4.Modelo físico

Neste capítulo é demonstrada a forma como partindo do modelo lógico apresentado no capítulo anterior, foi implementado o modelo físico.

## 4.1 Tradução do modelo lógico para um SGBD e consequente implementação

O primeiro passo para traduzir o modelo lógico num modelo físico foi escolher o SGBD(Sistema de Gestão e Base de Dados) que melhor se adequava às funcionalidades requeridas pelo sistema a implementar. Por uma questão de comodidade foi escolhido o MySQL, um sistema grátis e fácil de instalar, aceder e dar manutenção, que já tinha sido anteriormente introduzido durante as aulas da UC de Base de Dados. A linguagem do MySQL é de fácil compreensão, tornando-o por isso num sistema que permite implementações simples. Como é uma solução relativamente popular, existe toda uma comunidade que deu resposta a obstáculos que apareceram durante o processo de implementação. O MySQL também tem algumas limitações, mas dada a dimensão do problema, suporta bem as operações necessárias à gestão do mesmo. Questões de segurança e fortes mecanismos de proteção de dados foram também um dos motivos para a utilização do MySQL. Em termos de inserção de dados, este SGBD permite a criação de chaves primárias únicas, forçando a que os valores de índice sejam sempre distintos, e por isso leva a rapidez nas pesquisas e controlo de integridade. Permite também a especificação de chaves estrangeiras e domínios de atributos, e oferece suporte a transações, *triggers*, vistas, cursores, *stored* *procedures,* entre outros.

* + 1. Relações Base

Nesta secção foram escolhidas as formas de representação das tabelas e a informação nelas contida durante a implementação do modelo físico. Visto o modelo lógico ter sido esquematizado no Workbench do MySQL, este processo foi automático, pois durante a criação das tabelas do modelo lógico a informação sobre as entidades, domínio dos atributos, definição de chaves primária e estrangeiras foi representada em tabelas, e estas tabelas servem de suporte ao esquema físico.

A informação corresponde àquela no dicionário de dados, e a definição de cada uma das tabelas do físico consta em seguida.

Legenda:

