAINT estudante_utilizador_FK FOREIGN KEY
  (
    id
  )
  REFERENCES utilizador
  (
    id
  )
;
;
```

Figura 128 - Alter da Tabela "estudante".

```
ALTER TABLE ficheiro_entrega
  ADD CONSTRAINT ficheiro_entrega_FK FOREIGN KEY
  (
    entrega_id
  )
  REFERENCES entrega
  (
    id
  )
;
;

ALTER TABLE ficheiro_recurso
  ADD CONSTRAINT ficheiro_recurso_recurso_FK FOREIGN KEY
  (
    recurso_id
  )
  REFERENCES recurso
  (
    id
  )
;
;
```

Figura 129 - Alter das Tabelas "ficheiro_entrega" e "ficheiro_recurso".

```
ALTER TABLE gabinete_docente
    ADD CONSTRAINT gabinete_docente_docente_FK FOREIGN KEY
    (
        docente_id
    )
    REFERENCES docente
    (
        id
    )
;

ALTER TABLE gabinete_docente
    ADD CONSTRAINT gabinete_docente_espaco_FK FOREIGN KEY
    (
        espaco_id
    )
    REFERENCES espaco
    (
        id
    )
;
```

Figura 130 - Alter da Tabela "gabinete_docente".

```
ALTER TABLE horario_colaborador
    ADD CONSTRAINT horario_colaborador_horario_FK FOREIGN KEY
    (
        horario_id
    )
    REFERENCES horario
    (
        id
    )
;

ALTER TABLE horario_colaborador
    ADD CONSTRAINT horario_colaborador_n_doc_FK FOREIGN KEY
    (
        nao_docente_id
    )
    REFERENCES nao_docente
    (
        id
    )
;
```

Figura 131 - Alter da Tabela "horario_colaborador".

```
ALTER TABLE horario
    ADD CONSTRAINT horario_turno_FK FOREIGN KEY
    (
        turno_id
    )
    REFERENCES turno
    (
        id
    )
;
;
```

Figura 132 - Alter da Tabela "horario".

```
ALTER TABLE matricula
    ADD CONSTRAINT inscricao_curso_curso_FK FOREIGN KEY
    (
        curso_id
    )
    REFERENCES curso
    (
        id
    )
;
;

ALTER TABLE matricula
    ADD CONSTRAINT inscricao_curso_estudante_FK FOREIGN KEY
    (
        estudante_id
    )
    REFERENCES estudante
    (
        id
    )
;
;
```

Figura 133 - Alter da Tabela "matricula".

```
ALTER TABLE inscricao
    ADD CONSTRAINT inscricao_estado_inscricao_FK FOREIGN KEY
    (
        estado_inscricao_id
    )
    REFERENCES estado_inscricao
    (
        id
    )
;

ALTER TABLE inscricao
    ADD CONSTRAINT inscricao_turma_matricula_FK FOREIGN KEY
    (
        matricula_id
    )
    REFERENCES matricula
    (
        id
    )
;

ALTER TABLE inscricao
    ADD CONSTRAINT inscricao_turma_turma_FK FOREIGN KEY
    (
        turma_id
    )
    REFERENCES turma
    (
        id
    )
;
```

Figura 134 - Alter da Tabela "inscricao".



```
ALTER TABLE log
    ADD CONSTRAINT log_acao_log_FK FOREIGN KEY
    (
        acao_log_id
    )
    REFERENCES acao_log
    (
        id
    )
;

ALTER TABLE log
    ADD CONSTRAINT log_entidade_log_FK FOREIGN KEY
    (
        entidade_log_id
    )
    REFERENCES entidade_log
    (
        id
    )
;

ALTER TABLE log
    ADD CONSTRAINT log_utilizador_FK FOREIGN KEY
    (
        utilizador_id
    )
    REFERENCES utilizador
    (
        id
    )
;
```

Figura 135 - Alter da Tabela "log".

```
ALTER TABLE matricula
    ADD CONSTRAINT matricula_estado_matricula_FK FOREIGN KEY
    (
        estado_matricula_id
    )
    REFERENCES estado_matricula
    (
        id
    )
;
```

Figura 136 - Outro Alter da Tabela "matricula".

Instituto Politécnico de Coimbra
Instituto Superior de Engenharia de Coimbra

```
ALTER TABLE nao_docente
    ADD CONSTRAINT nao_docente_cargo_FK FOREIGN KEY
    (
        cargo_id
    )
    REFERENCES cargo
    (
        id
    )
;
;

ALTER TABLE nao_docente
    ADD CONSTRAINT nao_docente_colaborador_FK FOREIGN KEY
    (
        id
    )
    REFERENCES colaborador
    (
        id
    )
;
;

ALTER TABLE nao_docente
    ADD CONSTRAINT nao_docente_departamento_FK FOREIGN KEY
    (
        departamento_id
    )
    REFERENCES departamento
    (
        id
    )
;
;
```

Figura 137 - Alter da Tabela "nao_docente".

```
ALTER TABLE nota
    ADD CONSTRAINT nota_avaliacao_FK FOREIGN KEY
    (
        avaliacao_id
    )
    REFERENCES avaliacao
    (
        id
    )
;
;

ALTER TABLE nota
    ADD CONSTRAINT nota_estudante_FK FOREIGN KEY
    (
        estudante_id
    )
    REFERENCES estudante
    (
        id
    )
;
;
```

Figura 138 - Alter da Tabela "nota".

```
ALTER TABLE parcela_propina
    ADD CONSTRAINT parcela_propina_propina_FK FOREIGN KEY
    (
        propina_id
    )
    REFERENCES propina
    (
        id
    )
;

;
```

Figura 139 - Alter da Tabela "parcela_propina".

```
ALTER TABLE presenca
    ADD CONSTRAINT presenca_aula_FK FOREIGN KEY
    (
        aula_id
    )
    REFERENCES aula
    (
        id
    )
;

ALTER TABLE presenca
    ADD CONSTRAINT presenca_estudante_FK FOREIGN KEY
    (
        estudante_id
    )
    REFERENCES estudante
    (
        id
    )
;
```

Figura 140 - Alter da Tabela "presenca".

```
ALTER TABLE propina
    ADD CONSTRAINT propina_matricula_FK FOREIGN KEY
    (
        matricula_id
    )
    REFERENCES matricula
    (
        id
    )
;
```

Figura 141 - Alter da Tabela Propina.

```
ALTER TABLE recurso
    ADD CONSTRAINT recurso_docente_FK FOREIGN KEY
    (
        docente_id
    )
    REFERENCES docente
    (
        id
    )
;

ALTER TABLE recurso
    ADD CONSTRAINT recurso_turma_FK FOREIGN KEY
    (
        turma_id
    )
    REFERENCES turma
    (
        id
    )
;
```

Figura 142 - Alter da Tabela "recurso".

```
ALTER TABLE turma_docente
    ADD CONSTRAINT turma_docente_docente_FK FOREIGN KEY
    (
        docente_id
    )
    REFERENCES docente
    (
        id
    )
;

ALTER TABLE turma_docente
    ADD CONSTRAINT turma_docente_funcao_FK FOREIGN KEY
    (
        funcao_docente_id
    )
    REFERENCES funcao_docente
    (
        id
    )
;
```

Figura 143 - Alter da Tabela "turma_docente".

```
ALTER TABLE espaco
    ADD CONSTRAINT sala_andar_FK FOREIGN KEY
    (
        andar_id
    )
    REFERENCES andar
    (
        id
    )
;
```

Figura 144 - Outro Alter da Tabela "espaco".

```
ALTER TABLE turma
    ADD CONSTRAINT turma_turno_aulas_FK FOREIGN KEY
    (
        turno_aulas_id
    )
    REFERENCES turno_aulas
    (
        id
    )
;

ALTER TABLE turma
    ADD CONSTRAINT turma_unidade_curricular_FK FOREIGN KEY
    (
        unidade_curricular_id
    )
    REFERENCES unidade_curricular
    (
        id
    )
;
```

Figura 145 - Alter da Tabela "turma".

```
ALTER TABLE uc_curso
    ADD CONSTRAINT uc_curso_curso_FK FOREIGN KEY
    (
        curso_id
    )
    REFERENCES curso
    (
        id
    )
;

ALTER TABLE uc_curso
    ADD CONSTRAINT uc_curso_uc_FK FOREIGN KEY
    (
        unidade_curricular_id
    )
    REFERENCES unidade_curricular
    (
        id
    )
;
```

Figura 146 - Alter da Tabela "uc_curso".

```
ALTER TABLE uc_departamento
    ADD CONSTRAINT uc_depart_depart_FK FOREIGN KEY
    (
        departamento_id
    )
    REFERENCES departamento
    (
        id
    )
;

ALTER TABLE uc_departamento
    ADD CONSTRAINT uc_depart_uc_FK FOREIGN KEY
    (
        unidade_curricular_id
    )
    REFERENCES unidade_curricular
    (
        id
    )
;
```

Figura 147 - Alter da Tabela "uc_departamento".

```
ALTER TABLE uc_docente
    ADD CONSTRAINT uc_docente_docente_FK FOREIGN KEY
    (
        docente_id
    )
    REFERENCES docente
    (
        id
    )
;

ALTER TABLE turma_docente
    ADD CONSTRAINT uc_docente_turma_FK FOREIGN KEY
    (
        turma_id
    )
    REFERENCES turma
    (
        id
    )
;

ALTER TABLE uc_docente
    ADD CONSTRAINT uc_docente_uc_FK FOREIGN KEY
    (
        unidade_curricular_id
    )
    REFERENCES unidade_curricular
    (
        id
    )
;
```

Figura 148 - Alter da Tabela "uc_docente".

Desta forma, se explica um bocado mais facilmente os *ALTER TABLE*, bem como as cedências de chaves estrangeiras por parte de certas entidades a outras entidades.

Inserção de Dados

Devido à extensão do nosso sistema, nós optámos por fazer apenas duzentas inserções (*INSERT*) por entidade.

Como já referido anteriormente, a ferramenta usada para esta ocasião foi o *Mockaroo*.

Tivemos alguns problemas na inserção, tipicamente por que a ferramenta está em inglês e, portanto, gera dados falsos em inglês e sem estarem totalmente alienados com os atributos, como por exemplo, nos atributos que implicam introduzir nomes, a ferramenta *Mockaroo* achou por bem dar nomes de plantas e, portanto, este é um dos pequenos exemplos do quão mal a parte da transformação dos dados funcionou.

Conclusão – Primeira Fase

A conclusão será mais a título pessoal do grupo, se bem que em certas situações ao longo do documento, fomos intervindo para justificar certas coisas relativas ao projeto, deixando a escrita formal de lado.

Não consideramos de todo um projeto difícil, isto porque já tínhamos passado pela mesma situação no ano passado na disciplina de Bases de Dados I. O que custou mais e foi mais trabalhoso, foi a modelação em *Data Modeler* de um projeto tão extenso.

Ainda assim, fizemos o trabalho sabendo da proporção do mesmo e estamos prontos para acarretar com as possíveis consequências disso na próxima e última fase deste projeto, quando começarmos a englobar *triggers*, funções, procedimentos e cursores.

Podemos dizer que independentemente da apreciação que este nosso projeto possa vir a receber, nos esforçamos, soubemos dividir bem as tarefas, sabemos o que está feito e o que pode ter corrido menos bem e isso é certamente motivo de orgulho.

P.S. Este documento e consequente projeto dispensa a declaração de *webgrafia* ou bibliografia, visto que o presente documento, apenas explica a execução passo a passo do projeto.

Ademais é importante revelar, que este documento, foi submetido juntamente com os restantes ficheiros que complementam o projeto (modelo físico, *script ddl* e posterior ficheiro com os diversos dados inseridos no sistema de ensino superior).

Fase 2 - Sistema de Gestão de um Instituto de Ensino Superior



Mudanças de Paradigma

Na anterior fase, ou seja, na fase um do presente projeto foi relatado que “O nosso grupo escolheu o tema da universidade porque este tipo de sistema representa um cenário realista, complexo e altamente relevante no contexto atual. Universidades lidam diariamente com grandes volumes de dados relacionados a estudantes, professores, cursos, disciplinas, matrículas, avaliações, etc.

Ademais, é um tema com o qual todos temos familiaridade, o que facilita a compreensão dos requisitos e a definição das entidades e atributos necessários. Também consideramos que esta escolha proporciona oportunidades futuras de inclusão de outros conceitos aprendido na disciplina de Base de Dados II, como os triggers, funções, cursores e procedimentos.

Por fim, acreditamos que é uma excelente forma de desenvolver competências que poderão ser aplicadas em diversos contextos profissionais, dado que a gestão eficiente dos dados é uma necessidade presente em várias áreas.”.

O grupo continua a apostar no mesmo tema de projeto, mas desta vez com uma pequena mudança, ao invés de ser uma universidade/politécnico em andamento, o grupo decidiu que na verdade este é o primeiro ano de atividade do dito instituto superior. Esta decisão foi tomada por consequência da inserção de dados, para se tornar um cenário realista, seria natural haverem bastantes registo e ainda inserir novos registo relativos ao ano atual, mas por consequência do volume de registo e para obter um melhor desempenho computacional, ficou decidido que começar uma universidade do zero, seria o melhor caminho a se percorrer.

Outra mudança impactante foi realizada após a primeira fase. Por indicação/obrigação do docente, o grupo resolveu diminuir o número de tabelas que compõem o sistema, passando de 53 entidades para apenas 24 entidades, ficando assim o sistema reduzido, mas na mesma operacional, cobrindo a proposta de desenvolvimento original.

Mudanças no Modelo Físico

Devido à necessidade de realizar alterações no modelo físico do projeto, para deixar o projeto mais adequado ao requisitado pelo docente, foi necessário voltar à origem onde o desenvolvimento do projeto começou, ou seja, o modelo físico.

Se na primeira fase do projeto haviam 53 entidades, na segunda fase do projeto só há 24 entidades.

Devido ao volume gigantesco de dados inseridos o projeto iria ficar muito pesado e demandaria um poder de processamento e uso de memória maior do que provavelmente os computadores dos alunos poderiam aguentar.

Outro dos motivos para a mudança, foi o facto da quantidade exorbitante de operações PL/SQL que teriam de ser realizadas para pôr de pé tamanho projeto, devido à implementação de muitos *triggers*, muita indexação e sequenciação, o dobro do trabalho a realizar funções e procedimentos para manter tudo alienado com o objetivo do projeto.

A remoção de tantas entidades só foi possível graças a uma arquitetura modular, isto é, existem vários grupos de entidades que fazem mais ou menos parte de um grupo (formando módulos) e possuem relações entre elas, quase como se fosse uma arquitetura em estrela.

Posto isto, o projeto ficou com o seguinte modelo físico:

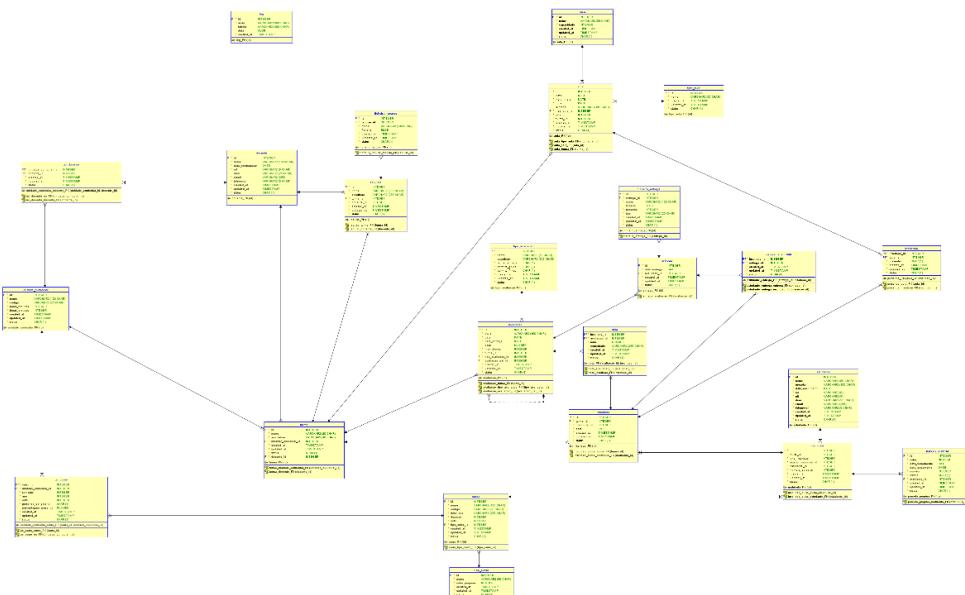


Figura 149 - Modelo Físico.

No modelo físico da página , é possível visualizar que ao contrário do modelo físico inicialmente desenvolvido na fase um e que tinha nove tipos de entidades, na fase dois do projeto e devido à drástica redução de entidades, agora são apenas seis tipos de entidades.

1. **Entidades Académicas Principais:** Estas representam os conceitos centrais do ensino, ou seja, tem uma estrutura mais pedagógica e curricular;
2. **Entidades de Pessoal:** Divididas entre docentes e não docentes, basicamente representam a gestão de recurso humanos da universidade;
3. **Entidades de Estudantes:** Mais focadas em dados pessoais e percurso académico;
4. **Entidades de Avaliação e Recursos Didáticos:** Neste caso, controlam as avaliações e materiais de apoio;
5. **Entidades Financeiras:** Relacionadas a propinas e pagamentos, comportam a gestão financeira;
6. **Entidades de Infraestrutura:** Isto é, relacionadas com espaços e edifícios, no fundo a infraestrutura física;

Resumindo por tipo, existem oito entidades que compõem o módulo referente a entidades académicas principais, uma tabela referente ao módulo de entidades de pessoal, quatro entidades que compõem o módulo de entidades relacionadas a estudantes, oito entidades que comportam o módulo de entidades de avaliação e recursos didáticos, uma entidade do módulo financeiro, uma entidade relacionada a infraestrutura e uma entidade chamada “logs”, que não se enquadra nas demais citadas acima, mas serve mais para controlo e auditoria.

O modelo físico, sofreu também algumas mudanças quanto aos atributos que compõem cada uma das entidades, alguns atributos foram removidos por serem considerados redundantes, enquanto que outros atributos sofreram apenas uma mudança do tipo de dados (ex: um *integer* passar a *money*). Existem também casos em que algumas entidades herdaram atributos de outras entidades que foram suprimidas no decorrer do desenvolvimento do projeto.

Com a supressão de certas entidades, alguns relacionamentos também mudaram (algo abordado mais à frente neste documento).

Entidades ao Pormenor - Mudanças

Neste tópico, fala-se das entidades, mas somente as que de alguma forma sofreram alterações da primeira para a segunda fase do projeto. Todas as entidades que possuírem a legenda “Sem alterações” são também referenciadas com uma imagem correspondente, embora não tenham sofrido quaisquer objeções/alterações.

Entidades Académicas Principais

Entidade “curso”: Sem alterações.

CURSO	
P	* id INTEGER
	* nome VARCHAR2 (100 CHAR)
	* codigo VARCHAR2 (20 CHAR)
	* descricao VARCHAR2 (255 CHAR)
	* duracao INTEGER
	* ects INTEGER
F	* tipo_curso_id INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
» curso_PK (id)	
» curso_tipo_curso_FK (tipo_curso_id)	

Figura 150 - Entidade "curso".

Entidade “tipo_curso”: Sem alterações.

tipo_curso	
P	* id INTEGER
	* nome VARCHAR2 (50 CHAR)
	* valor_propinas NUMBER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
» tipo_curso_PK (id)	

Figura 151 - Entidade "tipo_curso".

Entidade “unidade_curricular”: Sem alterações.

unidade_curricular	
P	* id INTEGER
	* nome VARCHAR2 (100 CHAR)
	* codigo VARCHAR2 (20 CHAR)
	* horas_teoricas INTEGER
	* horas_praticas INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
» unidade_curricular_PK (id)	

Figura 152 - Entidade "unidade_curricular".

Entidade “uc_curso”: O atributo “obrigacao” foi removido da tabela. Não existem mais alterações.

uc_curso	
PF*	curso_id INTEGER
PF*	unidade_curricular_id INTEGER
	* semestre INTEGER
	* ano INTEGER
	* ects INTEGER
	* presenca_obrigatoria CHAR (1)
	percentagem_presenca NUMBER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
» unidade_curricular_curso_PK (curso_id, unidade_curricular_id)	
» uc_curso_curso_FK (curso_id)	
» uc_curso_ue_FK (unidade_curricular_id)	

Figura 153 - Entidade "uc_curso".

Entidade “turma”: A chave estrangeira “turno_aulas_id” foi removida, no entanto a chave estrangeira “docente_id” foi adicionada, o que indica uma ligação desta tabela com a tabela “docente”.

turma	
P	* id INTEGER
	* nome VARCHAR2 (20 CHAR)
	* ano_letivo VARCHAR2 (10 CHAR)
F	* unidade_curricular_id INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
F	* docente_id INTEGER
» turma_PK (id)	
» turma_unidade_curricular_FK (unidade_curricular_id)	
» turma_docente_FK (docente_id)	

Figura 154 - Entidade "turma".

Entidade “aula”: A chave estrangeira “espaço_id” foi removida, mas duas novas chaves estrangeiras foram adicionadas a esta tabela, a “sala_id” e a “turma_id”.

aula	
P	* id
	INTEGER
*	data
	DATE
*	hora_inicio
	DATE
*	hora_fim
	DATE
	sumario
	VARCHAR2 (255 CHAR)
F	* tipo_aula_id
	INTEGER
F	* sala_id
	INTEGER
F	* turma_id
	INTEGER
	* created_at
	TIMESTAMP
	* updated_at
	TIMESTAMP
	* status
	CHAR (1)
» aula_PK (id)	
» aula_tipo_aula_FK (tipo_aula_id)	
» aula_sala_FK (sala_id)	
» aula_turma_FK (turma_id)	

Figura 155 - Entidade "aula".

Entidade “tipo_aula”: Sem alterações.

tipo_aula	
P	* id
	INTEGER
*	nome
	VARCHAR2 (50 CHAR)
*	created_at
	TIMESTAMP
*	updated_at
	TIMESTAMP
	* status
	CHAR (1)
» tipo_aula_PK (id)	

Figura 156 - Entidade "tipo_aula".

Entidade “uc_docente”: O atributo “funcao” foi removido. Esta entidade não regista mais alterações.

uc_docente	
PF	* unidade_curricular_id
PF	* docente_id
*	created_at
*	TIMESTAMP
*	updated_at
	TIMESTAMP
	* status
	CHAR (1)
» unidade_curricular_docente_PK (unidade_curricular_id, docente_id)	
» uc_docente_ue_FK (unidade_curricular_id)	
» uc_docente_docente_FK (docente_id)	

Figura 157 - Entidade "uc_docente".

Entidade de Pessoal

Entidade “docente”: No caso da entidade “docente”, a chave estrangeira “departamento_id” foi removida, mas outros campos foram herdados de outras tabelas suprimidas, tais como: “nome, data_contratacao, nif, iban, email e telemóvel”.

docente	
P	* id
	INTEGER
	* nome
	VARCHAR2 (100 CHAR)
	* data_contratacao
	DATE
	* nif
	VARCHAR2 (9 CHAR)
	iban
	VARCHAR2 (25 CHAR)
	* email
	VARCHAR2 (255)
	* telemovel
	VARCHAR2 (9 CHAR)
	* created_at
	TIMESTAMP
	* updated_at
	TIMESTAMP
	* status
	CHAR (1)
► docente_PK (id)	

Figura 158 - Entidade "docente".

Entidade de Estudantes

Entidade “estudante”: Relativamente a esta entidade, foram apenas adicionados dois novos campos, sendo eles o “telemóvel” e “email”.

estudante	
P	* id
	INTEGER
	* nome
	VARCHAR2 (50 CHAR)
	morada
	VARCHAR2 (100 CHAR)
	* data_nascimento
	DATE
	* cc
	VARCHAR2 (8)
	* nif
	VARCHAR2 (9)
	iban
	VARCHAR2 (25 CHAR)
	* email
	VARCHAR2 (255)
	* telemovel
	VARCHAR2 (9 CHAR)
	* created_at
	TIMESTAMP
	* updated_at
	TIMESTAMP
	* status
	CHAR (1)
► estudante_PK (id)	

Figura 159 - Entidade "estudante".

Entidade “matricula”: O atributo “numero_parcelas” foi adicionado. O atributo “estado_matricula_id” é suposto já não estar como atributo uma vez que a tabela “estado_matricula” foi eliminada.

matricula	
P	* id
F	* curso_id
	* ano_inscricao
	* estado_matricula_id
F	* estudante_id
	* numero_parcelas
	* created_at
	* updated_at
	* status
matricula_PK (id)	
inscricao_curso_curso_FK (curso_id)	
inscricao_estudante_FK (estudante_id)	

Figura 160 - Entidade "matricula".

Entidade “inscricao”: Esta entidade passou a possuir uma chave primária própria (denominada “id”), a antiga chave primária (“turma_id”) passou a ser apenas uma chave estrangeira.

inscricao	
P	* id
F	* turma_id
F	* matricula_id
	* data
	* created_at
	* updated_at
	* status
inscricao_PK (id)	
inscricao_turma_turma_FK (turma_id)	
inscricao_matricula_matricula_FK (matricula_id)	

Figura 161 - Entidade "inscricao".

Entidade “presenca”: Esta entidade possuía “estudante_id” e “aula_id” como chaves primárias e estrangeiras ao mesmo tempo, atualmente possui “inscricao_id” e “aula_id”, isto significa que a identificação única de cada “presença” é feita pela combinação do aluno (“inscricao_id”) e da aula (“aula_id”). Mas também indica que são chaves estrangeiras, porque estes campos são usados para criar a relação com outras tabelas.

presenca	
P	* inscricao_id INTEGER
P	* aula_id INTEGER
	* presente CHAR (1)
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
presenca_PK (aula_id, inscricao_id)	
presenca_aula_FK (aula_id)	
presenca_inscricao_FK (inscricao_id)	

Figura 162 - Entidade "presenca".

Entidade de Avaliação e Recursos Didáticos

Entidade “avaliacao”: Sem alterações.

avaliacao	
P	* id INTEGER
	* titulo VARCHAR2 (100 CHAR)
	* data DATE
	* data_entrega DATE
	* peso NUMBER
	* max_alunos INTEGER
F	* turma_id INTEGER
F	* tipo_avaliacao_id INTEGER
F	avaliacao_pai_id INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
avaliacao_PK (id)	
avaliacao_turma_FK (turma_id)	
avaliacao_tipo_avaliacao_FK (tipo_avaliacao_id)	
avaliacao_avaliacao_FK (avaliacao_pai_id)	



Figura 163 - Entidade "avaliacao".

Entidade “tipo_avaliacao”: Sem alterações.

tipo_avaliacao	
P	* id INTEGER
	* nome VARCHAR2 (50 CHAR)
	descricao VARCHAR2 (255 CHAR)
	* requer_entrega CHAR (1)
	* permite_grupo CHAR (1)
	* permite_filhos CHAR (1)
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
> tipo_avaliacao_PK (id)	

Figura 164 - Entidade "tipo_avaliacao".

Entidade “nota”: À semelhança da entidade “presenca”, esta sofre a mesma mudança, anteriormente as *primary/foreign keys* eram “estudante_id” e “avaliacao_id”, atualmente são “inscricao_id” e “avaliacao_id”.

nota	
PF*	inscricao_id INTEGER
PF*	avaliacao_id INTEGER
	* nota NUMBER
	comentario VARCHAR2 (255 CHAR)
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
> nota_PK (avaliacao_id, inscricao_id)	
> nota_avaliacao_FK (avaliacao_id)	
> nota_inscricao_FK (inscricao_id)	

Figura 165 - Entidade "nota".

Entidade “entrega”: Sem alterações.

entrega	
P	* id INTEGER
	* data_entrega DATE
F	* avaliacao_id INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
> entrega_PK (id)	
> entrega_avaliacao_FK (avaliacao_id)	

Figura 166 - Entidade "entrega".

Entidade “estudante_entrega”: Também sofreu uma alteração de *primary/foreign key*, na fase um tinha “estudante_id” e “entrega_id” e na fase dois passa a ter “inscrição_id” e “entrega_id”.

estudante_entrega	
P*	inscricao_id INTEGER
P*	entrega_id INTEGER
*	created_at TIMESTAMP
*	updated_at TIMESTAMP
*	status CHAR (1)
↳ estudante_entrega_PK(entrega_id, inscricao_id)	
↳ estudante_entrega_entrega_FK (entrega_id)	
↳ estudante_entrega_inscricao_FK (inscricao_id)	

Figura 167 - Entidade "estudante_entrega".

Entidade “ficheiro_entrega”: Sem alterações.

ficheiro_entrega	
P *	id INTEGER
F *	entrega_id INTEGER
*	nome VARCHAR2 (100 CHAR)
	ficheiro BLOB
*	tamanho INTEGER
*	tipo VARCHAR2 (20 CHAR)
*	created_at TIMESTAMP
*	updated_at TIMESTAMP
*	status CHAR (1)
↳ ficheiro_entrega_PK (id)	
↳ ficheiro_entrega_FK (entrega_id)	

Figura 168 - Entidade "ficheiro_entrega".

Entidade “recurso”: Sem alterações.

recurso	
P *	id INTEGER
*	nome VARCHAR2 (100 CHAR)
	descricao VARCHAR2 (255 CHAR)
F *	turma_id INTEGER
F *	docente_id INTEGER
*	created_at TIMESTAMP
*	updated_at TIMESTAMP
*	status CHAR (1)
↳ recurso_PK (id)	
↳ recurso_turma_FK (turma_id)	
↳ recurso_docente_FK (docente_id)	

Figura 169 - Entidade "recurso".

Entidade “ficheiro_recurso”: Sem alterações.

ficheiro_recurso	
P	* id INTEGER
F	* recurso_id INTEGER
	* nome VARCHAR2 (100 CHAR)
	ficheiro BLOB
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
↳ ficheiro_recurso_PK (id)	
↳ ficheiro_recurso_FK (recurso_id)	

Figura 170 - Entidade "ficheiro_recurso".

Entidades Financeiras

Entidade “parcela_propina”: A única alteração nesta tabela foi a mudança de chave estrangeira, anteriormente estava “propina_id”, atualmente está “matricula_id”, indicando que a tabela já não se relaciona com a tabela “propina” (suprimida).

parcela_propina	
P	* id INTEGER
	* valor NUMBER
	* data_vencimento DATE
	data_pagamento DATE
	* número INTEGER
	* estado CHAR (1)
F	* matricula_id INTEGER
	* created_at TIMESTAMP
	* updated_at TIMESTAMP
	* status CHAR (1)
↳ parcela_propina_PK (id)	
↳ parcela_propina_matricula_FK (matricula_id)	

Figura 171 - Entidade "parcela_propina".

Entidades de Infraestrutura

Entidade “sala”: As entidades que na fase um do projeto compunham as entidades de infraestrutura foram todas suprimidas e convertidas numa só entidade denominada “sala”.

sala	
P	• id
	INTEGER
• nome	VARCHAR2 (50 CHAR)
• capacidade	INTEGER
• created_at	TIMESTAMP
• updated_at	TIMESTAMP
• status	CHAR (1)
» sala_PK (id)	

Figura 172 - Entidade "sala".

Entidades de Utilidade Geral

Por fim a entidade “log” é considerada uma tabela de utilidade geral e de auditoria visto que a mesma guarda todas e quaisquer alterações e operações feitas no sistema todo. Anteriormente este módulo de logs possuía três tabelas, atualmente possui só uma.

log	
P	• id
	INTEGER
• acao	VARCHAR2 (100 CHAR)
• tabela	VARCHAR2 (100 CHAR)
• data	CLOB
• created_at	TIMESTAMP
» log_PK (id)	

Figura 173 - Entidade "log".

Relações do Modelo Físico - Mudanças

Ainda relativamente ao modelo físico, sabemos que é um modelo gigante e complexo, outrora não fosse a tentativa de representação real de um sistema de gestão de uma universidade.

Como tal e apesar das mudanças, surgiu a necessidade de alterar certos relacionamentos e a cardinalidade dos mesmos. Abaixo, segue uma tabela descritiva de todas as relações presentes no modelo, bem como as entidades que se relacionam e a cardinalidade das mesmas.

Entidade	Cardinalidade	Entidade Relacionada
aula	N:1	tipo_aula
aula	N:1	sala
aula	N:1	turma
avaliacao	N:1	turma
avaliacao	N:1	tipo_avaliacao
avaliacao	N:1(Não Obrigatório)	avaliacao (auto-relacionamento)
curso	N:1	tipo_curso
entrega	N:1	avaliacao
estudante_entrega	N:1	entrega
estudante_entrega	N:1	inscricao
ficheiro_entrega	N:1	entrega
ficheiro_recurso	N:1	recurso
matricula	N:1	curso
matricula	N:1	estudante
inscricao	N:1	matricula
inscricao	N:1	turma
nota	N:1	inscricao
nota	N:1	avaliacao
parcela_propina	N:1	matricula
presenca	N:1	aula
presenca	N:1	inscricao
recurso	N:1	docente
recurso	N:1	turma
turma	N:1	docente
turma	N:1	unidade_curricular
uc_curso	N:1	curso
uc_curso	N:1	unidade_curricular
uc_docente	N:1	docente
uc_docente	N:1	Unidade_curricular

Data Definition Language – DDL

Uma vez mais, por consequência da diminuição do projeto, também o DDL sofreu alterações.

Na primeira fase do projeto foram cerca de 1660 linhas de código DDL, atualmente esse número situa-se nas 792 linhas de código. As seguintes imagens que compõem este tópico denotam o novo *script* DDL já com as devidas reduções no sistema da universidade.

De referir ainda, que devido à extensão do *script* DDL, a figura seguinte, representa uma pequena parte das tabelas. O restante *script* acompanha o relatório aquando da sua submissão.

```
8      DROP TABLE aula CASCADE CONSTRAINTS
9      ;
10
11     DROP TABLE avaliacao CASCADE CONSTRAINTS
12     ;
13
14     DROP TABLE curso CASCADE CONSTRAINTS
15     ;
16
17     DROP TABLE docente CASCADE CONSTRAINTS
18     ;
19
20     DROP TABLE entrega CASCADE CONSTRAINTS
21     ;
22
23     DROP TABLE estudante CASCADE CONSTRAINTS
24     ;
25
26     DROP TABLE estudante_entrega CASCADE CONSTRAINTS
27     ;
28
29     DROP TABLE ficheiro_entrega CASCADE CONSTRAINTS
30     ;
31
32     DROP TABLE ficheiro_recurso CASCADE CONSTRAINTS
33     ;
34
35     DROP TABLE inscricao CASCADE CONSTRAINTS
36     ;
```

Figura 174 - *DROP* de Tabelas.

Logo após o *DROP*, foi possível arrancar para a criação da primeira tabela, no caso a tabela “aula”, é importante relembrar que o DDL quando está a ser desenvolvido, normalmente começa-se a criar tabelas pela ordem de eventos e dependências.

Instituto Politécnico de Coimbra
Instituto Superior de Engenharia de Coimbra

No caso, não foi isso que aconteceu, isto porque preferimos uma abordagem diferente (ordem alfabética) para não nos perdemos.

```
CREATE TABLE aula
(
    id          INTEGER NOT NULL ,
    data        DATE NOT NULL ,
    hora_inicio DATE NOT NULL ,
    hora_fim   DATE NOT NULL ,
    sumario    VARCHAR2 (255 CHAR) ,
    tipo_aula_id INTEGER NOT NULL ,
    sala_id     INTEGER NOT NULL ,
    turma_id   INTEGER NOT NULL ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE aula
ADD CONSTRAINT aula_PK PRIMARY KEY ( id ) ;
```

Figura 175 - CREATE e ALTER da Tabela “aula”.

```
CREATE TABLE avaliacao
(
    id          INTEGER NOT NULL ,
    titulo      VARCHAR2 (100 CHAR) NOT NULL ,
    data        DATE NOT NULL ,
    data_entrega DATE ,
    peso        NUMBER ,
    max_alunos INTEGER NOT NULL ,
    turma_id   INTEGER NOT NULL ,
    tipo_avaliacao_id INTEGER NOT NULL ,
    avaliacao_pai_id INTEGER ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE avaliacao
ADD CONSTRAINT avaliacao_PK PRIMARY KEY ( id ) ;
```

Figura 176 - CREATE e ALTER da Tabela “avaliacao”.

À semelhança da primeira fase do projeto, o DDL segue a mesma execução de CREATE e ALTER. O *ALTER*, que por meio de uma restrição (*CONSTRAINT*), define a chave primária da tabela como sendo o campo “id”. Isto garante que esse campo contenha valores únicos e não nulos, assegurando a identificação exclusiva de cada registo.

De modo geral todas a tabelas presentes no nosso *script* DDL, foram criadas da mesma forma e com os seus respetivos atributos, alteradas/modificadas usando os mesmos comandos/cláusulas, pelo que se entende que mais explicações não serão propriamente necessárias.

```
CREATE TABLE curso
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (100 CHAR) NOT NULL ,
    codigo      VARCHAR2 (20 CHAR) NOT NULL ,
    descricao   VARCHAR2 (255 CHAR) NOT NULL ,
    duracao     INTEGER NOT NULL ,
    ects        INTEGER NOT NULL ,
    max_alunos  INTEGER ,
    tipo_curso_id INTEGER NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE curso
ADD CONSTRAINT curso_PK PRIMARY KEY ( id ) ;
```

Figura 177 - *CREATE* e *ALTER* da Tabela "curso".

```
CREATE TABLE docente
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (100 CHAR) NOT NULL ,
    data_contratacao DATE NOT NULL ,
    cc          VARCHAR2 (12 CHAR) ,
    nif         VARCHAR2 (9 CHAR) NOT NULL ,
    iban        VARCHAR2 (25 CHAR) ,
    email       VARCHAR2 (255) NOT NULL ,
    telemovel   VARCHAR2 (9 CHAR) NOT NULL ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE docente
ADD CONSTRAINT docente_PK PRIMARY KEY ( id ) ;
```

Figura 178 - CREATE e ALTER da Tabela "docente".

```
CREATE TABLE entrega
(
    id          INTEGER NOT NULL ,
    data_entrega DATE NOT NULL ,
    avaliacao_id INTEGER NOT NULL ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE entrega
ADD CONSTRAINT entrega_PK PRIMARY KEY ( id ) ;
```

Figura 179 - CREATE e ALTER da Tabela "entrega".

```
CREATE TABLE estudante
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    morada      VARCHAR2 (100 CHAR) ,
    data_nascimento DATE NOT NULL ,
    codigo      VARCHAR2 (10 CHAR) ,
    cc          VARCHAR2 (12 CHAR) NOT NULL ,
    nif         VARCHAR2 (9) NOT NULL ,
    iban         VARCHAR2 (25 CHAR) ,
    email        VARCHAR2 (255) NOT NULL ,
    telemovel   VARCHAR2 (9 CHAR) NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE estudante
ADD CONSTRAINT estudante_PK PRIMARY KEY ( id );
```

Figura 180 - CREATE e ALTER da Tabela "estudante".

```
CREATE TABLE estudante_entrega
(
    inscricao_id INTEGER NOT NULL ,
    entrega_id   INTEGER NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE estudante_entrega
ADD CONSTRAINT estudante_entrega_PK PRIMARY KEY ( entrega_id, inscricao_id ) ;
```

Figura 181 - CREATE e ALTER da Tabela "estudante_entrega".

```
CREATE TABLE ficheiro_entrega
(
    id          INTEGER NOT NULL ,
    entrega_id INTEGER NOT NULL ,
    nome        VARCHAR2 (100 CHAR) NOT NULL ,
    ficheiro    BLOB ,
    tamanho     INTEGER NOT NULL ,
    tipo        VARCHAR2 (20 CHAR) NOT NULL ,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE ficheiro_entrega
    ADD CONSTRAINT ficheiro_entrega_PK PRIMARY KEY ( id ) ;
```

Figura 182 - CREATE e ALTER da Tabela "ficheiro_entrega".

```
CREATE TABLE ficheiro_recurso
(
    id          INTEGER NOT NULL ,
    recurso_id INTEGER NOT NULL ,
    nome        VARCHAR2 (100 CHAR) NOT NULL ,
    ficheiro    BLOB ,
    tamanho     INTEGER ,
    tipo        VARCHAR2 (20 CHAR) ,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE ficheiro_recurso
    ADD CONSTRAINT ficheiro_recurso_PK PRIMARY KEY ( id ) ;
```

Figura 183 - CREATE e ALTER da Tabela "ficheiro_recurso".

```
CREATE TABLE inscricao
(
    id          INTEGER NOT NULL ,
    turma_id    INTEGER NOT NULL ,
    matricula_id INTEGER NOT NULL ,
    data        DATE NOT NULL ,
    nota_final  NUMBER ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE inscricao
    ADD CONSTRAINT inscricao_PK PRIMARY KEY ( id ) ;
```

Figura 184 - CREATE e ALTER da Tabela "inscricao".

```
CREATE TABLE log
(
    id          INTEGER NOT NULL ,
    acao        VARCHAR2 (100 CHAR) NOT NULL ,
    tabela     VARCHAR2 (100 CHAR) ,
    data        CLOB ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL
)
;

ALTER TABLE log
    ADD CONSTRAINT log_PK PRIMARY KEY ( id ) ;
```

Figura 185 - CREATE e ALTER da Tabela "log".



```
CREATE TABLE matricula
(
    id          INTEGER NOT NULL ,
    curso_id    INTEGER NOT NULL ,
    ano_inscricao INTEGER NOT NULL ,
    estudante_id INTEGER NOT NULL ,
    estado_matricula VARCHAR2 (50 CHAR) NOT NULL ,
    numero_parcelas INTEGER NOT NULL ,
    media_geral   NUMBER ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE matricula
ADD CONSTRAINT matricula_PK PRIMARY KEY ( id ) ;
```

Figura 186 - CREATE e ALTER da Tabela "matricula".

```
CREATE TABLE nota
(
    inscricao_id INTEGER NOT NULL ,
    avaliacao_id INTEGER NOT NULL ,
    nota        NUMBER NOT NULL ,
    comentario  VARCHAR2 (255 CHAR) ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE nota
ADD CONSTRAINT nota_PK PRIMARY KEY ( avaliacao_id, inscricao_id ) ;
```

Figura 187 - CREATE e ALTER da Tabela "nota".



```
CREATE TABLE parcela_propina
(
    id          INTEGER NOT NULL ,
    valor        NUMBER NOT NULL ,
    data_vencimento DATE NOT NULL ,
    data_pagamento DATE ,
    numero       INTEGER NOT NULL ,
    estado        CHAR (1) NOT NULL ,
    matricula_id INTEGER NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE parcela_propina
ADD CONSTRAINT parcela_propina_PK PRIMARY KEY ( id ) ;
```

Figura 188 - CREATE e ALTER da Tabela "parcela_propina".

```
CREATE TABLE presenca
(
    inscricao_id INTEGER NOT NULL ,
    aula_id      INTEGER NOT NULL ,
    presente     CHAR (1) NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status        CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE presenca
ADD CONSTRAINT presenca_PK PRIMARY KEY ( aula_id, inscricao_id ) ;
```

Figura 189 - CREATE e ALTER da Tabela "presenca".

```
CREATE TABLE recurso
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (100 CHAR) NOT NULL ,
    descricao   VARCHAR2 (255 CHAR) ,
    turma_id    INTEGER NOT NULL ,
    docente_id  INTEGER NOT NULL ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE recurso
ADD CONSTRAINT recurso_PK PRIMARY KEY ( id ) ;
```

Figura 190 - CREATE e ALTER da Tabela "recurso".

```
CREATE TABLE sala
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    capacidade INTEGER NOT NULL ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE sala
ADD CONSTRAINT sala_PK PRIMARY KEY ( id ) ;
```

Figura 191 - CREATE e ALTER da Tabela "sala".

```
CREATE TABLE tipo_aula
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE tipo_aula
ADD CONSTRAINT tipo_aula_PK PRIMARY KEY ( id ) ;
```

Figura 192 - CREATE e ALTER da Tabela "tipo_aula".

```
CREATE TABLE tipo_avaliacao
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    descricao   VARCHAR2 (255 CHAR) ,
    requer_entrega CHAR (1) NOT NULL ,
    permite_grupo CHAR (1) NOT NULL ,
    permite_filhos CHAR (1) NOT NULL ,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status      CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE tipo_avaliacao
ADD CONSTRAINT tipo_avaliacao_PK PRIMARY KEY ( id ) ;
```

Figura 193 - CREATE e ALTER da Tabela "tipo_avaliacao".



```
CREATE TABLE tipo_curso
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (50 CHAR) NOT NULL ,
    valor_propinas NUMBER NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE tipo_curso
ADD CONSTRAINT tipo_curso_PK PRIMARY KEY ( id ) ;
```

Figura 194 - CREATE e ALTER da Tabela "tipo_curso".

```
CREATE TABLE turma
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (20 CHAR) NOT NULL ,
    ano_letivo   VARCHAR2 (10 CHAR) NOT NULL ,
    unidade_curricular_id INTEGER NOT NULL ,
    max_alunos   INTEGER ,
    docente_id   INTEGER NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE turma
ADD CONSTRAINT turma_PK PRIMARY KEY ( id ) ;
```

Figura 195 - CREATE e ALTER da Tabela "turma".

```
CREATE TABLE uc_curso
(
    curso_id          INTEGER NOT NULL ,
    unidade_curricular_id INTEGER NOT NULL ,
    semestre          INTEGER NOT NULL ,
    ano               INTEGER NOT NULL ,
    ects              INTEGER NOT NULL ,
    presenca_obrigatoria CHAR (1) NOT NULL ,
    percentagem_presenca NUMBER ,
    created_at        TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at        TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status             CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE uc_curso
ADD CONSTRAINT unidade_curricular_curso_PK PRIMARY KEY ( curso_id, unidade_curricular_id ) ;
```

Figura 196 - CREATE e ALTER da Tabela "uc_curso".

```
CREATE TABLE uc_docente
(
    unidade_curricular_id INTEGER NOT NULL ,
    docente_id            INTEGER NOT NULL ,
    created_at            TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at            TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status                CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE uc_docente
ADD CONSTRAINT unidade_curricular_docente_PK PRIMARY KEY ( unidade_curricular_id, docente_id ) ;
```

Figura 197 - CREATE e ALTER da Tabela "uc_docente".

```

CREATE TABLE unidade_curricular
(
    id          INTEGER NOT NULL ,
    nome        VARCHAR2 (100 CHAR) NOT NULL ,
    codigo      VARCHAR2 (20 CHAR) NOT NULL ,
    horas_teoricas INTEGER NOT NULL ,
    horas_praticas INTEGER NOT NULL ,
    created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    updated_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    status       CHAR (1) DEFAULT '1' NOT NULL
)
;

ALTER TABLE unidade_curricular
    ADD CONSTRAINT unidade_curricular_PK PRIMARY KEY ( id ) ;

```

Figura 198 - CREATE e ALTER da Tabela "unidade_curricular".

Após a criação das tabelas e dos demais processos atrelados à criação das mesmas, agora é necessário continuar a fazer *ALTER TABLE* mas desta vez por causa das chaves estrangeiras.

```

ALTER TABLE aula
    ADD CONSTRAINT aula_sala_FK FOREIGN KEY
    (
        sala_id
    )
    REFERENCES sala
    (
        id
    )
;

ALTER TABLE aula
    ADD CONSTRAINT aula_tipo_aula_FK FOREIGN KEY
    (
        tipo_aula_id
    )
    REFERENCES tipo_aula
    (
        id
    )
;

ALTER TABLE aula
    ADD CONSTRAINT aula_turma_FK FOREIGN KEY
    (
        turma_id
    )
    REFERENCES turma
    (
        id
    )
;

```

Figura 199 - ALTER da Tabela "aula".

```
ALTER TABLE avaliacao
    ADD CONSTRAINT avaliacao_avaliacao_FK FOREIGN KEY
    (
        avaliacao_pai_id
    )
    REFERENCES avaliacao
    (
        id
    )
;

ALTER TABLE avaliacao
    ADD CONSTRAINT avaliacao_tipo_avaliacao_FK FOREIGN KEY
    (
        tipo_avaliacao_id
    )
    REFERENCES tipo_avaliacao
    (
        id
    )
;

ALTER TABLE avaliacao
    ADD CONSTRAINT avaliacao_turma_FK FOREIGN KEY
    (
        turma_id
    )
    REFERENCES turma
    (
        id
    )
;
```

Figura 200 - ALTER da Tabela "avaliacao".



```
ALTER TABLE curso
    ADD CONSTRAINT curso_tipo_curso_FK FOREIGN KEY
    (
        tipo_curso_id
    )
    REFERENCES tipo_curso
    (
        id
    )
;

ALTER TABLE entrega
    ADD CONSTRAINT entrega_avaliacao_FK FOREIGN KEY
    (
        avaliacao_id
    )
    REFERENCES avaliacao
    (
        id
    )
;
```

Figura 201 - ALTER das Tabelas "curso" e "entrega".

```
ALTER TABLE estudante_entrega
    ADD CONSTRAINT estudante_entrega_entrega_FK FOREIGN KEY
    (
        entrega_id
    )
    REFERENCES entrega
    (
        id
    )
;

ALTER TABLE estudante_entrega
    ADD CONSTRAINT estudante_entrega_inscricao_FK FOREIGN KEY
    (
        inscricao_id
    )
    REFERENCES inscricao
    (
        id
    )
;
```

Figura 202 - ALTER da Tabela "estudante_entrega".

```
ALTER TABLE ficheiro_entrega
    ADD CONSTRAINT ficheiro_entrega_FK FOREIGN KEY
    (
        entrega_id
    )
    REFERENCES entrega
    (
        id
    )
;
;

ALTER TABLE ficheiro_recurso
    ADD CONSTRAINT ficheiro_recurso_recurso_FK FOREIGN KEY
    (
        recurso_id
    )
    REFERENCES recurso
    (
        id
    )
;
;
```

Figura 203 - ALTER Tabelas "ficheiro_entrega" e "ficheiro_recurso".

```
ALTER TABLE matricula
    ADD CONSTRAINT inscricao_curso_curso_FK FOREIGN KEY
    (
        curso_id
    )
    REFERENCES curso
    (
        id
    )
;
;

ALTER TABLE matricula
    ADD CONSTRAINT inscricao_curso_estudante_FK FOREIGN KEY
    (
        estudante_id
    )
    REFERENCES estudante
    (
        id
    )
;
;
```

Figura 204 - ALTER da Tabela "matricula".



```
ALTER TABLE inscricao
    ADD CONSTRAINT inscricao_matricula_FK FOREIGN KEY
    (
        matricula_id
    )
    REFERENCES matricula
    (
        id
    )
;

ALTER TABLE inscricao
    ADD CONSTRAINT inscricao_turma_FK FOREIGN KEY
    (
        turma_id
    )
    REFERENCES turma
    (
        id
    )
;
```

Figura 205 - ALTER da Tabela "inscricao".

```
ALTER TABLE nota
    ADD CONSTRAINT nota_avaliacao_FK FOREIGN KEY
    (
        avaliacao_id
    )
    REFERENCES avaliacao
    (
        id
    )
;

ALTER TABLE nota
    ADD CONSTRAINT nota_inscricao_FK FOREIGN KEY
    (
        inscricao_id
    )
    REFERENCES inscricao
    (
        id
    )
;
```

Figura 206 - ALTER da Tabela "nota".

```
ALTER TABLE parcela_propina
    ADD CONSTRAINT parcela_propina_matricula_FK FOREIGN KEY
    (
        matricula_id
    )
    REFERENCES matricula
    (
        id
    )
;
```

Figura 207 - ALTER da Tabela "parcela_propina".

```
ALTER TABLE presenca
    ADD CONSTRAINT presenca_aula_FK FOREIGN KEY
    (
        aula_id
    )
    REFERENCES aula
    (
        id
    )
;
ALTER TABLE presenca
    ADD CONSTRAINT presenca_inscricao_FK FOREIGN KEY
    (
        inscricao_id
    )
    REFERENCES inscricao
    (
        id
    )
;
```

Figura 208 - ALTER da Tabela "presenca".

```
ALTER TABLE recurso
    ADD CONSTRAINT recurso_docente_FK FOREIGN KEY
    (
        docente_id
    )
    REFERENCES docente
    (
        id
    )
;

ALTER TABLE recurso
    ADD CONSTRAINT recurso_turma_FK FOREIGN KEY
    (
        turma_id
    )
    REFERENCES turma
    (
        id
    )
;
```

Figura 209 - ALTER da Tabela "recurso".

```
ALTER TABLE turma
    ADD CONSTRAINT turma_docente_FK FOREIGN KEY
    (
        docente_id
    )
    REFERENCES docente
    (
        id
    )
;

ALTER TABLE turma
    ADD CONSTRAINT turma_unidade_curricular_FK FOREIGN KEY
    (
        unidade_curricular_id
    )
    REFERENCES unidade_curricular
    (
        id
    )
;
```

Figura 210 - ALTER da Tabela "turma".

```
ALTER TABLE uc_curso
    ADD CONSTRAINT uc_curso_curso_FK FOREIGN KEY
    (
        curso_id
    )
    REFERENCES curso
    (
        id
    )
;
;

ALTER TABLE uc_curso
    ADD CONSTRAINT uc_curso_uc_FK FOREIGN KEY
    (
        unidade_curricular_id
    )
    REFERENCES unidade_curricular
    (
        id
    )
;
;
```

Figura 211 - ALTER da Tabela "uc_curso".