**PK** – Primary Key

**NN** – Not Null

**UQ** – Unique

**B** – Binary

**UN** – Unsigned

**ZF** – Zero-Filed

**AI** – Auto Incremental

**G** – Generated Column

**Default/Expression** – Valor por defeito

* Cliente

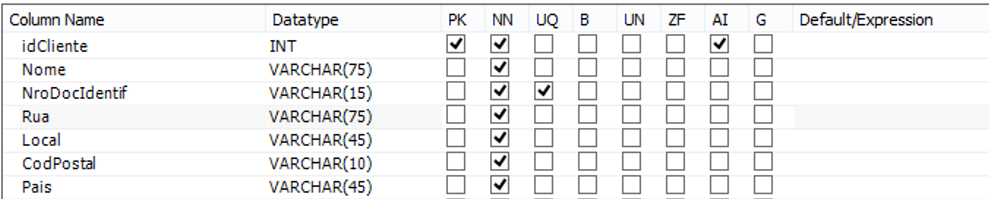


Figura 14 - Tabela representativa da entidade Cliente

* Reserva

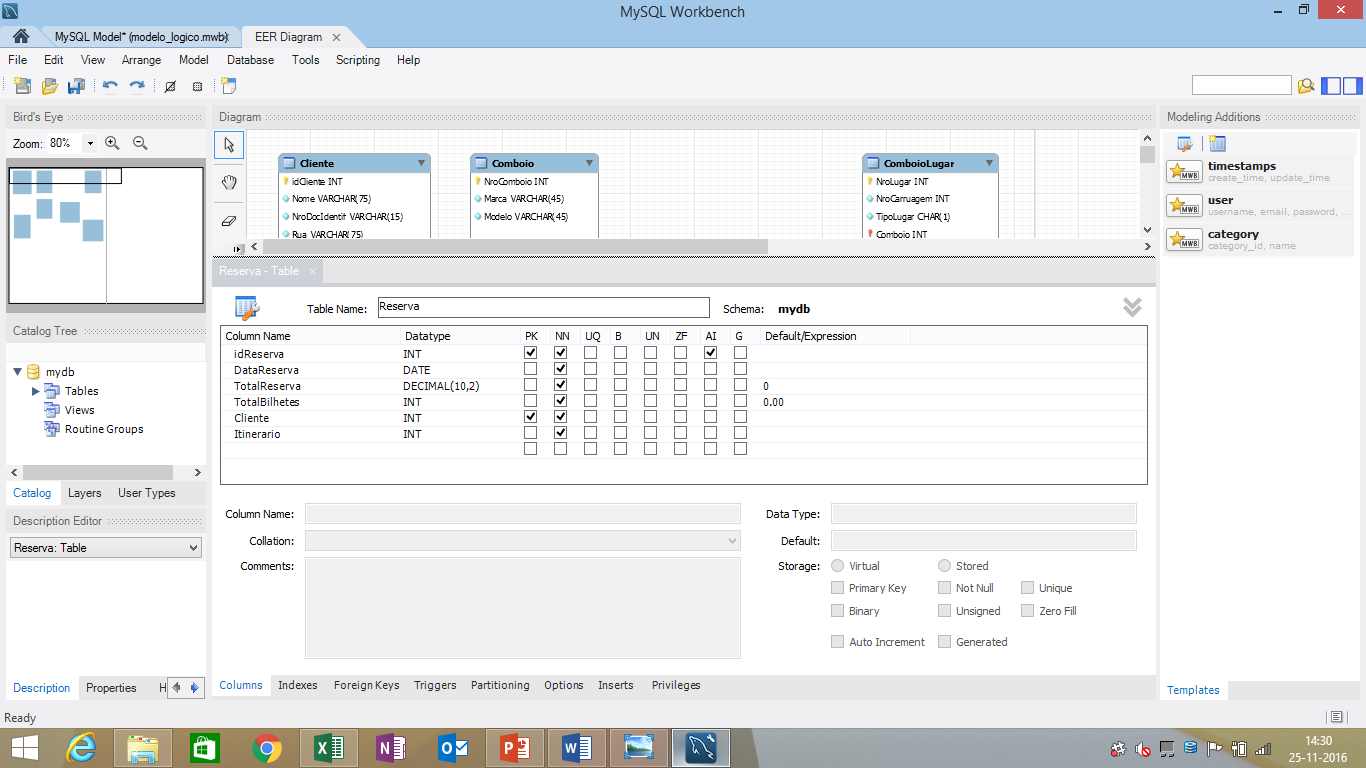


Figura 15 - Tabela representativa da entidade Reserva

* Percurso

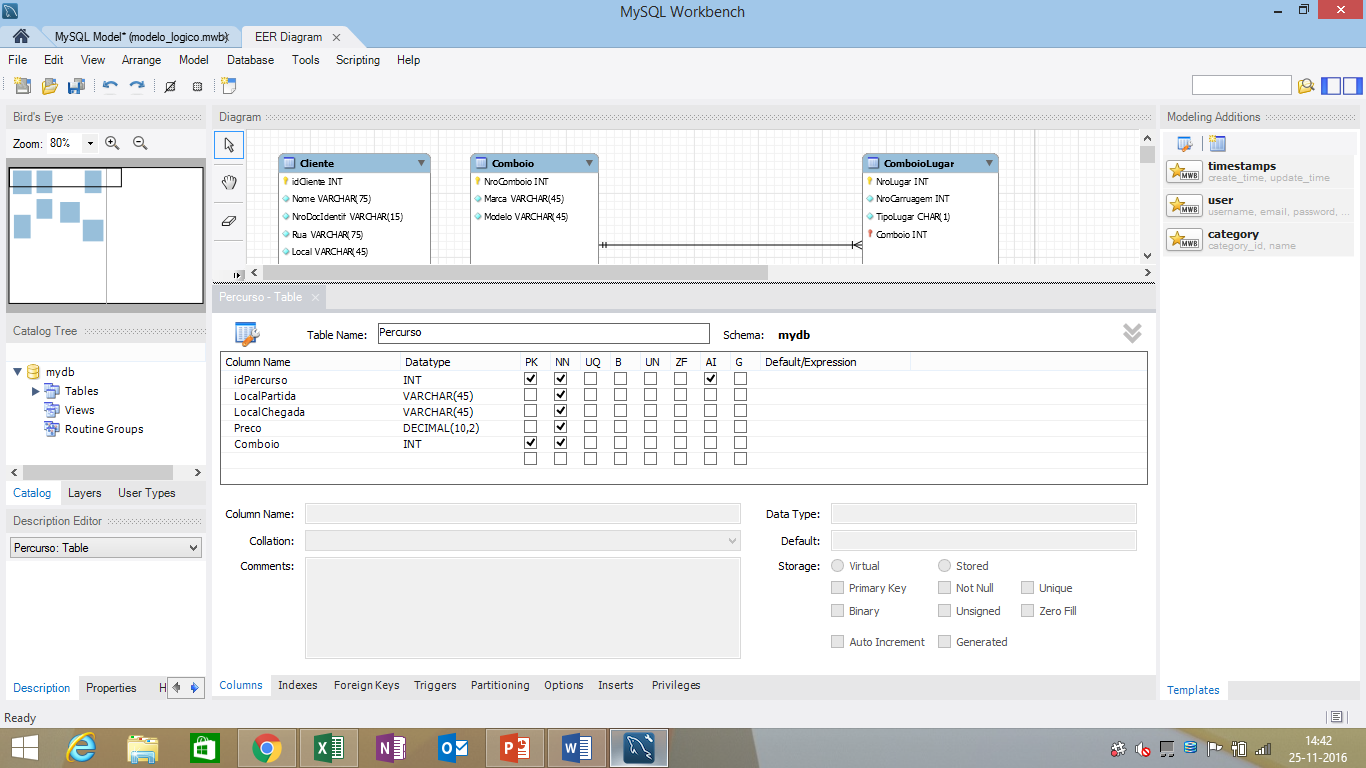


Figura 16 - Tabela representativa da entidade Percurso

* Itinerario

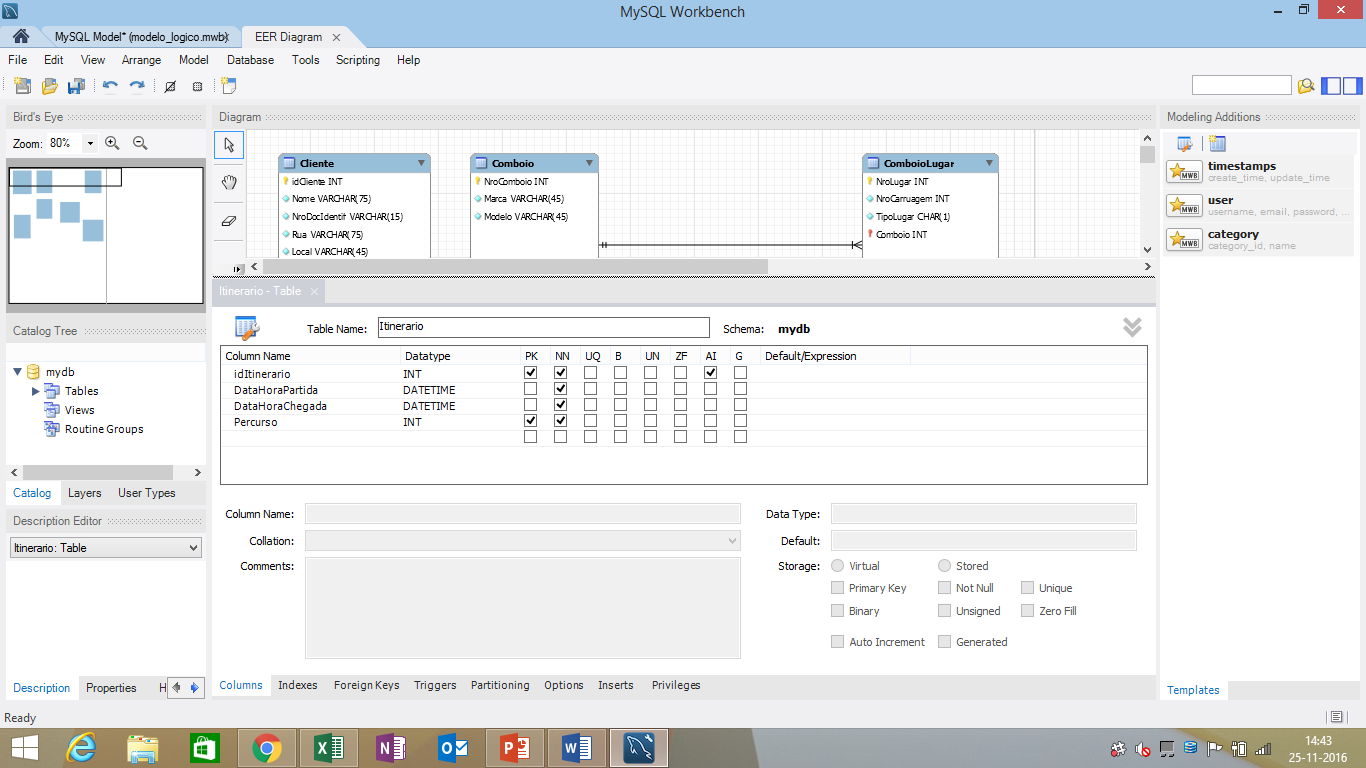