```
ALTER TABLE uc_docente
    ADD CONSTRAINT uc_docente_docente_FK FOREIGN KEY
    (
        docente_id
    )
    REFERENCES docente
    (
        id
    )
;
;

ALTER TABLE uc_docente
    ADD CONSTRAINT uc_docente_uc_FK FOREIGN KEY
    (
        unidade_curricular_id
    )
    REFERENCES unidade_curricular
    (
        id
    )
;
;
```

Figura 212 - ALTER da Tabela "uc_docente".

Inserção de Dados

Uma vez determinado o modelo físico do sistema, terem sido aplicadas as devidas alterações no modelo físico e no *script DDL*, chega à hora da verdade que no caso é a inserção de dados.

Devido à mudança de paradigma (tópico já desenvolvido, na página 88), o grupo só precisou de inserir dados uma única vez, isto porque, seria o primeiro ano de atividade da instituição de ensino superior.

Inicialmente houveram algumas dificuldades na inserção devido a tipos de dados de alguns atributos, nomeadamente atributos *timestamp* e datas. Para efeitos de primeira tentativa de inserção, foi usada a ferramenta *Mockaroo*, na sequência do uso dessa ferramenta nasceram as primeiras inconsistências e numa segunda instância o grupo chegou a recorrer a *ChatGPT* ou até mesmo ao uso da biblioteca *Faker* para gerar dados fictícios usando a linguagem *Python*.

Numa terceira e final instância, finalmente após ajustes nos tipo de dados de certos atributos, foi possível gerar dados com recurso a um agente de inteligência artificial chamado *Manus* e aí sim, foi possível uma primeira inserção de dados sem erros.

Quanto a números de dados inseridos por cada tabela, o projeto conta com:

- 1. Tabela “parcela_propina”:** Inserções 4800;
- 2. Tabela “presenca”:** Inserções 4500;
- 3. Tabela “inscricao”:** Inserções 4000;
- 4. Tabela “nota”:** Inserções 3200;
- 5. Tabela “estudante”:** Inserções 800;
- 6. Tabela “matricula”:** Inserções 800;
- 7. Tabela “ficheiro_entrega”:** Inserções 600;
- 8. Tabela “estudante_entrega”:** Inserções 600;
- 9. Tabela “aula”:** Inserções 600;
- 10. Tabela “entrega”:** Inserções 200;
- 11. Tabela “ficheiro_recurso”:** Inserções 150;
- 12. Tabela “recurso”:** Inserções 150;
- 13. Tabela “avaliacao”:** Inserções 120;
- 14. Tabela “turma”:** Inserções 40;
- 15. Tabela “uc_docente”:** Inserções 35;
- 16. Tabela “uc_curso”:** Inserções 30;
- 17. Tabela “docente”:** Inserções 25;

- 18. Tabela “unidade_curricular”:** Inserções 18;
- 19. Tabela “sala”:** Inserções 15;
- 20. Tabela “tipo_avaliacao”:** Inserções 7;
- 21. Tabela “curso”:** Inserções 3;
- 22. Tabela “tipo_curso”:** Inserções 3;
- 23. Tabela “tipo_aula”:** Inserções 2;

De notar ainda, que as inserções estão ordenadas por ordem sequencial e também que a quantidade de dados inseridos parece proporcional ao sistema e à mudança de paradigma.

Apesar de serem 24 tabelas, a tabela “*log*”, não conta com qualquer inserção, uma vez que a mesma recebe atualizações de todas as outras tabelas do sistema.

Operações PL/SQL

Criadas as bases do sistema e inseridos os dados, a próxima etapa desenvolvida diz respeito à criação de múltiplas operações na base de dados. Operações como funções, procedimentos, vistas, pacotes, sequências, cursores, entre outras, foram realizadas.

O sistema é caracterizado por uma arquitetura robusta que inclui: auditoria, isto é, detalha todas as operações DML, validação rigorosa de dados de entrada e regras de negócio (neste caso, uma universidade/politécnico), faz a automação de processos como a geração de números de aluno, presenças e planos de propinas e ainda faz o cálculo automático de médias e controlo de limites académicos (ECTS), entre muitas outras operações.

O presente tópico visa desmembrar o que foi feito em cada operação PL/SQL.

Ordem de Operações Necessária para o Sistema Funcionar

A ordem de execução dos *scripts* é crítica devido às dependências entre os objetos (por exemplo, *triggers* dependem de *packages*, que dependem de *sequences*). A ordem lógica de compilação é a seguinte:

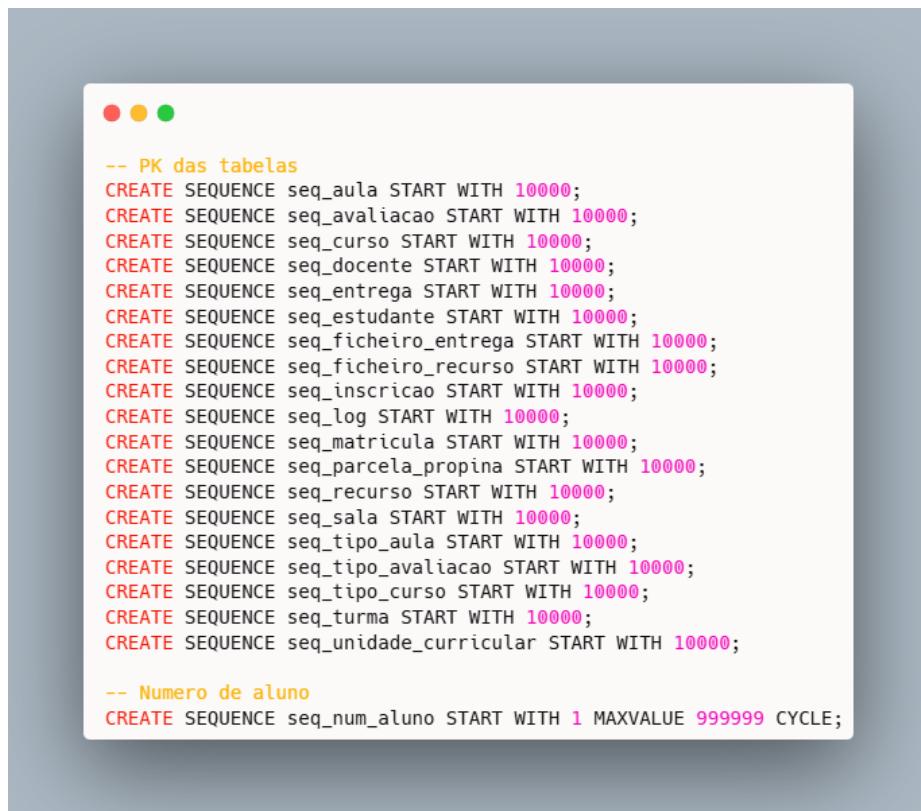
Nível	Tipo de Objeto	Descrição	Dependências Chave
1	<i>CREATE SEQUENCE</i>	Criação das sequências de auto-incremento para as chaves primárias e para o número de aluno.	Nenhuma
2	<i>PACKAGE</i> (Constantes)	Criação do “PKG_CONSTANTES” (<i>Specification e Body</i>), que define todos os parâmetros de configuração do sistema.	Nenhuma
3	<i>PACKAGE</i> (Core Utilities)	Criação dos pacotes “PKG_LOG”, “PKG_VALIDACAO” e “PKG_GESTAO_DADOS”.	PKG_CONSTANTES
4	<i>PACKAGE</i> (Buffer e Tesouraria)	Criação dos pacotes de utilidade “PKG_BUFFER_MATRICULA”, “PKG_BUFFER_NOTA”, “PKG_BUFFER_INSCRICAO” e “PKG_TESOURARIA”.	PKG_CONSTANTES e PKG_LOG
5	<i>TRIGGER</i> (Auto-Incremento)	Criação dos TRG_AUDIT_* que preenchem automaticamente as chaves primárias e o código estudante.	Sequences
6	<i>TRIGGER</i> (Auditoria)	Criação dos TRG_AUDIT_* que registam todas as operações DML na tabela “log”.	PKG_LOG
7	<i>TRIGGER</i> (Validação e Lógica)	Criação de todos os TRIGGERS de validação (TRG_VAL_*) e lógica de negócio (TRG_AUTO_*, TRG_MEDIA_*, TRG_INSCRICAO_*).	Packages de Nível 3 e 4
8	<i>FUNCTION</i>	Criação de funções auxiliares, como “FUN_GET_ASSIDUIDADE”.	Tabelas de Presença
9	<i>VIEW</i>	Criação das vistas de relatório (VW_*) para consultas complexas.	Funções, Packages e Tabelas

Nota: Na tabela da página 125, quando aparece nomes de operações como “TRG_VAL_*”, o asterisco indica que existem mais *TRIGGERS* de validação aplicados a diversas tabelas do sistema.

Operações PL/SQL do Sistema

Infraestrutura de Sequenciação e Identificação

O script inicia-se com a criação das estruturas de sequência para a base de dados. São definidas sequências individuais para cada tabela principal (“seq_aula”, “seq_avaliacao”, entre outras...) com início no valor 10.000, isto acaba por reservar os identificadores inferiores para dados de sistema. Destaca-se ainda que nestas linhas a criação da “seq_num_aluno”, configurada especificamente com a propriedade *CYCLE* e um valor máximo, destinada a gerir a numeração dos estudantes de forma rotativa.



```
-- PK das tabelas
CREATE SEQUENCE seq_aula START WITH 10000;
CREATE SEQUENCE seq_avaliacao START WITH 10000;
CREATE SEQUENCE seq_curso START WITH 10000;
CREATE SEQUENCE seq_docente START WITH 10000;
CREATE SEQUENCE seq_entrega START WITH 10000;
CREATE SEQUENCE seq_estudante START WITH 10000;
CREATE SEQUENCE seq_ficheiro_entrega START WITH 10000;
CREATE SEQUENCE seq_ficheiro_recurso START WITH 10000;
CREATE SEQUENCE seq_inscricao START WITH 10000;
CREATE SEQUENCE seq_log START WITH 10000;
CREATE SEQUENCE seq_matricula START WITH 10000;
CREATE SEQUENCE seq_parcela_propina START WITH 10000;
CREATE SEQUENCE seq_recurso START WITH 10000;
CREATE SEQUENCE seq_sala START WITH 10000;
CREATE SEQUENCE seq_tipo_aula START WITH 10000;
CREATE SEQUENCE seq_tipo_avaliacao START WITH 10000;
CREATE SEQUENCE seq_tipo_curso START WITH 10000;
CREATE SEQUENCE seq_turma START WITH 10000;
CREATE SEQUENCE seq_unidade_curricular START WITH 10000;

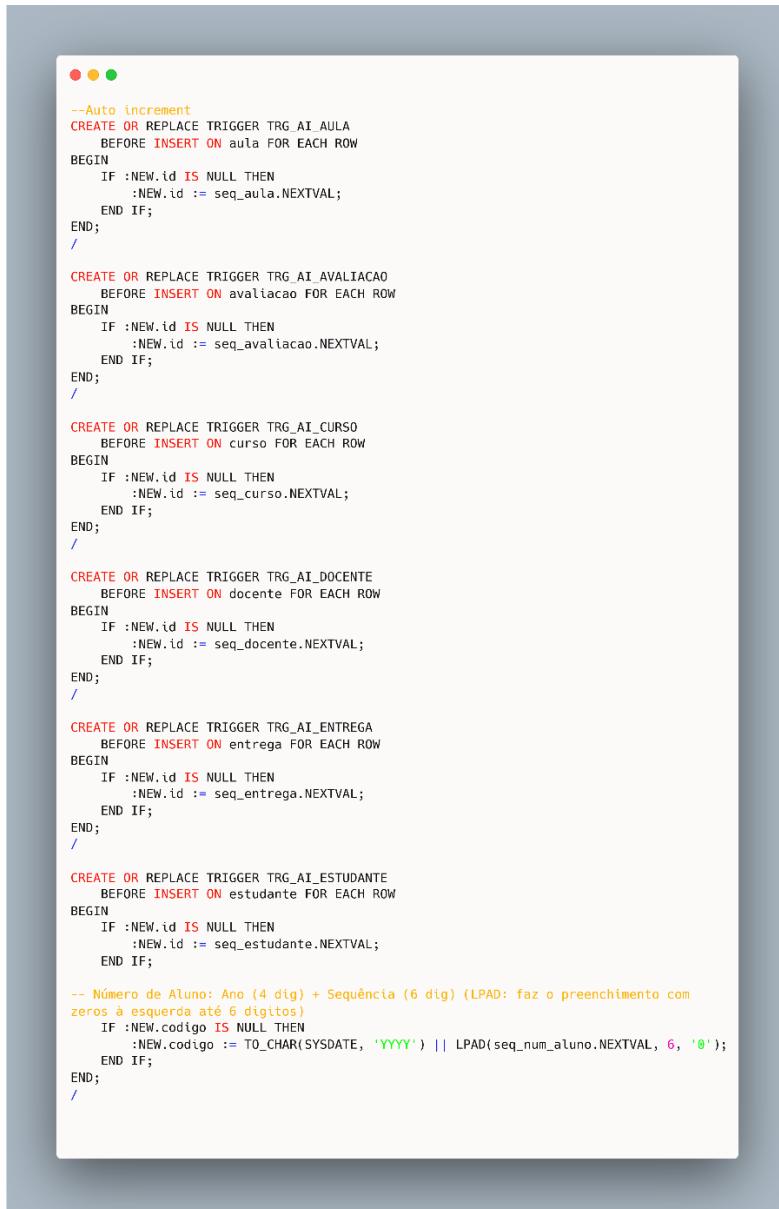
-- Número de aluno
CREATE SEQUENCE seq_num_aluno START WITH 1 MAXVALUE 999999 CYCLE;
```

Figura 213 - Sequenciação.



TRIGGERS de Auto-Incremento e Formatação de Código

Neste intervalo extenso de linhas de código, são definidos *TRIGGERS* do tipo *BEFORE INSERT* para cada tabela do sistema. A operação principal destes blocos é verificar se o campo “ID” do novo registo é nulo e, caso seja verdade, preenchê-lo com o próximo valor da sequência correspondente. Merece também destaque a lógica contida nas linhas referentes à tabela de estudantes, onde, além do “ID”, é concatenado o ano corrente com a sequência para formar o código de aluno mais padronizado (por exemplo, 2024000001).



```
--Auto increment
CREATE OR REPLACE TRIGGER TRG_AI_AULA
  BEFORE INSERT ON aula FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_aula.NEXTVAL;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_AI_AVALIACAO
  BEFORE INSERT ON avaliacao FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_avaliacao.NEXTVAL;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_AI_CURSO
  BEFORE INSERT ON curso FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_curso.NEXTVAL;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_AI_DOCENTE
  BEFORE INSERT ON docente FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_docente.NEXTVAL;
  END IF;
END;
/

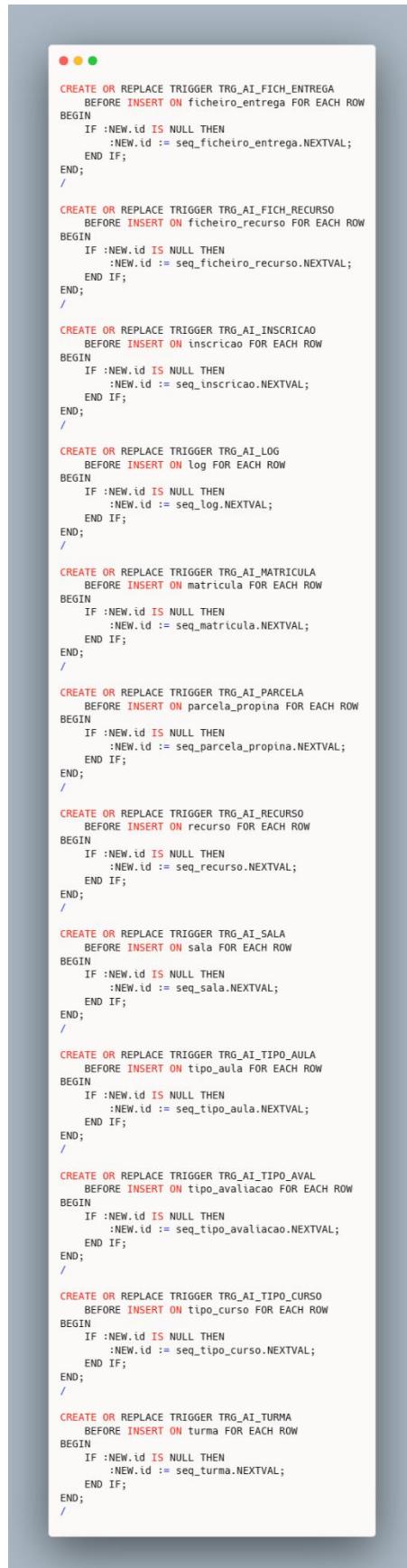
CREATE OR REPLACE TRIGGER TRG_AI_ENTREGA
  BEFORE INSERT ON entrega FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_entrega.NEXTVAL;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_AI_ESTUDANTE
  BEFORE INSERT ON estudiante FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_estudante.NEXTVAL;
  END IF;

  -- Número de Aluno: Ano (4 dig) + Sequência (6 dig) (LPAD: faz o preenchimento com zeros à esquerda até 6 dígitos)
  IF :NEW.codigo IS NULL THEN
    :NEW.codigo := TO_CHAR(SYSDATE, 'YYYY') || LPAD(seq_num_aluno.NEXTVAL, 6, '0');
  END IF;
END;
/
```

Figura 214 - TRIGGERS Auto_Incremento, Parte 1.

Instituto Politécnico de Coimbra
Instituto Superior de Engenharia de Coimbra



```
CREATE OR REPLACE TRIGGER TRG_AI_FICH_ENTREGA
  BEFORE INSERT ON ficheiro_entrega FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_ficheiro_entrega.NEXTVAL;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_AI_FICH_RECURSO
  BEFORE INSERT ON ficheiro_recurso FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_ficheiro_recurso.NEXTVAL;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_AI_INSCRICAO
  BEFORE INSERT ON inscricao FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_inscricao.NEXTVAL;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_AI_LOG
  BEFORE INSERT ON log FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_log.NEXTVAL;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_AI_MATRICULA
  BEFORE INSERT ON matricula FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_matricula.NEXTVAL;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_AI_PARCELA
  BEFORE INSERT ON parcela_propina FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_parcela_propina.NEXTVAL;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_AI_RECURSO
  BEFORE INSERT ON recurso FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_recurso.NEXTVAL;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_AI_SALA
  BEFORE INSERT ON sala FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_sala.NEXTVAL;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_AI_TIPO_AULA
  BEFORE INSERT ON tipo_aula FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_tipo_aula.NEXTVAL;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_AI_TIPO_AVAL
  BEFORE INSERT ON tipo_avaliacao FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_tipo_avaliacao.NEXTVAL;
  END IF;
END;
/

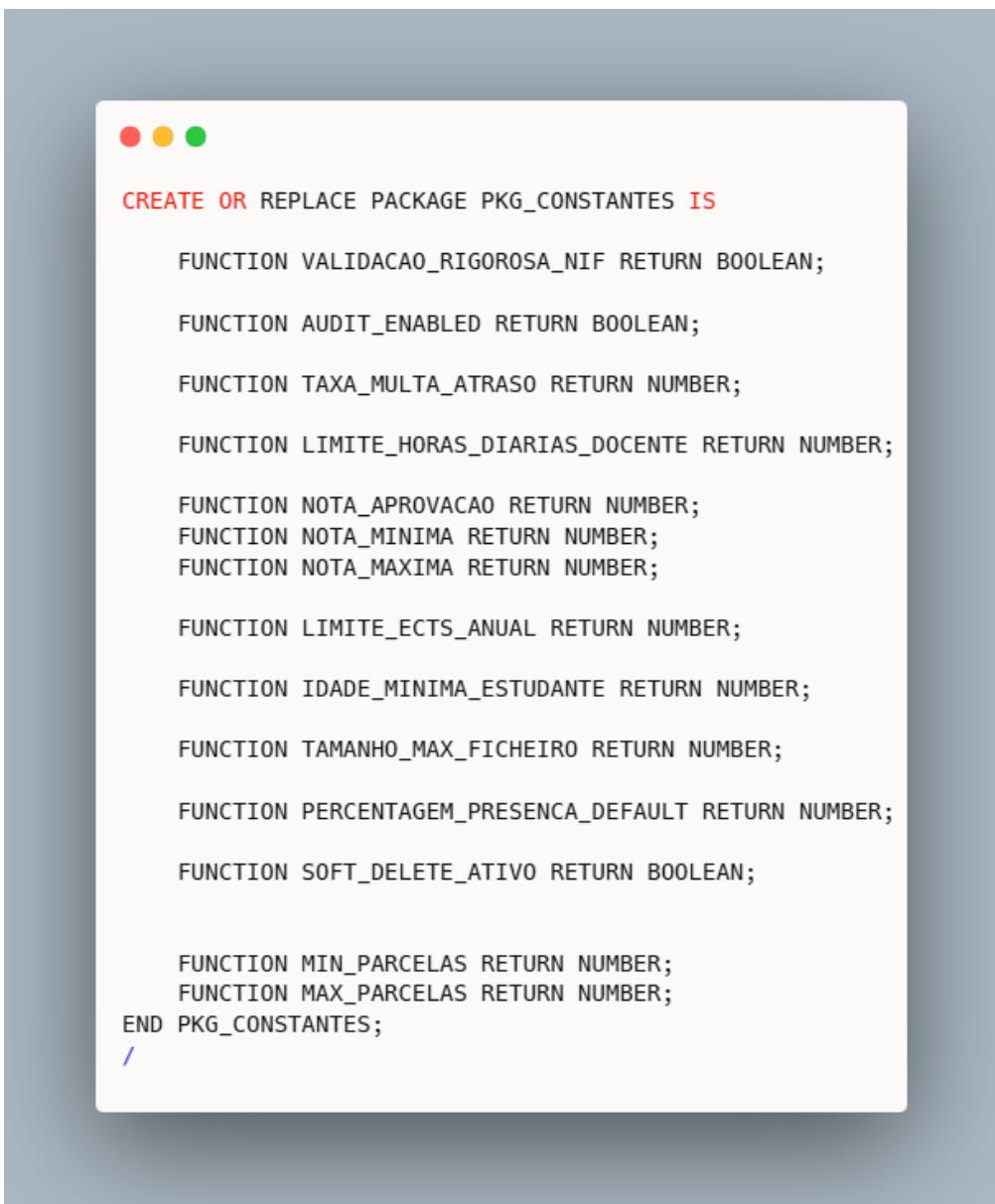
CREATE OR REPLACE TRIGGER TRG_AI_TIPO_CURSO
  BEFORE INSERT ON tipo_curso FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_tipo_curso.NEXTVAL;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER TRG_AI_TURMA
  BEFORE INSERT ON turma FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    :NEW.id := seq_turma.NEXTVAL;
  END IF;
END;
/
```

Figura 215 - TRIGGERS Auto-Incremento, Parte 2.

PACKAGE de Constantes e Configuração Global

Este bloco contém a especificação (*header*) e o corpo do pacote (*body*) “PKG_CONSTANTES”. A operação aqui realizada é a centralização de todas as regras de negócio que possam ser parametrizáveis. São definidas funções que acabam por retornar valores fixos para taxas de multas, limites de horas em que os docentes podem dar aulas, notas de aprovação, limites de ECTS anuais e também tamanhos fixos de ficheiros. Esta estrutura permite que a alteração de uma regra transversal ao sistema seja feita num único ponto de código PL/SQL.



```
CREATE OR REPLACE PACKAGE PKG_CONSTANTES IS

    FUNCTION VALIDACAO_RIGOROSA_NIF RETURN BOOLEAN;

    FUNCTION AUDIT_ENABLED RETURN BOOLEAN;

    FUNCTION TAXA_MULTA_ATRASO RETURN NUMBER;

    FUNCTION LIMITE_HORAS_DIARIAS_DOCENTE RETURN NUMBER;

    FUNCTION NOTA_APROVACAO RETURN NUMBER;
    FUNCTION NOTA_MINIMA RETURN NUMBER;
    FUNCTION NOTA_MAXIMA RETURN NUMBER;

    FUNCTION LIMITE_ECTS_ANUAL RETURN NUMBER;

    FUNCTION IDADE_MINIMA_ESTUDANTE RETURN NUMBER;

    FUNCTION TAMANHO_MAX_FICHEIRO RETURN NUMBER;

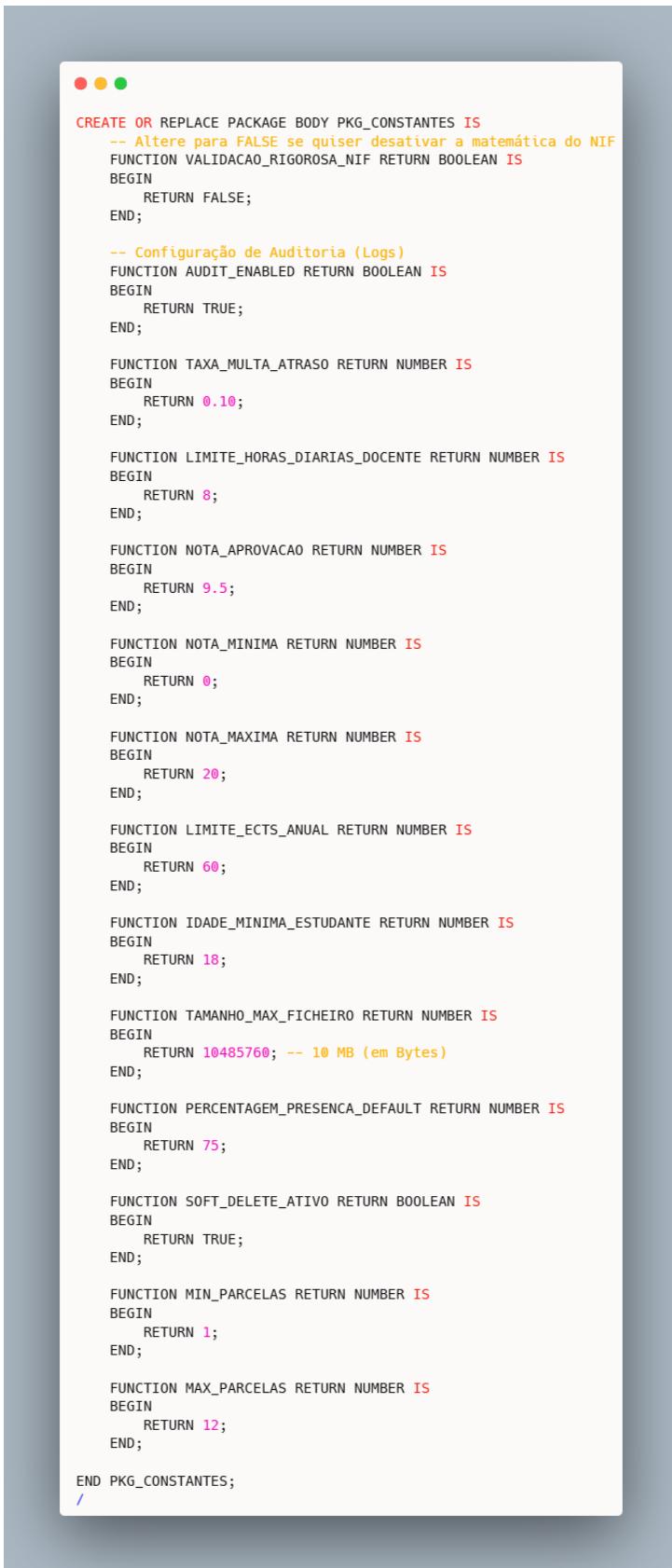
    FUNCTION PERCENTAGEM_PRESENCA_DEFAULT RETURN NUMBER;

    FUNCTION SOFT_DELETE_ATIVO RETURN BOOLEAN;

    FUNCTION MIN_PARCELAS RETURN NUMBER;
    FUNCTION MAX_PARCELAS RETURN NUMBER;
END PKG_CONSTANTES;
/
```

Figura 216 - PKG_CONSTANTES, Parte 1.





```
CREATE OR REPLACE PACKAGE BODY PKG_CONSTANTES IS
    -- Altere para FALSE se quiser desativar a matemática do NIF
    FUNCTION VALIDACAO_RIGOROSA_NIF RETURN BOOLEAN IS
        BEGIN
            RETURN FALSE;
        END;

    -- Configuração de Auditoria (Logs)
    FUNCTION AUDIT_ENABLED RETURN BOOLEAN IS
        BEGIN
            RETURN TRUE;
        END;

    FUNCTION TAXA_MULTA_ATRASO RETURN NUMBER IS
        BEGIN
            RETURN 0.10;
        END;

    FUNCTION LIMITE_HORAS_DIARIAS_DOCENTE RETURN NUMBER IS
        BEGIN
            RETURN 8;
        END;

    FUNCTION NOTA_APROVACAO RETURN NUMBER IS
        BEGIN
            RETURN 9.5;
        END;

    FUNCTION NOTA_MINIMA RETURN NUMBER IS
        BEGIN
            RETURN 0;
        END;

    FUNCTION NOTA_MAXIMA RETURN NUMBER IS
        BEGIN
            RETURN 20;
        END;

    FUNCTION LIMITE_ECTS_ANUAL RETURN NUMBER IS
        BEGIN
            RETURN 60;
        END;

    FUNCTION IDADE_MINIMA_ESTUDANTE RETURN NUMBER IS
        BEGIN
            RETURN 18;
        END;

    FUNCTION TAMANHO_MAX_FICHEIRO RETURN NUMBER IS
        BEGIN
            RETURN 10485760; -- 10 MB (em Bytes)
        END;

    FUNCTION PERCENTAGEM_PRESENCA_DEFAULT RETURN NUMBER IS
        BEGIN
            RETURN 75;
        END;

    FUNCTION SOFT_DELETE_ATIVO RETURN BOOLEAN IS
        BEGIN
            RETURN TRUE;
        END;

    FUNCTION MIN_PARCELAS RETURN NUMBER IS
        BEGIN
            RETURN 1;
        END;

    FUNCTION MAX_PARCELAS RETURN NUMBER IS
        BEGIN
            RETURN 12;
        END;
END PKG_CONSTANTES;
/
```

Figura 217 - PKG_CONSTANTES, Parte 2.

PACKAGE de Auditoria e Logs

Aqui é estabelecido o PACKAGE “PKG_LOG”, que é responsável pelo registo de atividades do sistema. A lógica implementada utiliza transações autónomas (*PRAGMA AUTONOMOUS_TRANSACTION*) para garantir que os registos de erros ou auditoria se mantêm persistentes na tabela “log” mesmo que a transação de negócios principal falhe ou sofra algum *rollback*. O PACKAGE expõe também que PROCEDURES para registar mensagens genéricas, erros críticos e operações de manipulação de dados (DML).



```
--LOGS (Auditoria)
CREATE OR REPLACE PACKAGE PKG_LOG AS
    PROCEDURE REGISTAR(p_acao VARCHAR2, p_msg VARCHAR2, p_tabela VARCHAR2 DEFAULT NULL);
    PROCEDURE REGISTAR_DML(p_tabela VARCHAR2, p_acao VARCHAR2, p_id_registro VARCHAR2);
    PROCEDURE ERRO(p_msg VARCHAR2, p_tabela VARCHAR2 DEFAULT NULL);
    PROCEDURE ALERTA(p_msg VARCHAR2, p_tabela VARCHAR2 DEFAULT NULL);
END PKG_LOG;
/

CREATE OR REPLACE PACKAGE BODY PKG_LOG AS
    PROCEDURE REGISTAR(p_acao VARCHAR2, p_msg VARCHAR2, p_tabela VARCHAR2 DEFAULT NULL)
    IS
        -- O PRAGMA AUTONOMOUS_TRANSACTION Garante que o registo no LOG seja persistido
        -- mesmo que a transação principal (que chamou este log) sofra um ROLLBACK
        -- devido a um erro.
        -- Essencial para auditoria e depuração.
        PRAGMA AUTONOMOUS_TRANSACTION;
    BEGIN
        -- Regista-se a auditoria estiver ativa OU se for um erro crítico
        IF PKG_CONSTANTES.AUDIT_ENABLED OR p_acao = 'ERRO' THEN
            INSERT INTO log (acao, tabela, data, created_at)
            VALUES (p_acao, p_tabela, p_msg, CURRENT_TIMESTAMP);
            COMMIT;
        END IF;
        EXCEPTION
            WHEN OTHERS THEN
                NULL;
        END;

        PROCEDURE REGISTAR_DML(p_tabela VARCHAR2, p_acao VARCHAR2, p_id_registro VARCHAR2)
        IS
        BEGIN
            REGISTAR(p_acao, 'Registro ID: ' || p_id_registro, p_tabela);
        END;

        PROCEDURE ERRO(p_msg VARCHAR2, p_tabela VARCHAR2 DEFAULT NULL) IS
        BEGIN
            REGISTAR('ERRO', p_msg, p_tabela);
        END;

        PROCEDURE ALERTA(p_msg VARCHAR2, p_tabela VARCHAR2 DEFAULT NULL) IS
        BEGIN
            REGISTAR('ALERTA', p_msg, p_tabela);
        END;
    END PKG_LOG;
/
```

Figura 218 - PACKAGE LOG.



PACKAGE de Gestão e Remoção de Dados

O *PACKAGE*, “PKG_GESTAO_DADOS” define a lógica de remoção de regtos. O código implementado consulta a configuração global para decidir entre dois modos de operação: a remoção física (um *DELETE*) ou uma desativação lógica (um *SOFT DELETE*), em que o registo é apenas marcado com “status” igual a 0. Este bloco inclui *PROCEDURES* distintos para tratar tabelas com chaves primárias simples e tabelas de relacionamentos com chaves compostas.

```

-- Dois tipos de Delete (Soft e Hard)
-- pode ser ativado/desativado com constante no PKG_CONSTANTES

CREATE OR REPLACE PACKAGE PKG_GESTAO_DADOS IS
    -- Versão para tabelas com Chave Primária simples (coluna ID)
    PROCEDURE PRC_REMOVER(p_nome_tabela IN VARCHAR2, p_id IN NUMBER);

    -- Versão para tabelas de ligação ou chaves compostas
    PROCEDURE PRC_REMOVER_RELACAO(
        p_nome_tabela IN VARCHAR2,
        p_id_1         IN NUMBER,
        p_col_1        IN VARCHAR2,
        p_id_2         IN NUMBER,
        p_col_2        IN VARCHAR2
    );
END PKG_GESTAO_DADOS;
/

CREATE OR REPLACE PACKAGE BODY PKG_GESTAO_DADOS IS
    -- REMOÇÃO SIMPLES (PK Simples)
    PROCEDURE PRC_REMOVER(p_nome_tabela IN VARCHAR2, p_id IN NUMBER) IS
        v_sql VARCHAR2(500);
        v_tab VARCHAR2(30) := p_nome_tabela;
    BEGIN
        IF PKG_CONSTANTES.SOFT_DELETE_ATIVO THEN
            v_sql := 'UPDATE ' || v_tab || ' SET status = ''0'', updated_at = CURRENT_TIMESTAMP WHERE id = :1';
            EXECUTE IMMEDIATE v_sql USING p_id;
        ELSE
            v_sql := 'DELETE FROM ' || v_tab || ' WHERE id = :1';
            EXECUTE IMMEDIATE v_sql USING p_id;
        END IF;
        EXCEPTION WHEN OTHERS THEN
           PKG_LOG.ERRO('Falha ao remover registo ID ' || p_id || ' em ' || v_tab || ':');
    || SQLERRM, v_tab);
    END PRC_REMOVER;

    -- PK composta / Tabelas de Ligação
    PROCEDURE PRC_REMOVER_RELACAO(
        p_nome_tabela IN VARCHAR2,
        p_id_1         IN NUMBER,
        p_col_1        IN VARCHAR2,
        p_id_2         IN NUMBER,
        p_col_2        IN VARCHAR2
    ) IS
        v_sql  VARCHAR2(1000);
        v_tab  VARCHAR2(30) := p_nome_tabela;
        v_where VARCHAR2(500);
    BEGIN
        v_where := ' WHERE ' || p_col_1 || ' = :1 AND ' || p_col_2 || ' = :2';

        IF PKG_CONSTANTES.SOFT_DELETE_ATIVO THEN
            v_sql := 'UPDATE ' || v_tab || ' SET status = ''0'', updated_at = CURRENT_TIMESTAMP' || v_where;
            EXECUTE IMMEDIATE v_sql USING p_id_1, p_id_2;
        ELSE
            v_sql := 'DELETE FROM ' || v_tab || v_where;
            EXECUTE IMMEDIATE v_sql USING p_id_1, p_id_2;
        END IF;
        EXCEPTION WHEN OTHERS THEN
           PKG_LOG.ERRO('Falha ao remover registo em ' || v_tab || ': ' || SQLERRM,
v_tab);
    END PRC_REMOVER_RELACAO;
END PKG_GESTAO_DADOS;
/

```

Figura 219 - PACKAGE “PKG_GESTAO_DADOS”.

PACKAGE de Validações Reutilizáveis

Este tópico serve para falar do “PKG_VALIDACAO”, que é uma “biblioteca” de funções para verificação de integridade dos dados. As operações incluem algoritmos matemáticos para validar o dígito de controlo do NIF (no caso do padrão nacional, máximo 9 dígitos), expressões regulares para validar formatos de cartão de cidadão, e-mail e IBAN (25 caracteres), bem como verificações de consistência para números de telemóvel e tamanhos de ficheiros binários.

```
-- Validações Gerais
CREATE OR REPLACE PACKAGE PKG_VALIDACAO IS
    FUNCTION FUN_VALIDAR_NIF(p_nif IN VARCHAR2) RETURN BOOLEAN;
    FUNCTION FUN_VALIDAR_CC(p_cc IN VARCHAR2) RETURN BOOLEAN;
    FUNCTION FUN_VALIDAR_EMAIL(p_email IN VARCHAR2) RETURN BOOLEAN;
    FUNCTION FUN_VALIDAR_IBAN(p_iban IN VARCHAR2) RETURN BOOLEAN;
    FUNCTION FUN_VALIDAR_TELEMOVEL(p_telemovel IN VARCHAR2) RETURN BOOLEAN;
    FUNCTION FUN_VALIDAR_TAMANHO_FICHEIRO(p_tamanho IN NUMBER) RETURN BOOLEAN;

    FUNCTION FUN_VALIDAR_STATUS(p_status IN VARCHAR2, p_tabela IN VARCHAR2) RETURN
VARCHAR2;
END PKG_VALIDACAO;
/

CREATE OR REPLACE PACKAGE BODY PKG_VALIDACAO IS

    -- Validação de Status (0 ou 1)
    FUNCTION FUN_VALIDAR_STATUS(p_status IN VARCHAR2, p_tabela IN VARCHAR2) RETURN
VARCHAR2 IS
    BEGIN
        IF p_status NOT IN ('0', '1') OR p_status IS NULL THEN
            PKG_LOG.ERRO('Status invalido (' || NVL(p_status, 'NULL') || ') na tabela '
|| p_tabela || '. Forcado a 0.', p_tabela);
            RETURN '0';
        END IF;
        RETURN p_status;
    END FUN_VALIDAR_STATUS;

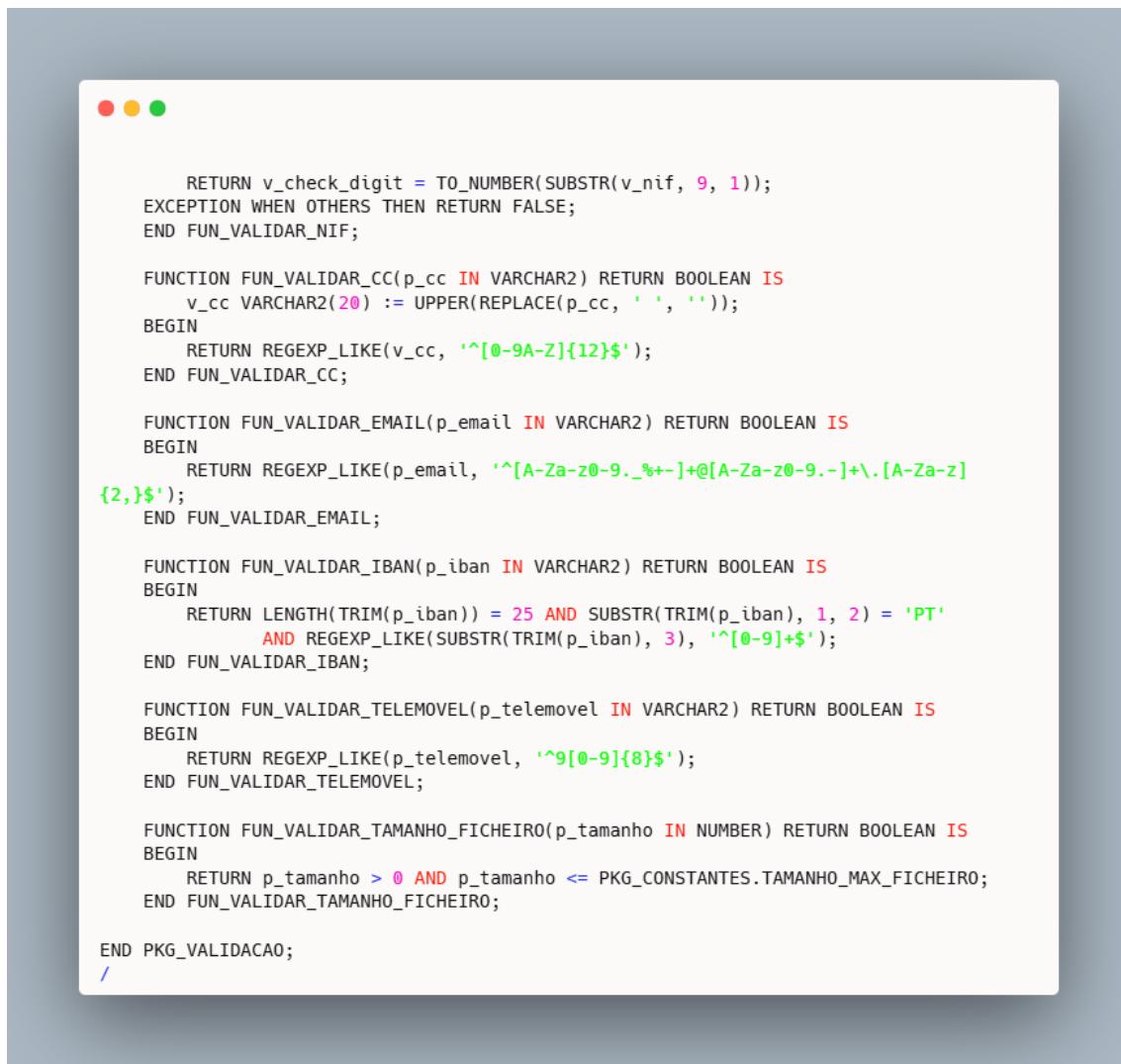
    FUNCTION FUN_VALIDAR_NIF(p_nif IN VARCHAR2) RETURN BOOLEAN IS
        v_nif VARCHAR2(9) := p_nif;
        v_soma NUMBER := 0;
        v_resto NUMBER;
        v_check_digit NUMBER;
        i NUMBER := 1;
    BEGIN
        -- Formato Básico (9 dígitos numéricos)
        IF LENGTH(v_nif) != 9 OR NOT REGEXP_LIKE(v_nif, '^[0-9]+$') THEN
            RETURN FALSE;
        END IF;

        -- Se não for rigorosa, o formato basta
        IF NOT PKG_CONSTANTES.VALIDACAO_RIGOROSA_NIF THEN
            RETURN TRUE;
        END IF;

        -- Algoritmo Modulo 11
        LOOP
            EXIT WHEN i > 8;
            v_soma := v_soma + TO_NUMBER(SUBSTR(v_nif, i, 1)) * (10 - i);
            i := i + 1;
        END LOOP;

        v_resto := MOD(v_soma, 11);
        v_check_digit := CASE WHEN v_resto IN (0, 1) THEN 0 ELSE 11 - v_resto END;
    END FUN_VALIDAR_NIF;
```

Figura 220 - PACKAGE "PKG_VALIDACAO", Parte 1.



The screenshot shows a code editor window with a dark theme. The code is a PL/SQL package named PKG_VALIDACAO. It contains several functions for validating different types of data:

```
    RETURN v_check_digit = TO_NUMBER(SUBSTR(v_nif, 9, 1));
EXCEPTION WHEN OTHERS THEN RETURN FALSE;
END FUN_VALIDAR_NIF;

FUNCTION FUN_VALIDAR_CC(p_cc IN VARCHAR2) RETURN BOOLEAN IS
    v_cc VARCHAR2(20) := UPPER(REPLACE(p_cc, ' ', '')); 
BEGIN
    RETURN REGEXP_LIKE(v_cc, '^[0-9A-Z]{12}$');
END FUN_VALIDAR_CC;

FUNCTION FUN_VALIDAR_EMAIL(p_email IN VARCHAR2) RETURN BOOLEAN IS
BEGIN
    RETURN REGEXP_LIKE(p_email, '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}+$');
END FUN_VALIDAR_EMAIL;

FUNCTION FUN_VALIDAR_IBAN(p_ibanc IN VARCHAR2) RETURN BOOLEAN IS
BEGIN
    RETURN LENGTH(TRIM(p_ibanc)) = 25 AND SUBSTR(TRIM(p_ibanc), 1, 2) = 'PT'
        AND REGEXP_LIKE(SUBSTR(TRIM(p_ibanc), 3), '^[0-9]{12}$');
END FUN_VALIDAR_IBAN;

FUNCTION FUN_VALIDAR_TELEMOVEL(p_telemovel IN VARCHAR2) RETURN BOOLEAN IS
BEGIN
    RETURN REGEXP_LIKE(p_telemovel, '^9[0-9]{8}$');
END FUN_VALIDAR_TELEMOVEL;

FUNCTION FUN_VALIDAR_TAMANHO_FICHEIRO(p_tamanho IN NUMBER) RETURN BOOLEAN IS
BEGIN
    RETURN p_tamanho > 0 AND p_tamanho <= PKG_CONSTANTES.TAMANHO_MAX_FICHEIRO;
END FUN_VALIDAR_TAMANHO_FICHEIRO;

END PKG_VALIDACAO;
/
```

Figura 221 - PACKAGE "PKG_VALIDACAO", Parte 2.

Buffers para Resolução de Tabelas Mutantes

No presente projeto, existem espalhados estrategicamente pelo código (antes dos TRIGGERS que os utilizam), encontram-se PACKAGES de buffer (PKG_BUFFER_MATRICULA, PKG_BUFFER_NOTA, PKG_BUFFER_INSCRICAO). Estes blocos de código declaram vetores para armazenar identificadores temporariamente durante a execução de TRIGGERS linha-a-linha. Esta operação é fundamental para contornar a restrição da “*Mutating Table*” do Oracle, para assim permitir que os dados sejam processados em lotes num momento posterior da transação.

```
● ● ●

-- PKG para gerir buffer de matrículas
CREATE OR REPLACE PACKAGE PKG_BUFFER_MATRICULA IS
    TYPE t_lista_ids IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
    v_ids_matricula t_lista_ids;

    PROCEDURE LIMPAR;
    PROCEDURE ADICIONAR(p_id NUMBER);
END PKG_BUFFER_MATRICULA;

CREATE OR REPLACE PACKAGE BODY PKG_BUFFER_MATRICULA IS
    PROCEDURE LIMPAR IS
    BEGIN
        v_ids_matricula.DELETE;
    END;

    PROCEDURE ADICIONAR(p_id NUMBER) IS
        v_idx NUMBER;
    BEGIN
        -- Evitar duplicados simples (opcional, mas bom para performance)
        v_idx := v_ids_matricula.COUNT + 1;
        v_ids_matricula(v_idx) := p_id;
    END;
END PKG_BUFFER_MATRICULA;
```

Figura 222 - PACKAGE "PKG_BUFFER_MATRICULA", Parte 1.

```
-- Buffer para cálculo de notas finais
CREATE OR REPLACE PACKAGE PKG_BUFFER_NOTA IS
    -- Listas simples de números (Arrays)
    TYPE t_lista_numeros IS TABLE OF NUMBER;

    -- Listas para guardar os pares (Inscrição, Avaliação Pai)
    v_ids_inscricao t_lista_numeros : t_lista_numeros();
    v_ids_pais      t_lista_numeros : t_lista_numeros();

    -- Lista para guardar inscrições que precisam de nota final
    v_ids_finais   t_lista_numeros : t_lista_numeros();

    -- Flag para evitar recursividade nos triggers
    g_a_calcular BOOLEAN : FALSE;

    PROCEDURE ADICIONAR_PAI(p_insc_id IN NUMBER, p_pai_id IN NUMBER);
    PROCEDURE ADICIONAR_FINAL(p_insc_id IN NUMBER);
    PROCEDURE LIMPAR;
END PKG_BUFFER_NOTA;

CREATE OR REPLACE PACKAGE BODY PKG_BUFFER_NOTA IS

    PROCEDURE ADICIONAR_PAI(p_insc_id IN NUMBER, p_pai_id IN NUMBER) IS
    BEGIN
        -- Adiciona ao final da lista (Simples e direto)
        v_ids_inscricao.EXTEND;
        v_ids_pais.EXTEND;

        v_ids_inscricao(v_ids_inscricao.LAST) := p_insc_id;
        v_ids_pais(v_ids_pais.LAST) := p_pai_id;
    END ADICIONAR_PAI;

    PROCEDURE ADICIONAR_FINAL(p_insc_id IN NUMBER) IS
    BEGIN
        v_ids_finais.EXTEND;
        v_ids_finais(v_ids_finais.LAST) := p_insc_id;
    END ADICIONAR_FINAL;

    PROCEDURE LIMPAR IS
    BEGIN
        -- Esvazia as listas
        v_ids_inscricao.DELETE;
        v_ids_pais.DELETE;
        v_ids_finais.DELETE;
    END LIMPAR;
END PKG_BUFFER_NOTA;
```

Figura 223 - PACKAGE "PKG_BUFFER_NOTA", Parte 2.

```
-- Buffer para gerir lista de inscrições a processar
CREATE OR REPLACE PACKAGE PKG_BUFFER_INSCRICAO IS
    TYPE r_insc IS RECORD (
        matricula_id NUMBER,
        turma_id      NUMBER,
        uc_id         NUMBER,
        ects          NUMBER,
        ano_letivo    VARCHAR2(10),
        curso_id      NUMBER
    );
    TYPE t_insc IS TABLE OF r_insc;
    v_lista_inscricoes t_insc : t_insc();

    PROCEDURE LIMPAR;
    PROCEDURE ADICIONAR(
        p_mat_id NUMBER,
        p_tur_id NUMBER,
        p_uc_id NUMBER,
        p_ects NUMBER,
        p_ano VARCHAR2,
        p_cur_id NUMBER
    );
END PKG_BUFFER_INSCRICAO;

CREATE OR REPLACE PACKAGE BODY PKG_BUFFER_INSCRICAO IS
    PROCEDURE LIMPAR IS
    BEGIN
        v_lista_inscricoes.DELETE;
    END;

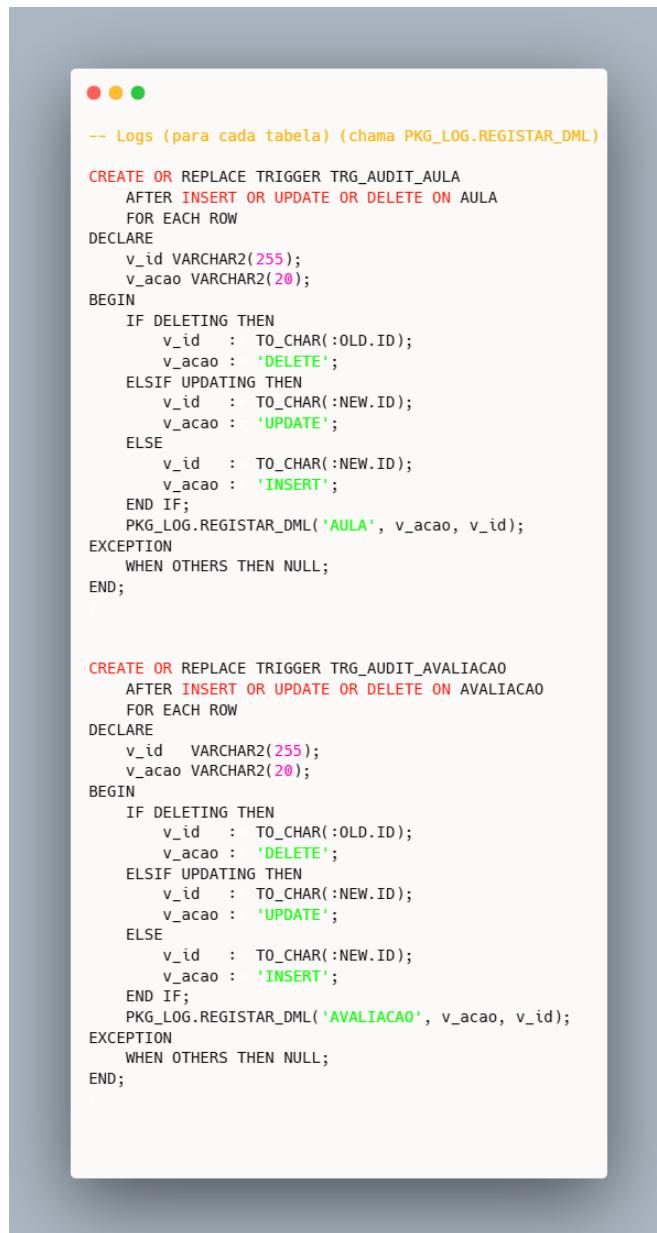
    PROCEDURE ADICIONAR(
        p_mat_id NUMBER,
        p_tur_id NUMBER,
        p_uc_id NUMBER,
        p_ects NUMBER,
        p_ano VARCHAR2,
        p_cur_id NUMBER
    ) IS
    BEGIN
        v_lista_inscricoes.EXTEND;
        v_lista_inscricoes(v_lista_inscricoes.LAST).matricula_id := p_mat_id;
        v_lista_inscricoes(v_lista_inscricoes.LAST).turma_id := p_tur_id;
        v_lista_inscricoes(v_lista_inscricoes.LAST).uc_id := p_uc_id;
        v_lista_inscricoes(v_lista_inscricoes.LAST).ects := p_ects;
        v_lista_inscricoes(v_lista_inscricoes.LAST).ano_letivo := p_ano;
        v_lista_inscricoes(v_lista_inscricoes.LAST).curso_id := p_cur_id;
    END;
END PKG_BUFFER_INSCRICAO;
```

Figura 224 - PACKAGE "PKG_BUFFER_INSCRICAO".

TRIGGERS de Auditoria Automática (DML)

Esta secção volumosa de código contém a definição de *TRIGGERS* para todas as tabelas do sistema, disparados após inserção, atualização ou remoção (*AFTER INSERT OR UPDATE OR DELETE*).

A função destes blocos é capturar o tipo de operação e o identificador do registo afetado, para de seguida invocar o “PKG_LOG” que cria um rastro de auditoria detalhada sem intervenção do próprio utilizador.



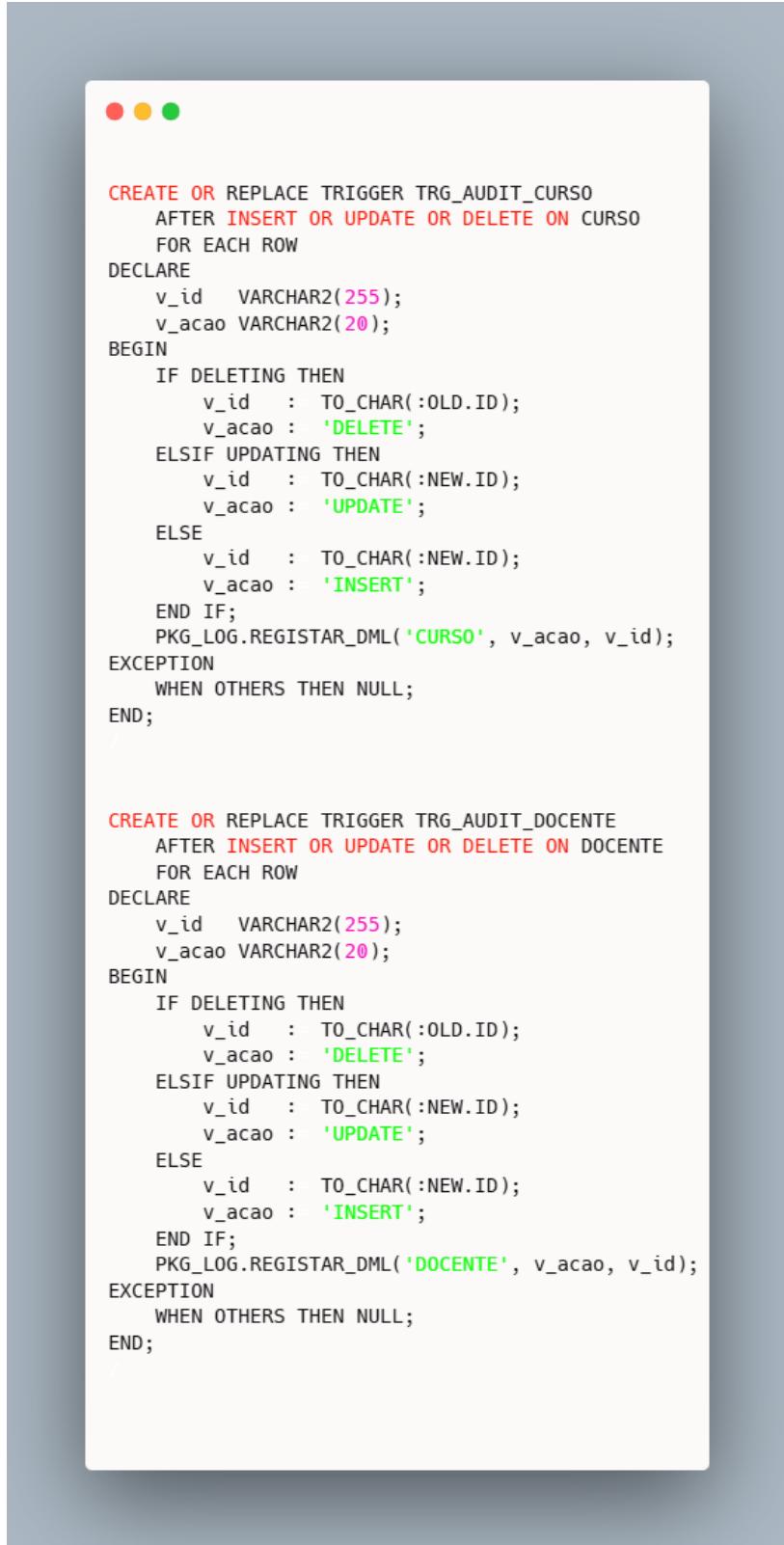
```
● ● ●