Figura 17 - Tabela representativa da entidade Itinerario

* Comboio

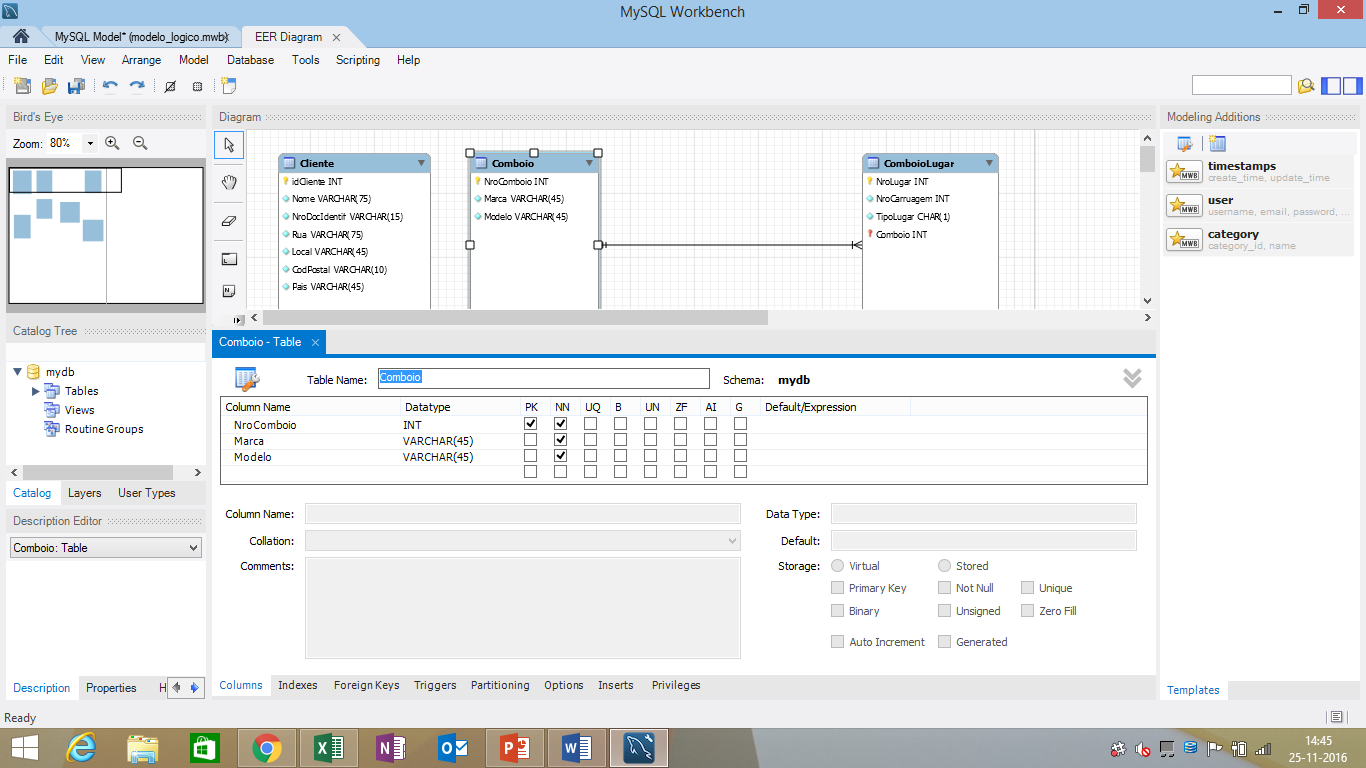


Figura 18 - Tabela representativa da entidade Comboio

* ComboioLugar

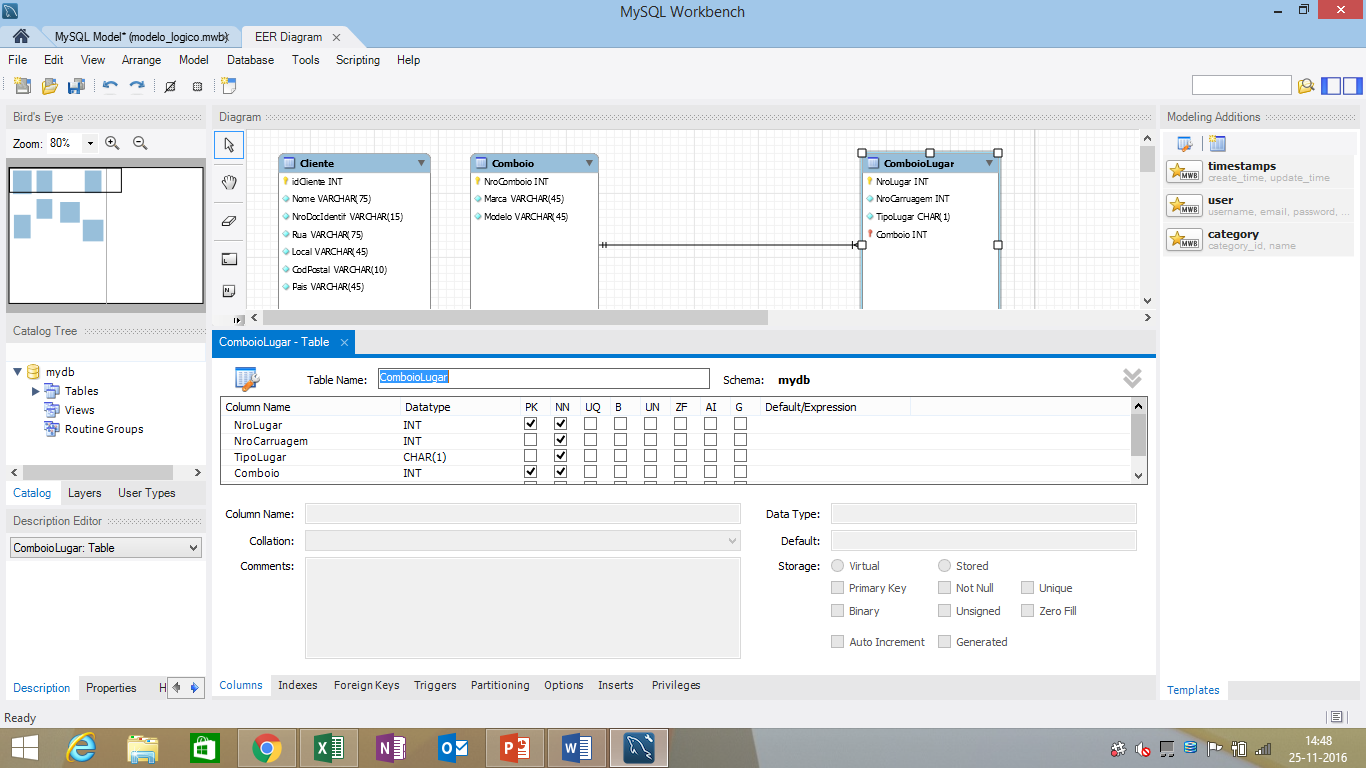


Figura 19 - Tabela representativa do atributo multivalor composto ComboioLugar

* ReservaBilhete

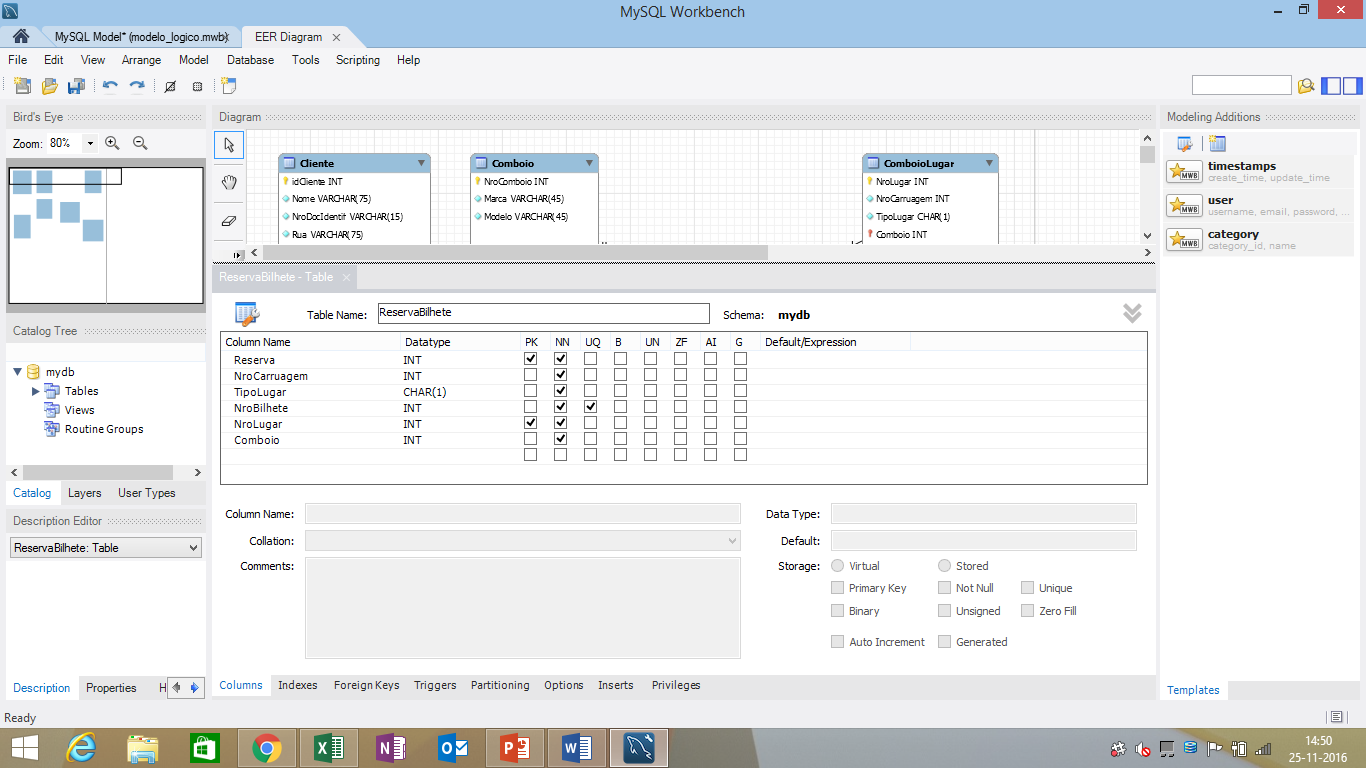


Figura 20 - Tabela representativa do atributo multivalor composto ReservaBilhete

* + 1. Representação de Atributos Derivados

Nesta fase é descrita a forma como os atributos derivados foram representados aquando a implementação física do esquema lógico. Dado nenhum ter sido definido, tal documentação não é expressa.