-- Logs (para cada tabela) (chama PKG_LOG.REGISTAR_DML)

CREATE OR REPLACE TRIGGER TRG_AUDIT_AULA
    AFTER INSERT OR UPDATE OR DELETE ON AULA
    FOR EACH ROW
DECLARE
    v_id VARCHAR2(255);
    v_acao VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('AULA', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

CREATE OR REPLACE TRIGGER TRG_AUDIT_AVALIACAO
    AFTER INSERT OR UPDATE OR DELETE ON AVALIACAO
    FOR EACH ROW
DECLARE
    v_id   VARCHAR2(255);
    v_acao VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('AVALIACAO', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
```

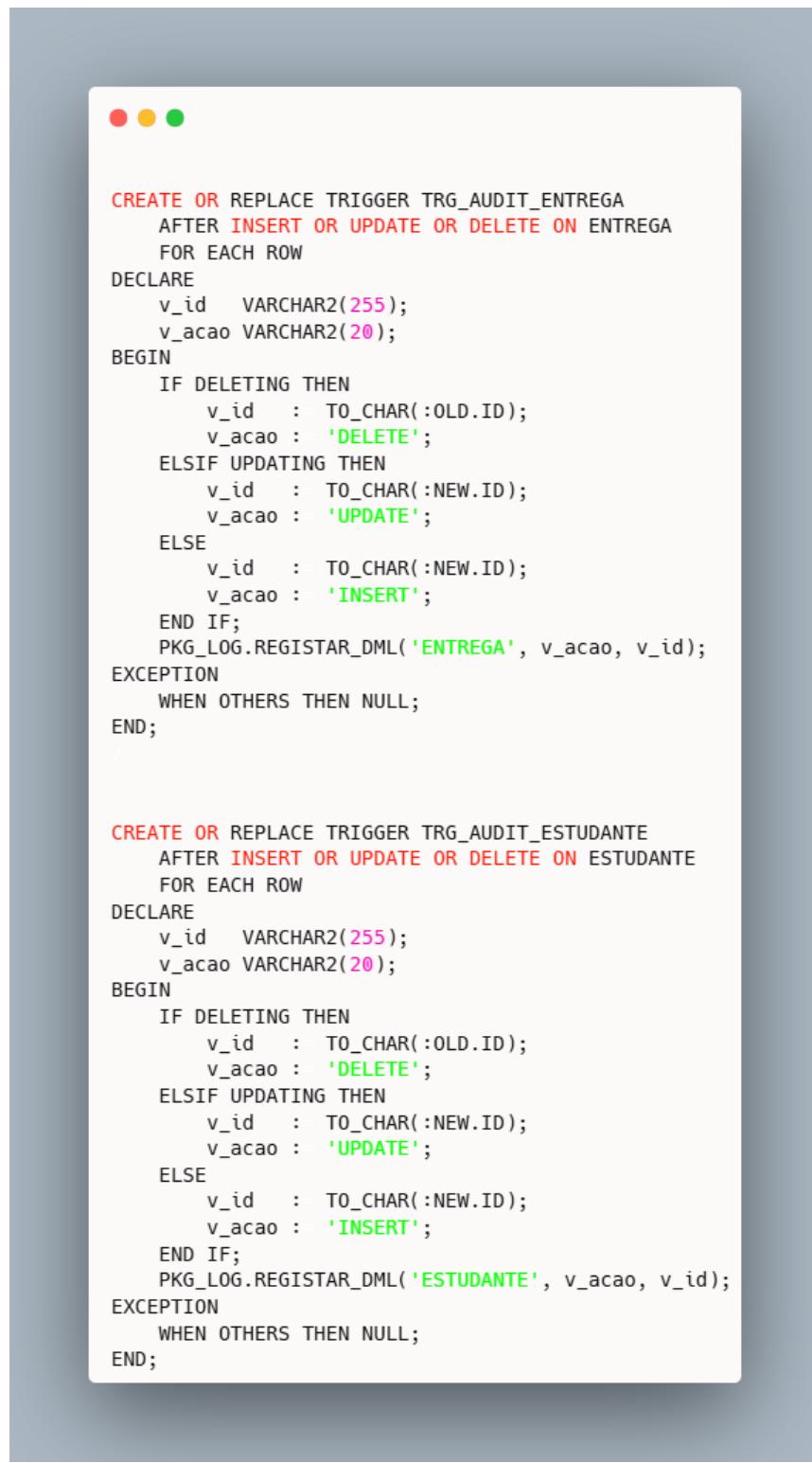
Figura 225 - TRIGGERS de Auditoria "aula" e "avaliacao".



```
CREATE OR REPLACE TRIGGER TRG_AUDIT_CURSO
AFTER INSERT OR UPDATE OR DELETE ON CURSO
FOR EACH ROW
DECLARE
    v_id    VARCHAR2(255);
    v_acao VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id    : TO_CHAR(:OLD.ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id    : TO_CHAR(:NEW.ID);
        v_acao : 'UPDATE';
    ELSE
        v_id    : TO_CHAR(:NEW.ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('CURSO', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

CREATE OR REPLACE TRIGGER TRG_AUDIT_DOCENTE
AFTER INSERT OR UPDATE OR DELETE ON DOCENTE
FOR EACH ROW
DECLARE
    v_id    VARCHAR2(255);
    v_acao VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id    : TO_CHAR(:OLD.ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id    : TO_CHAR(:NEW.ID);
        v_acao : 'UPDATE';
    ELSE
        v_id    : TO_CHAR(:NEW.ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('DOCENTE', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
```

Figura 226 - TRIGGER Auditoria "curso" e "docente".



```
CREATE OR REPLACE TRIGGER TRG_AUDIT_ENTREGA
AFTER INSERT OR UPDATE OR DELETE ON ENTREGA
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   := TO_CHAR(:OLD.ID);
        v_acao := 'DELETE';
    ELSIF UPDATING THEN
        v_id   := TO_CHAR(:NEW.ID);
        v_acao := 'UPDATE';
    ELSE
        v_id   := TO_CHAR(:NEW.ID);
        v_acao := 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('ENTREGA', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

CREATE OR REPLACE TRIGGER TRG_AUDIT_ESTUDANTE
AFTER INSERT OR UPDATE OR DELETE ON ESTUDANTE
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   := TO_CHAR(:OLD.ID);
        v_acao := 'DELETE';
    ELSIF UPDATING THEN
        v_id   := TO_CHAR(:NEW.ID);
        v_acao := 'UPDATE';
    ELSE
        v_id   := TO_CHAR(:NEW.ID);
        v_acao := 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('ESTUDANTE', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
```

Figura 227 - TRIGGERS Auditoria "entrega" e "estudante".

```
CREATE OR REPLACE TRIGGER TRG_AUDIT_ESTUDANTE_ENTREGA
  AFTER INSERT OR UPDATE OR DELETE ON ESTUDANTE_ENTREGA
  FOR EACH ROW
DECLARE
  v_id    VARCHAR2(255);
  v_acao VARCHAR2(20);
BEGIN
  IF DELETING THEN
    v_id    : TO_CHAR(:OLD.ENTREGA_ID);           TO_CHAR(:OLD.INSCRICAO_ID);
    v_acao : 'DELETE';
  ELSIF UPDATING THEN
    v_id    : TO_CHAR(:NEW.ENTREGA_ID);           TO_CHAR(:NEW.INSCRICAO_ID);
    v_acao : 'UPDATE';
  ELSE
    v_id    : TO_CHAR(:NEW.ENTREGA_ID);           TO_CHAR(:NEW.INSCRICAO_ID);
    v_acao : 'INSERT';
  END IF;
  PKG_LOG.REGISTAR_DML('ESTUDANTE_ENTREGA', v_acao, v_id);
EXCEPTION
  WHEN OTHERS THEN NULL;
END;

CREATE OR REPLACE TRIGGER TRG_AUDIT_FICHEIRO_ENTREGA
  AFTER INSERT OR UPDATE OR DELETE ON FICHEIRO_ENTREGA
  FOR EACH ROW
DECLARE
  v_id    VARCHAR2(255);
  v_acao VARCHAR2(20);
BEGIN
  IF DELETING THEN
    v_id    : TO_CHAR(:OLD.ID);
    v_acao : 'DELETE';
  ELSIF UPDATING THEN
    v_id    : TO_CHAR(:NEW.ID);
    v_acao : 'UPDATE';
  ELSE
    v_id    : TO_CHAR(:NEW.ID);
    v_acao : 'INSERT';
  END IF;
  PKG_LOG.REGISTAR_DML('FICHEIRO_ENTREGA', v_acao, v_id);
EXCEPTION
  WHEN OTHERS THEN NULL;
END;
```

Figura 228 - TRIGGERS Audit. "estudante_entrega" e "ficheiro_entrega".

```

CREATE OR REPLACE TRIGGER TRG_AUDIT_FICHEIRO_RECURSO
AFTER INSERT OR UPDATE OR DELETE ON FICHEIRO_RECURSO
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('FICHEIRO_RECURSO', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

CREATE OR REPLACE TRIGGER TRG_AUDIT_INSCRICAO
AFTER INSERT OR UPDATE OR DELETE ON INSCRICAO
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('INSCRICAO', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

CREATE OR REPLACE TRIGGER TRG_AUDIT_MATRICULA
AFTER INSERT OR UPDATE OR DELETE ON MATRICULA
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('MATRICULA', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

```

Figura 229 - TRIGGERS Audit. "ficheiro_recurso", "inscricao" e "matricula".

```

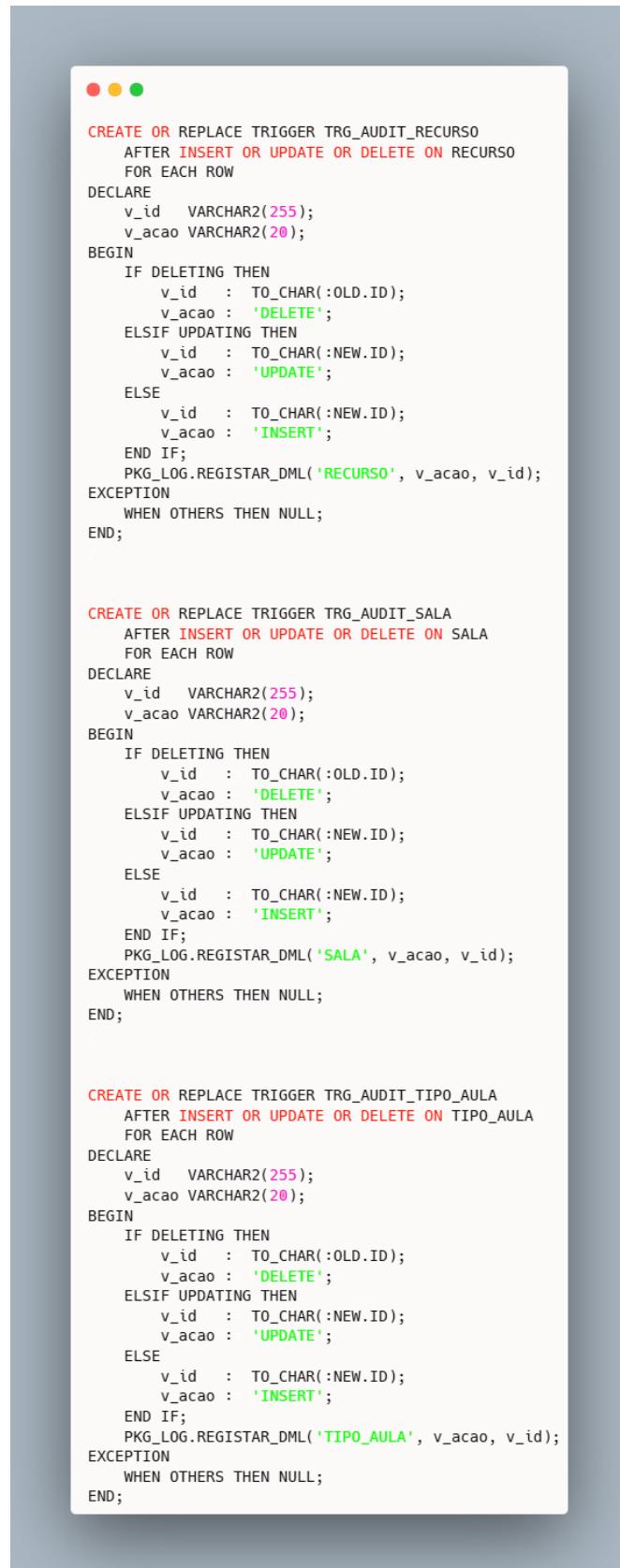
CREATE OR REPLACE TRIGGER TRG_AUDIT_NOTA
AFTER INSERT OR UPDATE OR DELETE ON NOTA
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.AVALIACAO_ID)      TO_CHAR(:OLD.INSCRICAO_ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.AVALIACAO_ID)      TO_CHAR(:NEW.INSCRICAO_ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.AVALIACAO_ID)      TO_CHAR(:NEW.INSCRICAO_ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('NOTA', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

CREATE OR REPLACE TRIGGER TRG_AUDIT_PARCELA_PROPINA
AFTER INSERT OR UPDATE OR DELETE ON PARCELA_PROPINA
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('PARCELA_PROPINA', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

CREATE OR REPLACE TRIGGER TRG_AUDIT_PRESENCA
AFTER INSERT OR UPDATE OR DELETE ON PRESENCA
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.AULA_ID)      TO_CHAR(:OLD.INSCRICAO_ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.AULA_ID)      TO_CHAR(:NEW.INSCRICAO_ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.AULA_ID)      TO_CHAR(:NEW.INSCRICAO_ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('PRESENCA', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

```

Figura 230 - TRIGGERS Audit. "nota", "parcela_propina" e "presenca".



```

CREATE OR REPLACE TRIGGER TRG_AUDIT_RECURSO
    AFTER INSERT OR UPDATE OR DELETE ON RECURSO
    FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('RECURSO', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

CREATE OR REPLACE TRIGGER TRG_AUDIT_SALA
    AFTER INSERT OR UPDATE OR DELETE ON SALA
    FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('SALA', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

CREATE OR REPLACE TRIGGER TRG_AUDIT_TIPO_AULA
    AFTER INSERT OR UPDATE OR DELETE ON TIPO_AULA
    FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('TIPO_AULA', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

```

Figura 231 - TRIGGERS Audit. "recurso", "sala" e "tipo_aula".

```

CREATE OR REPLACE TRIGGER TRG_AUDIT_TIPO_AVALIACAO
AFTER INSERT OR UPDATE OR DELETE ON TIPO_AVALIACAO
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id    := TO_CHAR(:OLD.ID);
        v_acao := 'DELETE';
    ELSIF UPDATING THEN
        v_id    := TO_CHAR(:NEW.ID);
        v_acao := 'UPDATE';
    ELSE
        v_id    := TO_CHAR(:NEW.ID);
        v_acao := 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('TIPO_AVALIACAO', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

CREATE OR REPLACE TRIGGER TRG_AUDIT_TIPO_CURSO
AFTER INSERT OR UPDATE OR DELETE ON TIPO_CURSO
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id    := TO_CHAR(:OLD.ID);
        v_acao := 'DELETE';
    ELSIF UPDATING THEN
        v_id    := TO_CHAR(:NEW.ID);
        v_acao := 'UPDATE';
    ELSE
        v_id    := TO_CHAR(:NEW.ID);
        v_acao := 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('TIPO_CURSO', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

CREATE OR REPLACE TRIGGER TRG_AUDIT_TURMA
AFTER INSERT OR UPDATE OR DELETE ON TURMA
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id    := TO_CHAR(:OLD.ID);
        v_acao := 'DELETE';
    ELSIF UPDATING THEN
        v_id    := TO_CHAR(:NEW.ID);
        v_acao := 'UPDATE';
    ELSE
        v_id    := TO_CHAR(:NEW.ID);
        v_acao := 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('TURMA', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

```

Figura 232 - TRIGGERS Audit. "tipo_avaliação", "curso" e "turma".

```
CREATE OR REPLACE TRIGGER TRG_AUDIT_UC_CURSO
AFTER INSERT OR UPDATE OR DELETE ON UC_CURSO
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.CURSO_ID)      TO_CHAR(:OLD.UNIDADE_CURRICULAR_ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.CURSO_ID)      TO_CHAR(:NEW.UNIDADE_CURRICULAR_ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.CURSO_ID)      TO_CHAR(:NEW.UNIDADE_CURRICULAR_ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('UC_CURSO', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

CREATE OR REPLACE TRIGGER TRG_AUDIT_UC_DOCENTE
AFTER INSERT OR UPDATE OR DELETE ON UC_DOCENTE
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.UNIDADE_CURRICULAR_ID)      TO_CHAR(:OLD.DOCENTE_ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.UNIDADE_CURRICULAR_ID)      TO_CHAR(:NEW.DOCENTE_ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.UNIDADE_CURRICULAR_ID)      TO_CHAR(:NEW.DOCENTE_ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('UC_DOCENTE', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
```

Figura 233 - TRIGGERS Audit. "uc_curso" e "uc_docente".



```
CREATE OR REPLACE TRIGGER TRG_AUDIT_UNIDADE_CURRICULAR
AFTER INSERT OR UPDATE OR DELETE ON UNIDADE_CURRICULAR
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('UNIDADE_CURRICULAR', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;

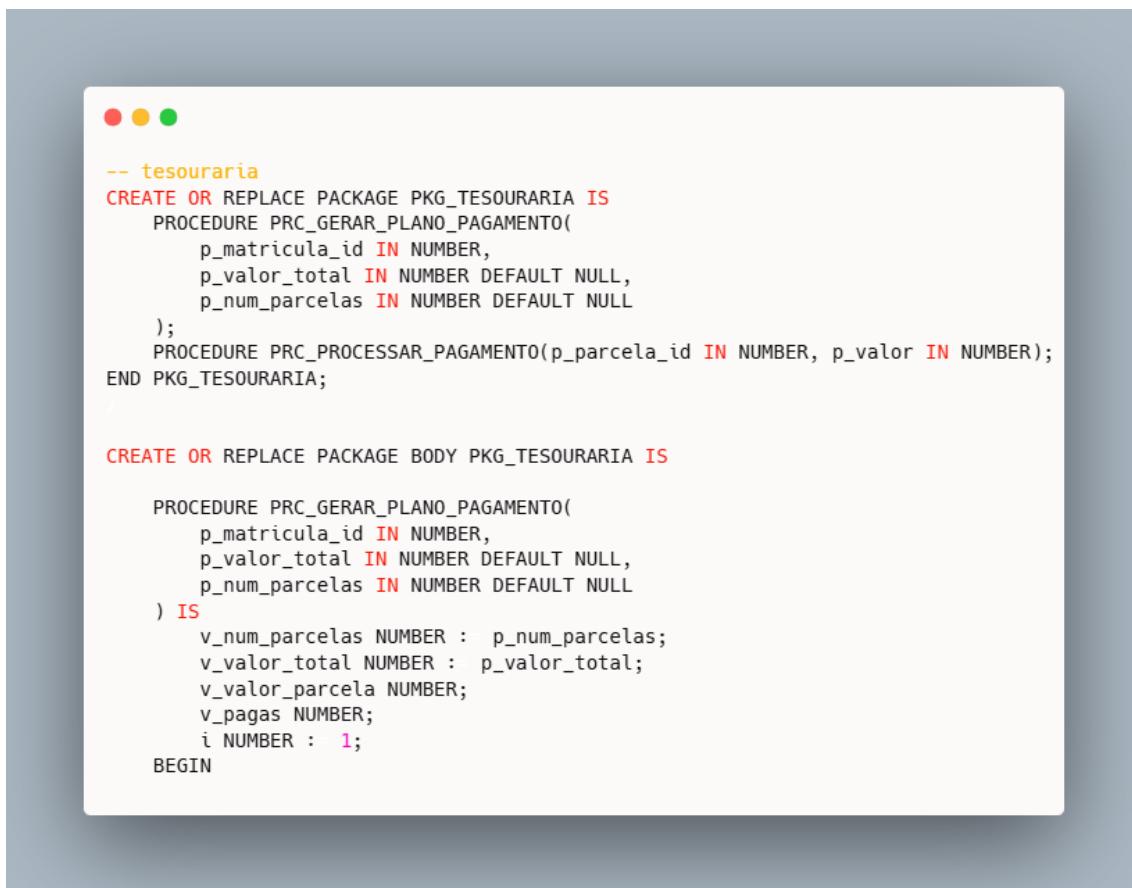
CREATE OR REPLACE TRIGGER TRG_AUDIT_FICHEIRO_RECURSO
AFTER INSERT OR UPDATE OR DELETE ON FICHEIRO_RECURSO
FOR EACH ROW
DECLARE
    v_id      VARCHAR2(255);
    v_acao    VARCHAR2(20);
BEGIN
    IF DELETING THEN
        v_id   : TO_CHAR(:OLD.ID);
        v_acao : 'DELETE';
    ELSIF UPDATING THEN
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'UPDATE';
    ELSE
        v_id   : TO_CHAR(:NEW.ID);
        v_acao : 'INSERT';
    END IF;
    PKG_LOG.REGISTAR_DML('FICHEIRO_RECURSO', v_acao, v_id);
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
```

Figura 234 - TRIGGERS Audit. "unidade_curricular" e "ficheiro_recurso".



PACKAGE de Tesouraria e Pagamentos

O PACKAGE “PKG_TESOURARIA” contém a lógica financeira. O PROCEDURE “PRC_GERAR_PLANO_PAGAMENTO” executa a operação de dividir o valor total das propinas pelo número de parcelas a pagar, inserindo os registo de dívida futura, como no *Inforestudante*. O PROCEDURE “PRC_PROCESSAR_PAGAMENTO” gera a liquidação, verifica a data atual face ao vencimento e aplica, se necessário, o cálculo da multa, ao definir nas constantes antes de fechar a parcela da propina como paga.



```
-- tesouraria
CREATE OR REPLACE PACKAGE PKG_TESOURARIA IS
    PROCEDURE PRC_GERAR_PLANO_PAGAMENTO(
        p_matricula_id IN NUMBER,
        p_valor_total IN NUMBER DEFAULT NULL,
        p_num_parcelas IN NUMBER DEFAULT NULL
    );
    PROCEDURE PRC_PROCESSAR_PAGAMENTO(p_parcela_id IN NUMBER, p_valor IN NUMBER);
END PKG_TESOURARIA;

CREATE OR REPLACE PACKAGE BODY PKG_TESOURARIA IS

    PROCEDURE PRC_GERAR_PLANO_PAGAMENTO(
        p_matricula_id IN NUMBER,
        p_valor_total IN NUMBER DEFAULT NULL,
        p_num_parcelas IN NUMBER DEFAULT NULL
    ) IS
        v_num_parcelas NUMBER : p_num_parcelas;
        v_valor_total NUMBER : p_valor_total;
        v_valor_parcela NUMBER;
        v_pagas NUMBER;
        i NUMBER : 1;
    BEGIN
```

Figura 235 - PACKAGES Tesouraria, Parte 1.

```

-- Verificar se já existem pagamentos para não sobre escrever plano ativo com histórico
    SELECT COUNT( ) INTO v_pagas FROM parcela_propina WHERE matricula_id
    p_matricula_id AND estado = '1';
    IF v_pagas = 0 THEN
        PKG_LOG.ALERTA('Plano de pagamento não gerado: Já existem parcelas pagas.', 'PARCELA_PROPINA');
        RETURN;
    END IF;

    DELETE FROM parcela_propina WHERE matricula_id = p_matricula_id;

    -- Se os valores não foram passados (chamada manual), busca na base
    IF v_valor_total IS NULL OR v_num_parcelas IS NULL THEN
        SELECT tc.valor_propinas, m.numero_parcelas
        INTO v_valor_total, v_num_parcelas
        FROM matricula m
        JOIN curso c ON m.curso_id = c.id
        JOIN tipo_curso tc ON c.tipo_curso_id = tc.id
        WHERE m.id = p_matricula_id;
    END IF;

    DBMS_OUTPUT.PUT_LINE('Valor Total: ' || v_valor_total || ', Parcelas: ' || v_num_parcelas);

    IF v_num_parcelas = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Número de parcelas inválido.');
        RETURN;
    END IF;

    v_valor.Parcela := v_valor_total / v_num_parcelas;

    LOOP
        EXIT WHEN i = v_num_parcelas;
        INSERT INTO parcela_propina (valor, data_vencimento, numero, estado,
        matricula_id, status)
        VALUES (v_valor.Parcela, ADD_MONTHS(SYSDATE, i), i, '0', p_matricula_id,
        '1');
        i := i + 1;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('Plano gerado com sucesso: ' || v_num_parcelas || ' parcelas.');

    EXCEPTION
        WHEN OTHERS THEN
            PKG_LOG.ERROR('PKG_TESOURARIA.PRC_GERAR_PLANO_PAGAMENTO: ' || SQLERRM,
            'PARCELA_PROPINA');
            RAISE;
    END PRC_GERAR_PLANO_PAGAMENTO;

    PROCEDURE PRC_PROCESSAR_PAGAMENTO(p_parcela_id IN NUMBER, p_valor IN NUMBER) IS
        v_vencimento DATE;
        v_orig NUMBER;
        v_multa NUMBER := 0;
    BEGIN
        SELECT data_vencimento, valor INTO v_vencimento, v_orig FROM parcela_propina
        WHERE id = p_parcela_id;
        IF SYSDATE > v_vencimento THEN
            v_multa := v_orig * PKG_CONSTANTES.TAXA_MULTA_ATRASO;
            PKG_LOG.ALERTA('Multa de ' || v_multa || ' aplicada.', 'PARCELA_PROPINA');
        END IF;
        UPDATE parcela_propina SET estado = '1', data_pagamento = SYSDATE, valor =
        v_orig + v_multa, updated_at = SYSDATE WHERE id = p_parcela_id;
    EXCEPTION
        WHEN OTHERS THEN
            PKG_LOG.ERROR('PKG_TESOURARIA.PRC_PROCESSAR_PAGAMENTO: ' || SQLERRM,
            'PARCELA_PROPINA');
    END PRC_PROCESSAR_PAGAMENTO;
END PKG_TESOURARIA;

```

Figura 236 - PACKAGES Tesouraria, Parte 2.



Validação e Cálculo de Notas

O seguinte bloco de código gera a integridade das notas. O *TRIGGER* “TRG_VAL_NOTAS” valida os limites da nota (0 a 20) e a consistência entre turmas. Segue-se um sistema de três *TRIGGERS* (“TRG_NOTA_MEDIA_BS”, “TRG_NOTA_MEDIA_AR” e “TRG_NOTA_MEDIA_AS”). A operação conjunta destes três *TRIGGERS* deteta alterações em notas parciais, armazena os ID's no *buffer* e, no final da transação, volta a calcular a média ponderada da avaliação “pai” com base nos pesos definidos.

```

● ● ●

-- CALCULO DE MÉDIAS
-- 1 - CÁLCULO DE MÉDIAS - TRIGGER 1: INICIALIZAÇÃO
CREATE OR REPLACE TRIGGER TRG_NOTA_MEDIA_BS
BEFORE INSERT OR UPDATE ON NOTA
BEGIN
    PKG_BUFFER_NOTA.LIMPAR;
END;

-- 2 - CÁLCULO DE MÉDIAS - TRIGGER 2: Verificação e Armazenamento
CREATE OR REPLACE TRIGGER TRG_NOTA_MEDIA_AR
AFTER INSERT OR UPDATE ON NOTA
FOR EACH ROW
DECLARE
    v_pai_id NUMBER;
BEGIN
    IF NOT PKG_BUFFER_NOTA.g_a_calcular THEN
        SELECT avaliacao_pai_id INTO v_pai_id
        FROM avaliacao
        WHERE id = :NEW.avaliacao_id;

        IF v_pai_id IS NOT NULL THEN
            PKG_BUFFER_NOTA.ADICIONAR_PAIS(:NEW.inscricao_id, v_pai_id);
        END IF;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN NULL;
    WHEN OTHERS THEN NULL;
END;

-- 3 - CÁLCULO DE MÉDIAS - TRIGGER 3: PROCESSAMENTO
CREATE OR REPLACE TRIGGER TRG_NOTA_MEDIA_AS
AFTER INSERT OR UPDATE ON NOTA
DECLARE
    v_nota_final NUMBER;
    v_idx NUMBER;
    v_ins_id NUMBER;
    v_pai_id NUMBER;
    CURSOR c_calculo(p_ins_id NUMBER, p_pai_id NUMBER) IS
        SELECT NVL(SUM(n.nota * a.peso), 0) -- peso pode ser null
        FROM nota n
        JOIN avaliacao a ON n.avaliacao_id = a.id
        WHERE a.avaliacao_pai_id = p_pai_id
        AND n.inscricao_id = p_ins_id;
BEGIN
    IF PKG_BUFFER_NOTA.v_ids_inscricao.COUNT > 0 THEN
        PKG_BUFFER_NOTA.g_a_calcular := TRUE;

        v_idx := PKG_BUFFER_NOTA.v_ids_inscricao.FIRST;
        LOOP
            EXIT WHEN v_idx IS NULL;

            v_ins_id := PKG_BUFFER_NOTA.v_ids_inscricao(v_idx);
            v_pai_id := PKG_BUFFER_NOTA.v_ids_pais(v_idx);

            -- Abrir cursor para calcular média
            OPEN c_calculo(v_ins_id, v_pai_id);
            FETCH c_calculo INTO v_nota_final;
            CLOSE c_calculo;

            -- Atualizar a nota do pai
            UPDATE nota
            SET nota = v_nota_final
            WHERE inscricao_id = v_ins_id AND avaliacao_id = v_pai_id;

            -- Se não atualizou, insere
            IF SQL_ROWCOUNT = 0 THEN
                INSERT INTO nota (inscricao_id, avaliacao_id, nota)
                VALUES (v_ins_id, v_pai_id, v_nota_final);
            END IF;

            v_idx := PKG_BUFFER_NOTA.v_ids_inscricao.NEXT(v_idx);
        END LOOP;

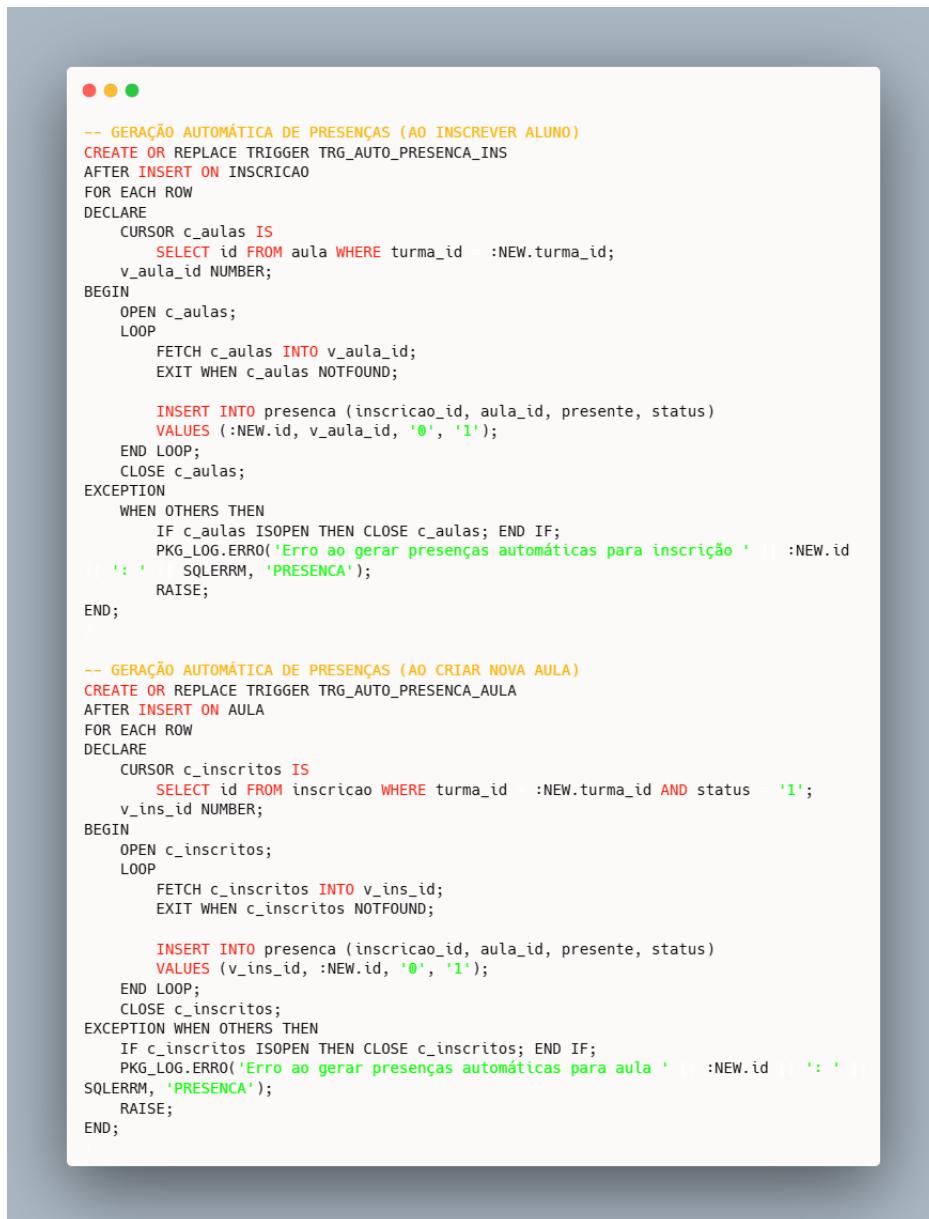
        PKG_BUFFER_NOTA.LIMPAR;
        PKG_BUFFER_NOTA.g_a_calcular := FALSE;
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        PKG_BUFFER_NOTA.g_a_calcular := FALSE;
        IF c_calculo ISOPEN THEN CLOSE c_calculo; END IF;
        PKG_LOG.ERRO('Erro no processamento de mediadas: ' || SQLERRM, 'NOTA');
    END;
END;

```

Figura 237 - TRIGGERS Valid. e Cálculo de Notas.

Gestão de Presenças e Regras de Avaliação

Neste tópico de operações são geridas as presenças e as submissões. Este bloco de código contém a automação que cria linhas de presença vazias ao inscrever alunos ou criar aulas. Valida-se também a hierarquia das avaliações, controlam-se as entregas de trabalhos, impedindo que o limite de alunos por grupo seja excedido e por consequência gera-se alertas automáticos caso a data de entrega do ficheiro seja posterior à data limite definida.



```
-- GERAÇÃO AUTOMÁTICA DE PRESENÇAS (AO INSCREVER ALUNO)
CREATE OR REPLACE TRIGGER TRG_AUTO_PRESENCA_INS
AFTER INSERT ON INSCRICAO
FOR EACH ROW
DECLARE
    CURSOR c_aulas IS
        SELECT id FROM aula WHERE turma_id = :NEW.turma_id;
    v_aula_id NUMBER;
BEGIN
    OPEN c_aulas;
    LOOP
        FETCH c_aulas INTO v_aula_id;
        EXIT WHEN c_aulas NOTFOUND;

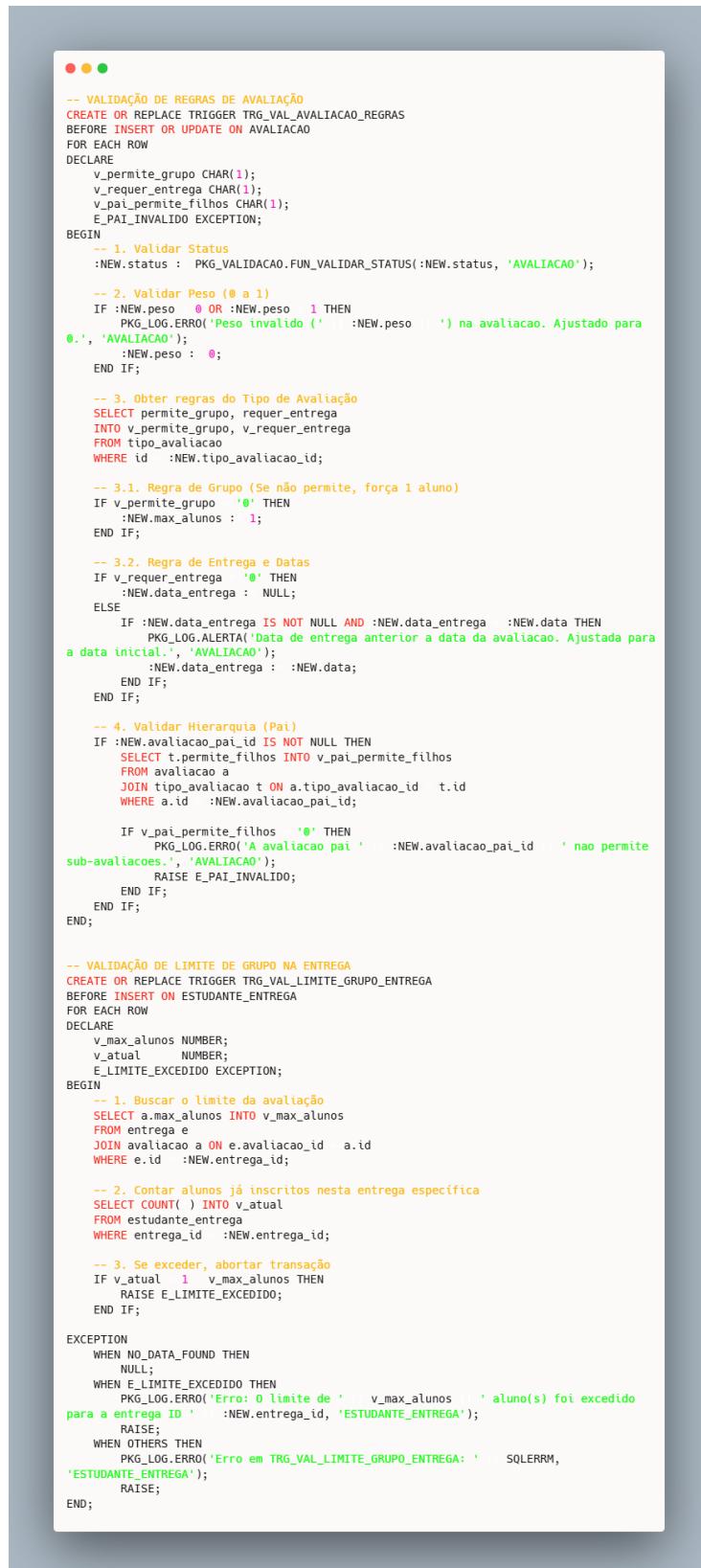
        INSERT INTO presenca (inscricao_id, aula_id, presente, status)
        VALUES (:NEW.id, v_aula_id, '0', '1');
    END LOOP;
    CLOSE c_aulas;
EXCEPTION
    WHEN OTHERS THEN
        IF c_aulas ISOPEN THEN CLOSE c_aulas; END IF;
        PKG_LOG.ERRO('Erro ao gerar presenças automáticas para inscrição ' || :NEW.id
        || ': ' || SQLERRM, 'PRESENCA');
        RAISE;
END;

-- GERAÇÃO AUTOMÁTICA DE PRESENÇAS (AO CRIAR NOVA AULA)
CREATE OR REPLACE TRIGGER TRG_AUTO_PRESENCA_AULA
AFTER INSERT ON AULA
FOR EACH ROW
DECLARE
    CURSOR c_inscritos IS
        SELECT id FROM inscricao WHERE turma_id = :NEW.turma_id AND status = '1';
    v_ins_id NUMBER;
BEGIN
    OPEN c_inscritos;
    LOOP
        FETCH c_inscritos INTO v_ins_id;
        EXIT WHEN c_inscritos NOTFOUND;

        INSERT INTO presenca (inscricao_id, aula_id, presente, status)
        VALUES (v_ins_id, :NEW.id, '0', '1');
    END LOOP;
    CLOSE c_inscritos;
EXCEPTION WHEN OTHERS THEN
    IF c_inscritos ISOPEN THEN CLOSE c_inscritos; END IF;
    PKG_LOG.ERRO('Erro ao gerar presenças automáticas para aula ' || :NEW.id
    || ': ' || SQLERRM, 'PRESENCA');
    RAISE;
END;
```

Figura 238 - TRIGGERS Gestão de Presenças.

Instituto Politécnico de Coimbra
Instituto Superior de Engenharia de Coimbra



```

-- VALIDAÇÃO DE REGRAS DE AVALIAÇÃO
CREATE OR REPLACE TRIGGER TRG_VAL_AVALIACAO_REGRAS
BEFORE INSERT OR UPDATE ON AVALIACAO
FOR EACH ROW
DECLARE
    v_permite_grupo CHAR(1);
    v_requer_entrega CHAR(1);
    v_pai_permite_filhos CHAR(1);
    E_PAIS_INVALIDO EXCEPTION;
BEGIN
    -- 1. Validar Status
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'AVALIACAO');

    -- 2. Validar Peso (@ a 1)
    IF :NEW.peso = 0 OR :NEW.peso < 1 THEN
        PKG_LOG.ERROR('Peso invalido (' || :NEW.peso || ') na avaliacao. Ajustado para
0.', 'AVALIACAO');
        :NEW.peso := 0;
    END IF;

    -- 3. Obter regras do Tipo de Avaliação
    SELECT permite_grupo, requer_entrega
    INTO v_permite_grupo, v_requer_entrega
    FROM tipo_avaliacao
    WHERE id = :NEW.tipo_avaliacao_id;

    -- 3.1. Regra de Grupo (Se não permite, força 1 aluno)
    IF v_permite_grupo = '0' THEN
        :NEW.max_alunos := 1;
    END IF;

    -- 3.2. Regra de Entrega e Datas
    IF v_requer_entrega = '0' THEN
        :NEW.data_entrega := NULL;
    ELSE
        IF :NEW.data_entrega IS NOT NULL AND :NEW.data_entrega < :NEW.data THEN
            PKG_LOG.ALERTA('Data de entrega anterior a data da avaliação. Ajustada para
a data inicial.', 'AVALIACAO');
            :NEW.data_entrega := :NEW.data;
        END IF;
    END IF;

    -- 4. Validar Hierarquia (Pai)
    IF :NEW.avaliacao_pai_id IS NOT NULL THEN
        SELECT t.permite_filhos INTO v_pai_permite_filhos
        FROM avaliacao a
        JOIN tipo_avaliacao t ON a.tipo_avaliacao_id = t.id
        WHERE a.id = :NEW.avaliacao_pai_id;

        IF v_pai_permite_filhos = '0' THEN
            PKG_LOG.ERROR('A avaliação pai (' || :NEW.avaliacao_pai_id || ') não permite
sub-avaliações.', 'AVALIACAO');
            RAISE E_PAIS_INVALIDO;
        END IF;
    END IF;
END;

-- VALIDAÇÃO DE LIMITE DE GRUPO NA ENTREGA
CREATE OR REPLACE TRIGGER TRG_VAL_LIMITE_GRUPO_ENTREGA
BEFORE INSERT ON ESTUDANTE_ENTREGA
FOR EACH ROW
DECLARE
    v_max_alunos NUMBER;
    v_atual NUMBER;
    E_LIMITE_EXCEDIDO EXCEPTION;
BEGIN
    -- 1. Buscar o limite da avaliação
    SELECT a.max_alunos INTO v_max_alunos
    FROM entrega e
    JOIN avaliacao a ON e.avaliacao_id = a.id
    WHERE e.id = :NEW.entrega_id;

    -- 2. Contar alunos já inscritos nesta entrega específica
    SELECT COUNT(1) INTO v_atual
    FROM estudante_entrega
    WHERE entrega_id = :NEW.entrega_id;

    -- 3. Se exceder, abortar transação
    IF v_atual > v_max_alunos THEN
        RAISE E_LIMITE_EXCEDIDO;
    END IF;

    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            NULL;
        WHEN E_LIMITE_EXCEDIDO THEN
            PKG_LOG.ERROR('Erro: O limite de ' || v_max_alunos || ' aluno(s) foi excedido
para a entrega ID ' || :NEW.entrega_id, 'ESTUDANTE_ENTREGA');
            RAISE;
        WHEN OTHERS THEN
            PKG_LOG.ERROR('Erro em TRG_VAL_LIMITE_GRUPO_ENTREGA: ' || SQLERRM,
'EESTUDANTE_ENTREGA');
            RAISE;
    END;
END;

```

Figura 239 - TRIGGERS Regras de Avaliação, Parte 1.

```
● ● ●

-- VALIDAÇÃO DE ENTREGAS
CREATE OR REPLACE TRIGGER TRG_VAL_ENTREGA_REGRAS
BEFORE INSERT OR UPDATE ON ENTREGA
FOR EACH ROW
DECLARE
    v_req_entrega CHAR(1);
    v_data_inicio DATE;
    v_data_fim      DATE;
BEGIN
    -- 1. Validar Status
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'ENTREGA');

    -- 2. Obter dados da Avaliação
    SELECT ta.requer_entrega, a.data, a.data_entrega
    INTO v_req_entrega, v_data_inicio, v_data_fim
    FROM avaliacao a
    JOIN tipo_avaliacao ta ON a.tipo_avaliacao_id = ta.id
    WHERE a.id = :NEW.avaliacao_id;

    -- 3. Validar se tipo requer entrega
    IF v_req_entrega = '0' THEN
        PKG_LOG.ALERTA('A avaliacao ' || :NEW.avaliacao_id || ' nao requer entrega de
ficheiros.', 'ENTREGA');
    END IF;

    -- 4. Validar Prazos (Gera Alerta, não impede - permite entrega atrasada mas
    regista)
    IF :NEW.data_entrega < v_data_inicio THEN
        PKG_LOG.ALERTA('Entrega efetuada ANTES da data de inicio da avaliacao (' ||
TO_CHAR(v_data_inicio, 'YYYY-MM-DD') || ').', 'ENTREGA');
    ELSIF v_data_fim IS NOT NULL AND :NEW.data_entrega < v_data_fim THEN
        PKG_LOG.ALERTA('Entrega FORA DO PRAZO. Limite era: ' || TO_CHAR(v_data_fim,
'YYYY-MM-DD HH24:MI'), 'ENTREGA');
    END IF;

    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            PKG_LOG.ERRO('Avaliacao ID ' || :NEW.avaliacao_id || ' nao encontrada ao
validar entrega.', 'ENTREGA');
        WHEN OTHERS THEN
            PKG_LOG.ERRO('Erro na validacao de entrega: ' || SQLERRM, 'ENTREGA');
    END;

```

Figura 240 - TRIGGERS Regras de Avaliação, Parte 2.

Validação de Dados Pessoais (Estudantes e Docentes)

Estes TRIGGERS asseguram a qualidade dos dados dos intervenientes. Para o estudante, verifica-se a idade mínima e a validade dos documentos de identificação. Para o docente, o sistema valida a data de contratação (impede naturalmente datas futuras) e a integridade dos contactos e documentos fiscais, acabando por bloquear a inserção caso as validações falhem.



```
● ● ●

-- VALIDAÇÃO DE DADOS (ESTUDANTE E DOCENTE)
CREATE OR REPLACE TRIGGER TRG_VAL_DADOS_ESTUDANTE
BEFORE INSERT OR UPDATE ON ESTUDANTE
FOR EACH ROW
DECLARE
    E_DADOS_INVALIDOS EXCEPTION;
BEGIN
    -- 1. Validar Status
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'ESTUDANTE');

    -- 2. Validar Data de Nascimento (Idade Mínima Parametrizada)
    IF :NEW.data_nascimento IS NULL OR :NEW.data_nascimento < ADD_MONTHS(SYSDATE,
PKG_CONSTANTES.IDADE_MINIMA_ESTUDANTE 12) THEN
        PKG_LOG.ERRO('Data de nascimento invalida ou idade inferior a '
PKG_CONSTANTES.IDADE_MINIMA_ESTUDANTE ' anos para aluno: ' || :NEW.nome,
'ESTUDANTE');
        RAISE E_DADOS_INVALIDOS;
    END IF;

    -- 3. Validar Telemóvel
    IF :NEW.telemovel IS NOT NULL AND NOT
PKG_VALIDACAO.FUN_VALIDAR_TELEMOVEL(:NEW.telemovel) THEN
        PKG_LOG.ERRO('Telemovel invalido: ' || :NEW.telemovel, 'ESTUDANTE');
        RAISE E_DADOS_INVALIDOS;
    END IF;

    -- 4. Validar Identidade (NIF, CC, Email, IBAN)
    IF NOT PKG_VALIDACAO.FUN_VALIDAR_NIF(:NEW.nif) THEN
        PKG_LOG.ERRO('NIF inválido para estudante: ' || :NEW.nif, 'ESTUDANTE');
        RAISE E_DADOS_INVALIDOS;
    END IF;

    IF :NEW.cc IS NOT NULL AND NOT PKG_VALIDACAO.FUN_VALIDAR_CC(:NEW.cc) THEN
        PKG_LOG.ERRO('CC inválido para estudante: ' || :NEW.cc, 'ESTUDANTE');
        RAISE E_DADOS_INVALIDOS;
    END IF;

    IF :NEW.email IS NOT NULL AND NOT PKG_VALIDACAO.FUN_VALIDAR_EMAIL(:NEW.email) THEN
        PKG_LOG.ERRO('Email inválido para estudante: ' || :NEW.email, 'ESTUDANTE');
        RAISE E_DADOS_INVALIDOS;
    END IF;

    IF :NEW.iban IS NOT NULL AND NOT PKG_VALIDACAO.FUN_VALIDAR_IBAN(:NEW.iban) THEN
        PKG_LOG.ERRO('IBAN inválido para estudante: ' || :NEW.iban, 'ESTUDANTE');
        RAISE E_DADOS_INVALIDOS;
    END IF;
EXCEPTION
    WHEN E_DADOS_INVALIDOS THEN RAISE;
    WHEN OTHERS THEN
        PKG_LOG.ERRO('Erro inesperado na validacao de estudante: ' || SQLERRM,
'ESTUDANTE');
        RAISE;
END;
```

Figura 241 - TRIGGER Valid. de Dados dos Estudantes.



```
CREATE OR REPLACE TRIGGER TRG_VAL_DADOS_DOCENTE
BEFORE INSERT OR UPDATE ON DOCENTE
FOR EACH ROW
DECLARE
    E_DADOS_INVALIDOS EXCEPTION;
BEGIN
    -- 1. Validar Status
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'DOCENTE');

    -- 2. Validar Data de Contratação (Não pode ser futura)
    IF :NEW.data_contratacao > SYSDATE THEN
        PKG_LOG.ERRO('Data de contratação no futuro: '
                      TO_CHAR(:NEW.data_contratacao, 'DD/MM/YYYY'), 'DOCENTE');
        RAISE E_DADOS_INVALIDOS;
    END IF;

    -- 3. Validar Identidade e Contactos
    IF :NEW.telemovel IS NULL OR NOT
        PKG_VALIDACAO.FUN_VALIDAR_TELEMOVEL(:NEW.telemovel) THEN
            PKG_LOG.ERRO('Telemóvel inválido ou ausente para docente: '
                          NVL(:NEW.telemovel, 'NULL'), 'DOCENTE');
            RAISE E_DADOS_INVALIDOS;
    END IF;

    -- NIF
    IF NOT PKG_VALIDACAO.FUN_VALIDAR_NIF(:NEW.nif) THEN
        PKG_LOG.ERRO('NIF inválido para docente: ' || :NEW.nif, 'DOCENTE');
        RAISE E_DADOS_INVALIDOS;
    END IF;

    -- CC
    IF :NEW.cc IS NOT NULL AND NOT PKG_VALIDACAO.FUN_VALIDAR_CC(:NEW.cc) THEN
        PKG_LOG.ERRO('CC inválido para docente: ' || :NEW.cc, 'DOCENTE');
        RAISE E_DADOS_INVALIDOS;
    END IF;

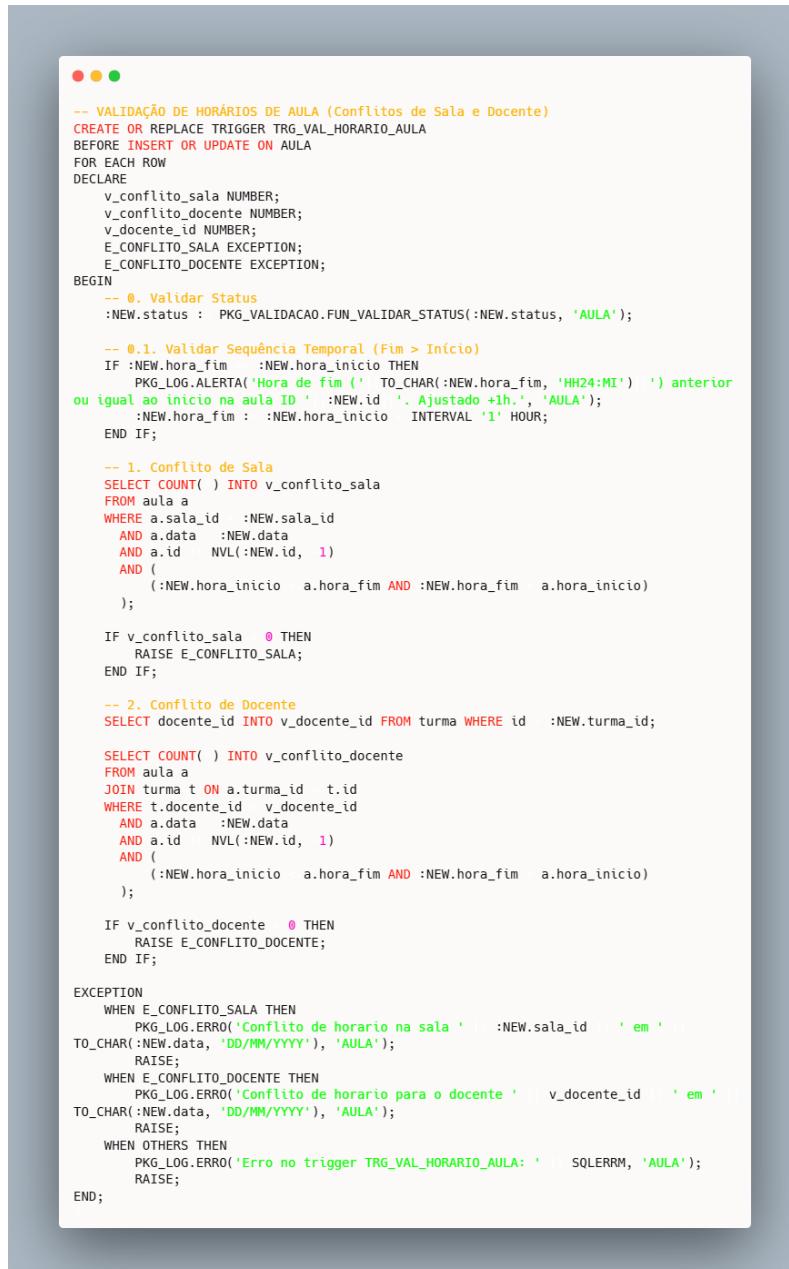
    -- Email
    IF :NEW.email IS NOT NULL AND NOT PKG_VALIDACAO.FUN_VALIDAR_EMAIL(:NEW.email) THEN
        PKG_LOG.ERRO('Email inválido: ' || :NEW.email, 'DOCENTE');
        RAISE E_DADOS_INVALIDOS;
    END IF;

    -- IBAN
    IF :NEW.iban IS NOT NULL AND NOT PKG_VALIDACAO.FUN_VALIDAR_IBAN(:NEW.iban) THEN
        PKG_LOG.ERRO('IBAN inválido para docente: ' || :NEW.iban, 'DOCENTE');
        RAISE E_DADOS_INVALIDOS;
    END IF;
EXCEPTION
    WHEN E_DADOS_INVALIDOS THEN RAISE;
    WHEN OTHERS THEN
        PKG_LOG.ERRO('Erro inesperado na validação do docente: ' || SQLERRM,
                      'DOCENTE');
        RAISE;
END;
```

Figura 242 - TRIGGER Valid. de Dados dos Docentes.

Gestão de Horários e Inscrições em Unidades Curriculares

Este próximo bloco de código contém a lógica de deteção de conflitos nos horários, isto porque impede a sobreposição de aulas na mesma sala ou com o mesmo docente. De seguida, o sistema de inscrição utiliza *buffers* para validar regras complexas, isto é, impede a duplicação de inscrições na mesma unidade curricular e ainda calcula o somatório de ECTS do aluno, acabando por bloquear a inscrição se o limite anual for ultrapassado.



```

-- VALIDAÇÃO DE HORÁRIOS DE AULA (Conflitos de Sala e Docente)
CREATE OR REPLACE TRIGGER TRG_VAL_HORARIO_AULA
BEFORE INSERT OR UPDATE ON AULA
FOR EACH ROW
DECLARE
    v_conflito_sala NUMBER;
    v_conflito_docente NUMBER;
    v_docente_id NUMBER;
    E_CONFLITO_SALA EXCEPTION;
    E_CONFLITO_DOCENTE EXCEPTION;
BEGIN
    -- 0. Validar Status
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'AULA');

    -- 0.1. Validar Sequência Temporal (Fim > Início)
    IF :NEW.hora_fim < :NEW.hora_inicio THEN
        PKG_LOG.ALERTA('Hora de fim (' || TO_CHAR(:NEW.hora_fim, 'HH24:MI') || ') anterior
ou igual ao inicio na aula ID ' || :NEW.id || '. Ajustado +1h.', 'AULA');
        :NEW.hora_fim := :NEW.hora_inicio INTERVAL '1' HOUR;
    END IF;

    -- 1. Conflito de Sala
    SELECT COUNT( ) INTO v_conflito_sala
    FROM aula a
    WHERE a.sala_id = :NEW.sala_id
        AND a.data = :NEW.data
        AND a.id = NVL(:NEW.id, 1)
        AND (
            (:NEW.hora_inicio < a.hora_fim AND :NEW.hora_fim < a.hora_inicio)
        );
    IF v_conflito_sala > 0 THEN
        RAISE E_CONFLITO_SALA;
    END IF;

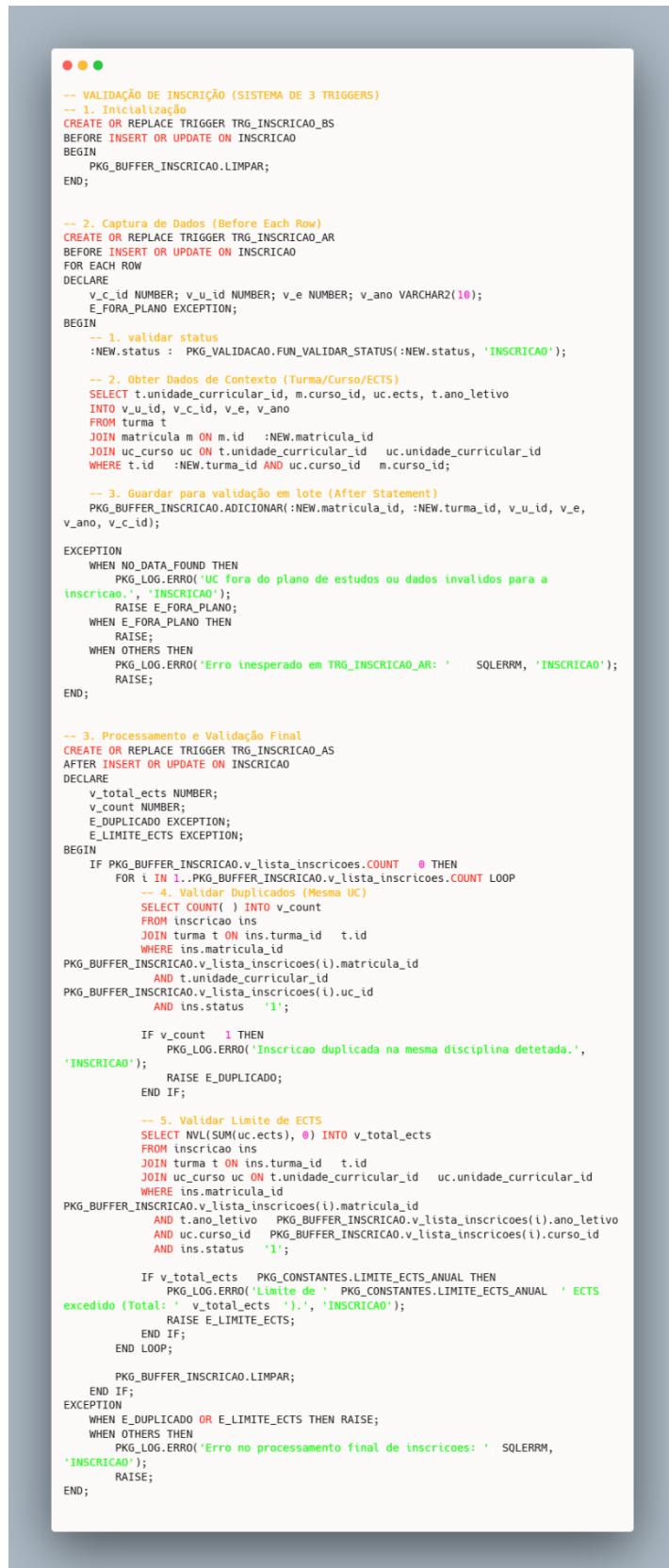
    -- 2. Conflito de Docente
    SELECT docente_id INTO v_docente_id
    FROM turma t
    JOIN aula a ON a.turma_id = t.id
    WHERE t.docente_id = v_docente_id
        AND a.data = :NEW.data
        AND a.id = NVL(:NEW.id, 1)
        AND (
            (:NEW.hora_inicio < a.hora_fim AND :NEW.hora_fim < a.hora_inicio)
        );
    IF v_conflito_docente > 0 THEN
        RAISE E_CONFLITO_DOCENTE;
    END IF;

    EXCEPTION
        WHEN E_CONFLITO_SALA THEN
            PKG_LOG.ERRO('Conflito de horario na sala ' || :NEW.sala_id || ' em '
TO_CHAR(:NEW.data, 'DD/MM/YYYY'), 'AULA');
            RAISE;
        WHEN E_CONFLITO_DOCENTE THEN
            PKG_LOG.ERRO('Conflito de horario para o docente ' || v_docente_id || ' em '
TO_CHAR(:NEW.data, 'DD/MM/YYYY'), 'AULA');
            RAISE;
        WHEN OTHERS THEN
            PKG_LOG.ERRO('Erro no trigger TRG_VAL_HORARIO_AULA: ' || SQLERRM, 'AULA');
            RAISE;
    END;
END;

```

Figura 243 - TRIGGER Validação de Horários.

Instituto Politécnico de Coimbra
Instituto Superior de Engenharia de Coimbra



```

-- VALIDAÇÃO DE INSCRIÇÃO (SISTEMA DE 3 TRIGGERS)
-- 1. Inicialização
CREATE OR REPLACE TRIGGER TRG_INSCRIÇÃO_BS
BEFORE INSERT OR UPDATE ON INSCRIÇÃO
BEGIN
    PKG_BUFFER_INSCRIÇÃO.LIMPAR;
END;

-- 2. Captura de Dados (Before Each Row)
CREATE OR REPLACE TRIGGER TRG_INSCRIÇÃO_AR
BEFORE INSERT OR UPDATE ON INSCRIÇÃO
FOR EACH ROW
DECLARE
    v_c_id NUMBER; v_u_id NUMBER; v_e NUMBER; v_ano VARCHAR2(10);
    E_FORA_PLANO EXCEPTION;
BEGIN
    -- 1. validar status
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'INSCRIÇÃO');

    -- 2. Obter Dados de Contexto (Turma/Curso/ECTS)
    SELECT t.unidade_curricular_id, m.curso_id, uc.ects, t.ano_letivo
    INTO v_c_id, v_u_id, v_e, v_ano
    FROM turma t
    JOIN matricula m ON m.id = :NEW.matricula_id
    JOIN uc_curso uc ON t.unidade_curricular_id = uc.unidade_curricular_id
    WHERE t.id = :NEW.turma_id AND uc.curso_id = m.curso_id;

    -- 3. Guardar para validação em lote (After Statement)
    PKG_BUFFER_INSCRIÇÃO.ADICIONAR(:NEW.matricula_id, :NEW.turma_id, v_u_id, v_e,
    v_ano, v_c_id);

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        PKG_LOG.ERRO('UC fora do plano de estudos ou dados invalidos para a
inscrição.', 'INSCRIÇÃO');
        RAISE E_FORA_PLANO;
    WHEN E_FORA_PLANO THEN
        RAISE;
    WHEN OTHERS THEN
        PKG_LOG.ERRO('Erro inesperado em TRG_INSCRIÇÃO_AR: ' || SQLERRM, 'INSCRIÇÃO');
        RAISE;
END;

-- 3. Processamento e Validação Final
CREATE OR REPLACE TRIGGER TRG_INSCRIÇÃO_AS
AFTER INSERT OR UPDATE ON INSCRIÇÃO
DECLARE
    v_total_ects NUMBER;
    v_count NUMBER;
    E_DUPPLICADO EXCEPTION;
    E_LIMITE_ECTS EXCEPTION;
BEGIN
    IF PKG_BUFFER_INSCRIÇÃO.v_lista_inscrições.COUNT > 0 THEN
        FOR i IN 1..PKG_BUFFER_INSCRIÇÃO.v_lista_inscrições.COUNT LOOP
            -- 4. Validar Duplicados (Mesma UC)
            SELECT COUNT() INTO v_count
            FROM inscrição ins
            JOIN turma t ON ins.turma_id = t.id
            WHERE ins.matricula_id
            PKG_BUFFER_INSCRIÇÃO.v_lista_inscrições(i).matricula_id
            AND t.unidade_curricular_id
            PKG_BUFFER_INSCRIÇÃO.v_lista_inscrições(i).uc_id
            AND ins.status = 'I';

            IF v_count > 1 THEN
                PKG_LOG.ERRO('Inscrição duplicada na mesma disciplina detetada.',
'INSCRIÇÃO');
                RAISE E_DUPPLICADO;
            END IF;

            -- 5. Validar Limite de ECTS
            SELECT NVL(SUM(uc.ects), 0) INTO v_total_ects
            FROM inscrição ins
            JOIN turma t ON ins.turma_id = t.id
            JOIN uc_curso uc ON t.unidade_curricular_id = uc.unidade_curricular_id
            WHERE ins.matricula_id
            PKG_BUFFER_INSCRIÇÃO.v_lista_inscrições(i).matricula_id
            AND t.ano_letivo = PKG_BUFFER_INSCRIÇÃO.v_lista_inscrições(i).ano_letivo
            AND uc.curso_id = PKG_BUFFER_INSCRIÇÃO.v_lista_inscrições(i).curso_id
            AND ins.status = 'I';

            IF v_total_ects > PKG_CONSTANTES.LIMITE_ECTS_ANUAL THEN
                PKG_LOG.ERRO('Limite de ' || PKG_CONSTANTES.LIMITE_ECTS_ANUAL || ' ECTS
excedido (Total: ' || v_total_ects || ').', 'INSCRIÇÃO');
                RAISE E_LIMITE_ECTS;
            END IF;
        END LOOP;
        PKG_BUFFER_INSCRIÇÃO.LIMPAR;
    END IF;
EXCEPTION
    WHEN E_DUPPLICADO OR E_LIMITE_ECTS THEN RAISE;
    WHEN OTHERS THEN
        PKG_LOG.ERRO('Erro no processamento final de inscrições: ' || SQLERRM,
'INSCRIÇÃO');
        RAISE;
END;

```

Figura 244 - TRIGGER Valid. Inscrição em U. Curricular.

Cálculo de Médias e Validação de Matrícula

O sistema de cálculo da média final de curso faz a atualização da ficha do aluno sempre que uma nota é fechada. Mais abaixo, acontece a validação da matrícula também, para assim impedir que o estudante tenha múltiplas matrículas ativas no mesmo curso, faz também a validação do plano de parcelas de propina.

Por fim, o *TRIGGER* invoca automaticamente a tesouraria para gerar as propinas assim que uma matrícula é criada.

```
-- ATUALIZAÇÃO AUTOMÁTICA DA MÉDIA GERAL (SISTEMA DE 3 TRIGGERS)
-- Trigger 1: Inicialização (Before Statement)
CREATE OR REPLACE TRIGGER TRG_MEDIA_MAT_BS
BEFORE UPDATE OF nota_final ON INSCRICAO
BEGIN
    PKG_BUFFER_MATRICULA.LIMPAR;
END;

-- Trigger 2: Captura (After Row)
CREATE OR REPLACE TRIGGER TRG_MEDIA_MAT_AR
AFTER UPDATE OF nota_final ON INSCRICAO
FOR EACH ROW
BEGIN
    -- Se a nota mudou, marca a matrícula para recálculo
    IF :NEW.nota_final IS NOT NULL AND (:OLD.nota_final IS NULL OR :NEW.nota_final <> :OLD.nota_final) THEN
        PKG_BUFFER_MATRICULA.ADICIONAR(:NEW.matricula_id);
    END IF;
END;

-- Trigger 3: Processamento (After Statement)
CREATE OR REPLACE TRIGGER TRG_MEDIA_MAT_AS
AFTER UPDATE OF nota_final ON INSCRICAO
DECLARE
    v_media_ponderada NUMBER;
    v_total_ects NUMBER;
    v_mat_id NUMBER;
    v_idx NUMBER;
BEGIN
    IF PKG_BUFFER_MATRICULA.v_ids_matricula.COUNT > 0 THEN
        v_idx := PKG_BUFFER_MATRICULA.v_ids_matricula.FIRST;
        LOOP
            EXIT WHEN v_idx IS NULL;
            v_mat_id := PKG_BUFFER_MATRICULA.v_ids_matricula(v_idx);

            -- Calcular média
            SELECT NVL(SUM(i.nota_final * uc.ects), 0), NVL(SUM(uc.ects), 0)
            INTO v_media_ponderada, v_total_ects
            FROM inscricao i
            JOIN turma t ON i.turma_id = t.id
            JOIN uc_curso uc ON t.unidade_curricular_id = uc.unidade_curricular_id
            WHERE i.matricula_id = v_mat_id
            AND i.nota_final = PKG_CONSTANTES.NOTA_APROVACAO
            AND i.status = '1';

            IF v_total_ects > 0 THEN
                UPDATE matricula
                SET media_geral = ROUND(v_media_ponderada / v_total_ects, 2)
                WHERE id = v_mat_id;
            END IF;

            v_idx := PKG_BUFFER_MATRICULA.v_ids_matricula.NEXT(v_idx);
        END LOOP;
        PKG_BUFFER_MATRICULA.LIMPAR;
    END IF;
    EXCEPTION WHEN OTHERS THEN
        PKG_LOG.ERRO('Erro no cálculo de média: ' || SQLERRM, 'MATRÍCULA');
    END;
END;
```

Figura 245 - TRIGGER Atualização Automática da Média Geral.



```

-- VALIDAÇÃO DE MATRÍCULA (DUPLICIDADE, PARCELAS E STATUS)
CREATE OR REPLACE TRIGGER TRG_VAL_MATRICULA
BEFORE INSERT OR UPDATE ON MATRICULA
FOR EACH ROW
DECLARE
    E_DUPPLICADO EXCEPTION;
    E_PARCELAS_INVALIDAS EXCEPTION;
    v_count NUMBER;
BEGIN
    -- 1. Validar Status
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'MATRICULA');

    -- 2. Validar Número de Parcelas (Min 1, Max 12)
    IF :NEW.numero_parcelas < PKG_CONSTANTES.MIN_PARCELAS OR :NEW.numero_parcelas > PKG_CONSTANTES.MAX_PARCELAS THEN
        PKG_LOG.ERRO('Número de parcelas invalido (' || :NEW.numero_parcelas || '). Deve ser entre ' ||
                     PKG_CONSTANTES.MIN_PARCELAS || ' e ' || PKG_CONSTANTES.MAX_PARCELAS ||
                     ', ' || 'MATRICULA');
        RAISE E_PARCELAS_INVALIDAS;
    END IF;

    -- 3. Impedir Matrícula Ativa duplicada no mesmo Curso
    IF (:NEW.status = '1' AND :NEW.estado_matricula = 'Ativa') AND
        (:INSERTING OR (:OLD.estado_matricula = 'Ativa') OR (:OLD.status = '1')) THEN
        SELECT COUNT( ) INTO v_count
        FROM matricula
        WHERE estudiante_id = :NEW.estudiante_id
            AND curso_id = :NEW.curso_id
            AND estado_matricula = 'Ativa'
            AND status = '1'
            AND id = NVL(:NEW.id, 1);

        IF v_count = 0 THEN
            PKG_LOG.ERRO('Aluno ja tem uma matricula ativa neste curso.', 'MATRICULA');
            RAISE E_DUPPLICADO;
        END IF;
    END IF;

    EXCEPTION
        WHEN E_PARCELAS_INVALIDAS THEN RAISE;
        WHEN E_DUPPLICADO THEN RAISE;
        WHEN OTHERS THEN
            PKG_LOG.ERRO('Erro na validacao de matricula: ' || SQLERRM, 'MATRICULA');
            RAISE;
    END;

    -- GERAR PROPINAS AUTOMATICAMENTE AO MATRICULAR
    CREATE OR REPLACE TRIGGER TRG_AUTO_GERAR_PROPINAS
    AFTER INSERT ON MATRICULA
    FOR EACH ROW
    DECLARE
        v_valor_total NUMBER;
    BEGIN
        SELECT tc.valor_propinas INTO v_valor_total
        FROM curso c
        JOIN tipo_curso tc ON c.tipo_curso_id = tc.id
        WHERE c.id = :NEW.curso_id;

        PKG_TESOURARIA.PRC_GERAR_PLANO_PAGAMENTO(:NEW.id, v_valor_total,
        :NEW.numero_parcelas);
        EXCEPTION WHEN OTHERS THEN
            PKG_LOG.ERRO('Erro ao gerar propinas para matricula ' || :NEW.id || ': ' || SQLERRM, 'PARCELA_PROPINA');
    END;

```

Figura 246 - TRIGGERS Valid. de Matrícula e Gerar Propinas pós Matrícula.

Validações de Integridade Diversas e Segurança

Este é talvez o trecho mais longo e trabalhoso de *TRIGGERS* pois, aplica regras específicas a várias tabelas de suporte. Valida capacidades de salas, duração de cursos, tamanhos de ficheiros de entrega e recurso, e integridade de pagamentos. Destaca-se também o *TRIGGER* de segurança, que visa proteger a tabela "log" contra possíveis alterações ou apagamentos manuais, garantindo que os dados se mantêm imutáveis.



```
-- VALIDAÇÃO DE SALA (STATUS E CAPACIDADE)
CREATE OR REPLACE TRIGGER TRG_VAL_SALA
BEFORE INSERT OR UPDATE ON SALA
FOR EACH ROW
BEGIN
    -- 1. Validar Status (0 ou 1)
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'SALA');

    -- 2. Garantir Capacidade Positiva
    IF :NEW.capacidade = 0 OR :NEW.capacidade IS NULL THEN
        PKG_LOG.ERRO('Capacidade invalida (' || NVL(TO_CHAR(:NEW.capacidade), 'NULL') || ') na sala ' || :NEW.nome || '. Forcada a 1.', 'SALA');
        :NEW.capacidade := 1;
    END IF;
END;

-- VALIDAÇÃO DE CURSO (STATUS, DURACAO, ECTS)
CREATE OR REPLACE TRIGGER TRG_VAL_CURSO
BEFORE INSERT OR UPDATE ON CURSO
FOR EACH ROW
BEGIN
    -- 1. Validar Status
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'CURSO');

    -- 2. Validar Duração (Positiva)
    IF :NEW.duracao = 0 OR :NEW.duracao IS NULL THEN
        PKG_LOG.ERRO('Duracao invalida (' || NVL(TO_CHAR(:NEW.duracao), 'NULL') || ') no curso ' || :NEW.nome || '. Ajustada para 1 ano.', 'CURSO');
        :NEW.duracao := 1;
    END IF;

    -- 3. Validar ECTS (Positivos)
    IF :NEW.ects = 0 OR :NEW.ects IS NULL THEN
        PKG_LOG.ERRO('ECTS invalidos (' || NVL(TO_CHAR(:NEW.ects), 'NULL') || ') no curso ' || :NEW.nome || '. Ajustado para 0.', 'CURSO');
        :NEW.ects := 0;
    END IF;
END;
```

Figura 247 - TRIGGERS Validação Salas e Cursos.



```
● ● ●

-- VALIDAÇÃO DE FICHEIRO DE ENTREGA (STATUS E TAMANHO)
CREATE OR REPLACE TRIGGER TRG_VAL_FICHEIRO_ENTREGA
BEFORE INSERT OR UPDATE ON FICHEIRO_ENTREGA
FOR EACH ROW
DECLARE
    E_TAMANHO_INVALIDO EXCEPTION;
BEGIN
    -- 1. Validar Status
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'FICHEIRO_ENTREGA');

    -- 2. Validar Tamanho
    IF NOT PKG_VALIDACAO.FUN_VALIDAR_TAMANHO_FICHEIRO(:NEW.tamanho) THEN
        PKG_LOG.ERRO('Tamanho de ficheiro invalido: ' || NVL(TO_CHAR(:NEW.tamanho),
        'NULL'))
        ' (Max: ' || PKG_CONSTANTES.TAMANHO_MAX_FICHEIRO || ')';
        RAISE E_TAMANHO_INVALIDO;
    END IF;
EXCEPTION
    WHEN E_TAMANHO_INVALIDO THEN RAISE;
    WHEN OTHERS THEN
        PKG_LOG.ERRO('Erro na validacao de ficheiro: ' || SQLERRM, 'FICHEIRO_ENTREGA');
        RAISE;
END;

-- VALIDAÇÃO DE FICHEIRO DE RECURSO (STATUS E TAMANHO) TABELA TEM QUE SER ALTERADA
CREATE OR REPLACE TRIGGER TRG_VAL_FICHEIRO_RECURSO
BEFORE INSERT OR UPDATE ON FICHEIRO_RECURSO
FOR EACH ROW
DECLARE
    E_TAMANHO_INVALIDO EXCEPTION;
BEGIN
    -- 1. Validar Status
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'FICHEIRO_RECURSO');

    -- 2. Validar Tamanho (obtendo o tamanho do BLOB)
    IF NOT PKG_VALIDACAO.FUN_VALIDAR_TAMANHO_FICHEIRO(:NEW.tamanho) THEN
        PKG_LOG.ERRO('Tamanho de ficheiro de recurso invalido: ' ||
        NVL(TO_CHAR(:NEW.tamanho), 'NULL'))
        ' (Max: ' || PKG_CONSTANTES.TAMANHO_MAX_FICHEIRO || ')';
        RAISE E_TAMANHO_INVALIDO;
    END IF;
EXCEPTION
    WHEN E_TAMANHO_INVALIDO THEN RAISE;
    WHEN OTHERS THEN
        PKG_LOG.ERRO('Erro na validacao de ficheiro de recurso: ' || SQLERRM,
        'FICHEIRO_RECURSO');
        RAISE;
END;
```

Figura 248 - TRIGGERS Valid. de Fich. de Entrega e Fich. de Recurso.



```

-- VALIDAÇÃO DE PARCELA DE PROPINA
CREATE OR REPLACE TRIGGER TRG_VAL_PARCELA_PROPINA
BEFORE INSERT OR UPDATE ON PARCELA_PROPINA
FOR EACH ROW
DECLARE
    v_total_curso NUMBER;
    v_num_parcelas NUMBER;
    v_mat_id NUMBER;
    E_DADOS_INVALIDOS EXCEPTION;
BEGIN
    -- 1. Validar Status
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.estado, 'PARCELA_PROPINA');

    -- 2. Validar Valor Positivo
    IF :NEW.valor < 0 THEN
        PKG_LOG.ERRO('Valor da parcela deve ser maior que 0.', 'PARCELA_PROPINA');
        RAISE E_DADOS_INVALIDOS;
    END IF;

    -- 3. Validar Data de Vencimento (Futura no Registo)
    IF INSERTING THEN
        IF :NEW.data_vencimento <= TRUNC(SYSDATE) THEN
            PKG_LOG.ERRO('Data de vencimento deve ser futura (''DD/MM/YYYY'').', 'PARCELA_PROPINA');
            RAISE E_DADOS_INVALIDOS;
        END IF;
    END IF;

    -- 4. Consistência de Pagamento
    IF :NEW.estado = '0' THEN
        :NEW.data_pagamento := NULL;
    ELSIF :NEW.estado = '1' AND :NEW.data_pagamento IS NULL THEN
        :NEW.data_pagamento := SYSDATE;
    END IF;

    EXCEPTION
        WHEN E_DADOS_INVALIDOS THEN RAISE;
        WHEN OTHERS THEN
            PKG_LOG.ERRO('Erro na validacao de parcela: '||SQLERRM, 'PARCELA_PROPINA');
            RAISE;
    END;

-- VALIDAÇÃO DE PRESENÇA (Integridade Académica)
CREATE OR REPLACE TRIGGER TRG_VAL_PRESENCA
BEFORE INSERT OR UPDATE ON PRESENCA
FOR EACH ROW
DECLARE
    E_DADOS_INVALIDOS EXCEPTION;
BEGIN
    -- 1. Validar Status
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.estado, 'PRESENCA');

    -- 2. Validar valor do campo presente ('0' ou '1')
    IF :NEW.presente NOT IN ('0', '1') THEN
        :NEW.presente := '0';
    END IF;

    EXCEPTION
        WHEN OTHERS THEN
            PKG_LOG.ERRO('Erro na validacao de presenca: '||SQLERRM, 'PRESENCA');
            RAISE;
    END;

```

Figura 249 - TRIGGERS Valid. de Parc. de Propinas e Presenças.



```

--PROTEÇÃO DA TABELA DE LOGS (IMUTABILIDADE COM RASTRO)
CREATE OR REPLACE TRIGGER TRG_PROTEGER_LOG
BEFORE DELETE OR UPDATE ON LOG
FOR EACH ROW
DECLARE
    E_IMUTAVEL EXCEPTION;
    v_operacao VARCHAR2(20);
BEGIN
    v_operacao : CASE WHEN DELETING THEN 'DELETE' ELSE 'UPDATE' END;
    PKG_LOG.REGISTAR('VIOLACAO_SEGURANCA',
                      'Tentativa de '||v_operacao||' no log id: '||:OLD.id,
                      'LOG');
    RAISE E_IMUTAVEL;
END;

```

Figura 250 - TRIGGER Imutabilidade de Logs.

```

-- VALIDAÇÃO DE RECURSO
CREATE OR REPLACE TRIGGER TRG_VAL_RECURSO
BEFORE INSERT OR UPDATE ON RECURSO
FOR EACH ROW
DECLARE
    v_turma_docente NUMBER;
    E_DOCENTE_NAO_TURMA EXCEPTION;
BEGIN
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'RECURSO');

    -- Verifica se o docente é o responsável pela turma
    SELECT docente_id INTO v_turma_docente FROM turma WHERE id = :NEW.turma_id;

    IF :NEW.docente_id = v_turma_docente THEN
        RAISE E_DOCENTE_NAO_TURMA;
    END IF;
EXCEPTION
    WHEN E_DOCENTE_NAO_TURMA THEN
        PKG_LOG.ERRO('Docente '||:NEW.docente_id||' não pertence a turma
                      :NEW.turma_id, 'RECURSO');
        RAISE;
END;

-- VALIDAÇÃO DE TIPO_AULA E TIPO_AVALIACAO
CREATE OR REPLACE TRIGGER TRG_VAL_TIPO_AULA
BEFORE INSERT OR UPDATE ON TIPO_AULA
FOR EACH ROW
BEGIN
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'TIPO_AULA');
END;

CREATE OR REPLACE TRIGGER TRG_VAL_TIPO_AVALIACAO
BEFORE INSERT OR UPDATE ON TIPO_AVALIACAO
FOR EACH ROW
BEGIN
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'TIPO_AVALIACAO');

    IF :NEW.requer_entrega NOT IN ('0','1') THEN :NEW.requer_entrega := '0'; END IF;
    IF :NEW.permite_grupo NOT IN ('0','1') THEN :NEW.permite_grupo := '0'; END IF;
    IF :NEW.permite_filhos NOT IN ('0','1') THEN :NEW.permite_filhos := '0'; END IF;
END;

```

Figura 251 - TRIGGERS Validação de Recursos, Tipos de Aula e Tipos Avaliação.

```

● ● ●

-- VALIDAÇÃO DE TIPO_CURSO E UNIDADE_CURRICULAR
CREATE OR REPLACE TRIGGER TRG_VAL_TIPO_CURSO
BEFORE INSERT OR UPDATE ON TIPO_CURSO
FOR EACH ROW
BEGIN
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'TIPO_CURSO');
    IF :NEW.valor_propinas < 0 THEN
        PKG_LOG.ERRO('Valor de propinas negativo', 'TIPO_CURSO');
        :NEW.valor_propinas := 0;
    END IF;
END;

CREATE OR REPLACE TRIGGER TRG_VAL_UC
BEFORE INSERT OR UPDATE ON UNIDADE_CURRICULAR
FOR EACH ROW
BEGIN
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'UNIDADE_CURRICULAR');
    IF :NEW.horas_teoricas < 0 THEN :NEW.horas_teoricas := 0; END IF;
    IF :NEW.horas_praticas < 0 THEN :NEW.horas_praticas := 0; END IF;
END;

-- VALIDAÇÃO DE TURMA
CREATE OR REPLACE TRIGGER TRG_VAL_TURMA
BEFORE INSERT OR UPDATE ON TURMA
FOR EACH ROW
DECLARE
    v_exists NUMBER;
    E_DOCENTE_INVALIDO EXCEPTION;
BEGIN
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'TURMA');

    -- Verifica se o par (UC, Docente) existe na tabela de competências uc_docente
    SELECT COUNT( ) INTO v_exists
    FROM uc_docente
    WHERE unidade_curricular_id = :NEW.unidade_curricular_id
        AND docente_id = :NEW.docente_id;

    IF v_exists < 0 THEN
        RAISE E_DOCENTE_INVALIDO;
    END IF;

    IF :NEW.max_alunos < 1 THEN :NEW.max_alunos := 1; END IF;
EXCEPTION
    WHEN E_DOCENTE_INVALIDO THEN
        PKG_LOG.ERRO('O docente ' || :NEW.docente_id || ' não está habilitado para a UC
        :NEW.unidade_curricular_id, 'TURMA');
        RAISE;
END;

```

Figura 252 - TRIGGERS Valid. do Tipo de Curso, U. Curricular e Turma.

```

-- VALIDAÇÃO DE UC_CURSO
CREATE OR REPLACE TRIGGER TRG_VAL_UC_CURSO
BEFORE INSERT OR UPDATE ON UC_CURSO
FOR EACH ROW
DECLARE
    v_duracao_curso NUMBER;
    E_DURACAO_EXCEDIDA EXCEPTION;
    E_PRESENCA_INVALIDA EXCEPTION;
BEGIN
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'UC_CURSO');

    -- Validar duração
    SELECT duracao INTO v_duracao_curso FROM curso WHERE id = :NEW.curso_id;
    IF :NEW.ano < v_duracao_curso THEN
        RAISE E_DURACAO_EXCEDIDA;
    END IF;

    -- Validar regras de presença
    IF :NEW.presenca_obrigatoria NOT IN ('0','1') THEN :NEW.presenca_obrigatoria := '0';
    END IF;

    IF :NEW.presenca_obrigatoria = '1' AND :NEW.percentagem_presenca IS NULL THEN
        :NEW.percentagem_presenca := PKG_CONSTANTES.PERCENTAGEM_PRESENCA_DEFAULT;
        PKG_LOG.ERRO('Percentagem de presença não definida. Aplicado default: ' ||
                      :NEW.percentagem_presenca, 'UC_CURSO');
    END IF;

    IF :NEW.presenca_obrigatoria = '0' THEN
        :NEW.percentagem_presenca := NULL;
    END IF;

EXCEPTION
    WHEN E_DURACAO_EXCEDIDA THEN
        PKG_LOG.ERRO('Ano ' || :NEW.ano || ' superior a duração do curso
        (' || v_duracao_curso || ')', 'UC_CURSO');
        RAISE;
    WHEN E_PRESENCA_INVALIDA THEN
        PKG_LOG.ERRO('Percentagem de presença obrigatória não definida', 'UC_CURSO');
        RAISE;
END;

-- VALIDAÇÃO DE UC_DOCENTE
CREATE OR REPLACE TRIGGER TRG_VAL_UC_DOCENTE
BEFORE INSERT OR UPDATE ON UC_DOCENTE
FOR EACH ROW
BEGIN
    :NEW.status := PKG_VALIDACAO.FUN_VALIDAR_STATUS(:NEW.status, 'UC_DOCENTE');
END;

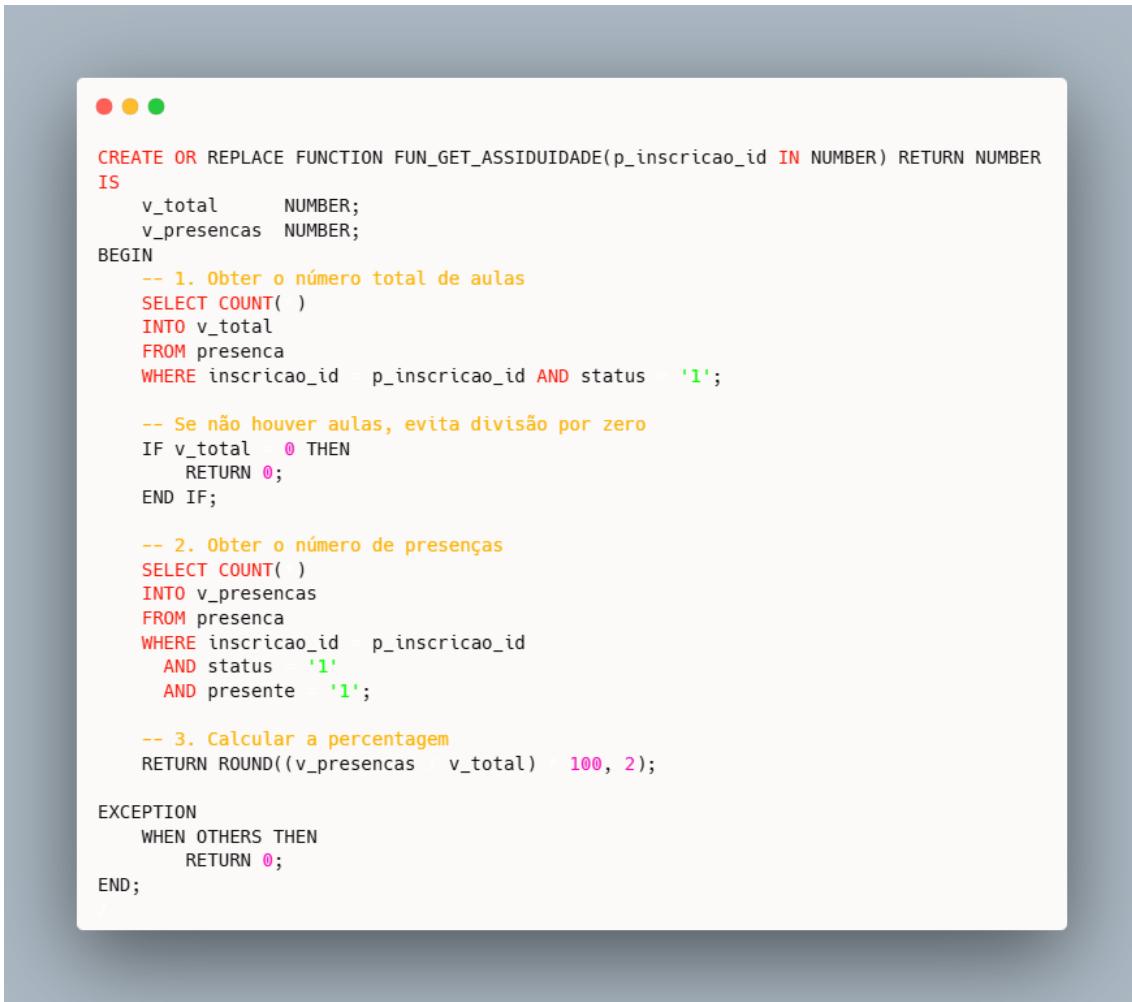
```

Figura 253 - TRIGGERS Valid. de UC_Curso e UC_Docente.



Report de Vistas e Funções de Auxílio

Por fim, o *script* encerra com a criação de objetos de leitura. As próximas linhas de código definem a função de cálculo de assiduidade, enquanto que as restantes criam vistas que consolidam dados para relatórios, incluindo pautas de turma, alertas de reprovação por faltas, perfis académicos, ocupação de salas, relatórios de dívidas e estatísticas de ocupação de cursos.



```
CREATE OR REPLACE FUNCTION FUN_GET_ASSIDUIDADE(p_inscricao_id IN NUMBER) RETURN NUMBER
IS
    v_total      NUMBER;
    v_presencas  NUMBER;
BEGIN
    -- 1. Obter o número total de aulas
    SELECT COUNT( )
    INTO v_total
    FROM presenca
    WHERE inscricao_id = p_inscricao_id AND status = '1';

    -- Se não houver aulas, evita divisão por zero
    IF v_total = 0 THEN
        RETURN 0;
    END IF;

    -- 2. Obter o número de presenças
    SELECT COUNT( )
    INTO v_presencas
    FROM presenca
    WHERE inscricao_id = p_inscricao_id
        AND status = '1'
        AND presente = '1';

    -- 3. Calcular a percentagem
    RETURN ROUND((v_presencas / v_total) * 100, 2);

EXCEPTION
    WHEN OTHERS THEN
        RETURN 0;
END;
```

Figura 254 - FUNCTION de Assiduidade.



```
-- PAUTA POR TURMA
-- Mostra notas finais e o estado de aprovação
CREATE OR REPLACE VIEW VW_PAUTA_TURMA AS
SELECT
    t.nome as TURMA,
    uc.nome as UNIDADE_CURRICULAR,
    e.codigo as CODIGO_ALUNO,
    e.nome as NOME_ESTUDANTE,
    i.nota_final as NOTA_FINAL,
    CASE
        WHEN i.nota_final     PKG_CONSTANTES.NOTA_APROVACAO THEN 'APROVADO'
        ELSE 'REPROVADO'
    END as RESULTADO
FROM inscricao i
JOIN turma t ON i.turma_id = t.id
JOIN unidade_curricular uc ON t.unidade_curricular_id = uc.id
JOIN matricula m ON i.matricula_id = m.id
JOIN estudante e ON m.estudante_id = e.id
WHERE i.status = '1';

-- ALERTAS DE ASSIDUIDADE
-- Identifica alunos em risco de reprovação por faltas.
CREATE OR REPLACE VIEW VW_ALERTA_ASSIDUIDADE AS
SELECT
    e.nome as NOME_ESTUDANTE,
    t.nome as NOME_TURMA,
    (100 - FUN_GET_ASSIDUIDADE(i.id)) as PERC_FALTAS
FROM inscricao i
JOIN matricula m ON i.matricula_id = m.id
JOIN estudante e ON m.estudante_id = e.id
JOIN turma t ON i.turma_id = t.id
WHERE i.status = '1'
    AND (100 - FUN_GET_ASSIDUIDADE(i.id)) < 25;
```

Figura 255 - VIEWS Pautas por Turma e Assiduidade.

```

● ● ●

-- PERFIL ACADÉMICO DO ALUNO
-- Resumo de ECTS conquistados e média global do curso.
CREATE OR REPLACE VIEW VW_PERFIL_ACADEMICO_ALUNO AS
SELECT
    e.codigo AS num_mecanografico,
    e.nome AS nome_estudante,
    c.nome AS nome_curso,
    COUNT(i.id) AS total_UCS_inscritas,
    SUM(CASE WHEN i.nota_final      PKG_CONSTANTES.NOTA_APROVACAO THEN ucc.ects ELSE 0
END) AS ects_concluidos,
    ROUND(AVG(i.nota_final), 2) AS media_global
FROM estudante e
JOIN matricula m ON e.id   m.estudante_id
JOIN curso c ON m.curso_id  c.id
LEFT JOIN inscricao i ON m.id   i.matricula_id
LEFT JOIN turma t ON i.turma_id  t.id
LEFT JOIN unidade_curricular uc ON t.unidade_curricular_id  uc.id
LEFT JOIN uc_curso ucc ON uc.id   ucc.unidade_curricular_id AND c.id   ucc.curso_id
WHERE m.status  '1'
GROUP BY e.codigo, e.nome, c.nome;

-- OCUPAÇÃO DE SALAS (HOJE)
CREATE OR REPLACE VIEW VW_OCUPACAO_SALAS_HOJE AS
SELECT
    s.nome AS sala,
    TO_CHAR(a.hora_inicio, 'HH24:MI') AS inicio,
    TO_CHAR(a.hora_fim, 'HH24:MI') AS fim,
    uc.nome AS unidade_curricular,
    d.nome AS docente,
    t.nome AS turma
FROM sala s
JOIN aula a ON s.id   a.sala_id
JOIN turma t ON a.turma_id  t.id
JOIN unidade_curricular uc ON t.unidade_curricular_id  uc.id
JOIN docente d ON t.docente_id  d.id
WHERE TRUNC(a.data)  TRUNC(SYSDATE)
    AND a.status  '1';

-- CARGA HORÁRIA DOS DOCENTES
CREATE OR REPLACE VIEW VW_CARGA_HORARIA_DOCENTE AS
SELECT
    d.nome AS nome_docente,
    COUNT(DISTINCT t.id) AS total_turmas,
    SUM(ROUND((a.hora_fim - a.hora_inicio) / 24, 2)) AS total_horas_semanais
FROM docente d
JOIN turma t ON d.id   t.docente_id
JOIN aula a ON t.id   a.turma_id
WHERE d.status  '1'
GROUP BY d.nome;

```

Figura 256 - VIEWS Perfil Acad., Ocupação de Sala/Dia e Horário dos Docentes.



```

-- RELATÓRIO DE DÍVIDAS
-- Lista alunos com pagamentos em atraso e o valor total em falta.
CREATE OR REPLACE VIEW VW_RELATORIO_DIVIDAS AS
SELECT
    e.nome AS estudante,
    e.telemovel,
    c.nome AS curso,
    COUNT(p.id) AS parcelas_em_atraso,
    SUM(p.valor) AS valor_total_divida
FROM parcela_propina p
JOIN matricula m ON p.matricula_id = m.id
JOIN estudante e ON m.estudante_id = e.id
JOIN curso c ON m.curso_id = c.id
WHERE p.estado = '0' -- Não Pago
    AND p.data_vencimento = SYSDATE
    AND p.status = '1'
GROUP BY e.nome, e.telemovel, c.nome;

-- RECEITA PREVISTA VS REALIZADA POR CURSO
-- Análise financeira de desempenho por curso.
CREATE OR REPLACE VIEW VW_FINANCIERO_CURSOS AS
SELECT
    c.nome AS curso,
    SUM(CASE WHEN p.estado = '1' THEN p.valor ELSE 0 END) AS total_recebido,
    SUM(CASE WHEN p.estado = '0' THEN p.valor ELSE 0 END) AS total_pendente
FROM curso c
JOIN matricula m ON c.id = m.curso_id
JOIN parcela_propina p ON m.id = p.matricula_id
WHERE p.status = '1'
GROUP BY c.nome;

-- CALENDÁRIO DE AVALIAÇÕES
-- Para consulta dos alunos e planeamento de salas.
CREATE OR REPLACE VIEW VW_EXAMES_PROXIMOS AS
SELECT
    av.data AS data_exame,
    uc.nome AS unidade_curricular,
    av.titulo AS avaliacao,
    ta.nome AS tipo,
    t.nome AS turma
FROM avaliacao av
JOIN turma t ON av.turma_id = t.id
JOIN unidade_curricular uc ON t.unidade_curricular_id = uc.id
JOIN tipo_avaliacao ta ON av.tipo_avaliacao_id = ta.id
WHERE av.data BETWEEN SYSDATE AND SYSDATE + 30
    AND av.status = '1'
ORDER BY av.data ASC;

```

Figura 257 - VIEWS Rel. de Dívidas, Análise Financeira Curso e Calendário de Aval.

```
-- MONITORAÇÃO DE ENTREGAS DE GRUPO
-- Cruza quem entregou trabalhos e o tamanho dos ficheiros.
CREATE OR REPLACE VIEW VW_MONITOR_ENTREGAS AS
SELECT
    av.titulo AS avaliacao,
    en.id AS entrega_id,
    e.nome AS aluno,
    fe.nome AS ficheiro,
    ROUND(fe.tamanho  1024, 2) AS tamanho_mb,
    en.data_entrega AS data_submissao
FROM entrega en
JOIN avaliacao av ON en.avaliacao_id  av.id
JOIN estudante_entrega ee ON en.id  ee.entrega_id
JOIN inscricao i ON ee.inscricao_id  i.id
JOIN matricula m ON i.matricula_id  m.id
JOIN estudante e ON m.estudante_id  e.id
LEFT JOIN ficheiro_entrega fe ON en.id  fe.entrega_id
WHERE en.status  '1';

-- ESTATÍSTICA DE OCUPAÇÃO DE CURSOS
-- Verifica a taxa de preenchimento de vagas por curso.
CREATE OR REPLACE VIEW VW_ESTATISTICA_VAGAS AS
SELECT
    c.nome AS curso,
    c.max_alunos AS vagas_totais,
    COUNT(m.id) AS alunos_matriculados,
    c.max_alunos  COUNT(m.id) AS vagas_livres,
    CASE
        WHEN c.max_alunos IS NOT NULL AND c.max_alunos  0
            THEN ROUND((COUNT(m.id)  c.max_alunos)  100, 2)
        ELSE "Não existe limite de alunos"
    END AS taxa_preenchimento
FROM curso c
LEFT JOIN matricula m ON c.id  m.curso_id AND m.status  '1'
WHERE c.status  '1'
GROUP BY c.nome, c.max_alunos;
```

Figura 258 - VIEWS Monitorização Entregas em Grupo e Estatística Ocupação Cursos.



Nota: A ferramenta utilizada para servir de tela para o código visto ao longo do último tópico desenvolvido, chama-se de Carbon (<https://carbon.now.sh/>), contudo devido ao estilo utilizado, os “/” com que todos os scripts PL/SQL possuem no final do script, ficou com uma coloração branca que se confunde facilmente com o fundo da tela, no entanto eles estão lá.

Cálculo do Espaço Físico que a Base de Dados Ocupa em Produção

Chegando ao último capítulo do presente projeto, é facultado que seja feito o cálculo do espaço físico que a base de dados ocupa aquando da sua produção.

Para este efeito, as duas tabelas escolhidas foram a tabela “ficheiro_recurso” e a tabela “estudante_entrega”.

Parâmetros de Estimativa do Espaço Ocupado

Header Fixo: 84 bytes

Header Variável: 5 bytes por cada registo (2 bytes header do registo + 1 byte com o Nº de colunas) + (2 bytes no row directory)

PCTFREE: X (Entre 1 a 25)

PCTUSED: Y (Entre 40 a 95)

Tamanho do Bloco: 4096 bytes (por defeito)

Parâmetros a Calcular

Funções da Estimativa do Espaço Ocupado:

1. Tamanho Médio do Registo
2. Espaço Livre no Bloco
3. Nº de Registros por Bloco

Cálculo do Initial:

1. Nº de Blocos
2. Espaço Inicial da Tabela

Cálculo do Next:

1. Nº de Blocos (Blocos Previstos)
2. Espaço Next da Tabela

Cálculos da Tabela “estudante_entrega”:

1. Tamanho Médio do Registo: $4 + 4 + 11 + 11 + 1 + 5 + 5 = 41\text{bytes}$
2. Espaço Livre no Bloco: $4096 * (100 - 10) / 100 - 84 = 3602,4\text{ bytes}$
3. Número de Registros por Bloco: $3602,4 / 41 = 81,863 \approx 81$

Cálculo do Initial:

4. Número de Blocos: $600 / 81 = 7,407 \approx 8$
5. Espaço Inicial da Tabela: $8 * 4096 = 32768\text{ bytes}$

Cálculo do Next:

6. Número de Blocos Previstos: $600 / 81 = 7,407 \approx 8$
7. Espaço Next da Tabela: $8 * 4096 = 32768\text{ bytes}$

Cálculos da Tabela “ficheiro_recurso”:

1. Tamanho Médio do Registo: $4 + 4 + 100 + 40 + 20 + 20 + 11 + 11 + 1 + 5 + 9 = 225\text{ bytes}$
2. Espaço Livre no Bloco: $4096 * (100 - 10) / 100 - 84 = 3602,4\text{ bytes}$
3. Número de Registros por Bloco: $3602,4 / 225 = 16,010 \approx 16$

Cálculo do Initial:

4. Número de Blocos: $150 / 16 = 9,375 \approx 10$
5. Espaço Inicial da Tabela: $10 * 4096 = 40960\text{ bytes}$

Cálculo do Next:

6. Número de Blocos Previstos: $150 / 16 = 9,375 \approx 10$
7. Espaço Next da Tabela: $10 * 4096 = 40960\text{ bytes}$

Em relação aos cálculos da página 170, no cálculo da tabela “ficheiro_recurso” foi usado um ponteiro, isto porque o tamanho de um ficheiro BLOB é muito grande e iria deixar o N.R.B a 0 ou pelo menos arredondado a 0, e quando o N.R.B é o cálculo torna-se impossível, para contornar a situação foi o usado um ponteiro e os cálculos foram feitos em função do tamanho do ponteiro, que no caso tem um tamanho definido de 40 bytes.

Outra situação a ser notada é o facto de como é o primeiro ano de uso deste sistema (segundo o especificado nas mudanças efetuadas para a segunda fase do projeto), significa que o cálculo do *INITIAL* e do *NEXT* irá ser igual e com os mesmos valores.

Conclusão – Segunda Fase

A conclusão será mais a título pessoal do grupo, se bem que em certas situações ao longo do documento, fomos intervindo para justificar certas coisas relativas ao projeto, deixando a escrita formal de lado.

Foi bastante trabalhoso o desenvolvimento deste projeto, uma vez que foi um projeto enorme, sabíamos disso e sabíamos das consequências que isso poderia acarretar, ainda assim achamos que foi um projeto bem conseguido e nos dará arcaboiço para o futuro. Descobrimos e aprendemos coisas novas, que não se falaram nas aulas e isso alegra-nos, porque quer dizer que fomos audazes na busca por informação e quisemos melhorar/aperfeiçoar o que sabíamos ou não ainda não sabíamos.

Podemos dizer que independentemente da apreciação que este nosso projeto possa vir a receber, nos esforçamos, soubemos dividir bem as tarefas, sabemos o que foi feito e o que pode ter corrido menos bem e isso é certamente motivo de orgulho.

P.S. Este documento e consequente projeto dispensa a declaração de *webgrafia* ou *bibliografia*, visto que o presente documento, apenas explica a execução passo a passo do projeto.

Ademais é importante revelar, que este documento, foi submetido juntamente com os restantes ficheiros que complementam o projeto.

FIM

175