* + 1. Restrições Gerais

As restrições do modelo foram implementadas por via de métodos, dado o suporte limitado do *mysql* à instrução CHECK da linguagem *SQL,* que impede que algumas das verificações apresentadas pudessem estar associadas às tabelas propriamente ditas.

* Utilizador não pode exceder máximo de renovações permitidas

Esta restrição foi garantida no método de renovação de requisição, através do seguinte código:

SELECT NrRenovacoes, NroMaxRenovacoes INTO v\_numRenovacoes, v\_numMaxRenovacoes

FROM requisicao

WHERE idRequisicao = p\_idRequisicao;

IF v\_numRenovacoes = v\_numMaxRenovacoes THEN SET mustRollback = 1; END IF;

* Utilizador não pode efectuar requisição se o livro não for requisitável ou tiver reservas por parte de outros utilizadores

Esta restrição foi garantida no método de requisição de um exemplar, através do seguinte código:

SELECT E.Disponibilidade INTO v\_disponibilidadeExemplar

FROM Exemplar E

WHERE E.idExemplar = p\_IdExemplar;

IF v\_disponibilidadeExemplar != 2 THEN SET mustRollback = 1; END IF;

SELECT COUNT(Exemplar) INTO v\_numeroReservasOutrosUtilizadores

FROM `exemplar-reservado-utilizador` ERU

WHERE Exemplar = p\_IdExemplar AND Utilizador != p\_Utilizador AND Estado = 1;

IF v\_numeroReservasOutrosUtilizadores > 0 THEN SET mustRollback = 1; END IF;

* Utilizador apenas pode efectuar reserva se o exemplar for requisitável

Esta restrição foi garantida no método de reserva de um exemplar, através do seguinte código:

SELECT E.Disponibilidade INTO v\_disponibilidade

FROM exemplar E

WHERE E.idExemplar = p\_idExemplar;

SELECT COUNT(Exemplar) INTO v\_numReservas

FROM `exemplar-reservado-utilizador`

WHERE Exemplar = p\_idExemplar AND (Estado = 0 OR Estado = 1);

IF v\_disponibilidade = 0 THEN SET mustRollback = 1; END IF;

IF v\_disponibilidade = 1 OR v\_numReservas > 0 THEN SET v\_estadoAInserir = 0; END IF;

IF v\_disponibilidade = 2 AND v\_numReservas = 0 THEN SET v\_estadoAInserir = 1; END IF;

## 4.2 Escolha de Índices

Considerou-se que os índices para as chaves primárias e estrangeiras são suficientes para garantir alguma eficiência nas queries.

## 4.3 Análise de Transações

Nesta secção analisa-se de que forma as transações analisadas no na validação do modelo conceptual e lógico foram implementadas usando SQL.

### 4.3.1 Qual(is) a(s) localização(ões) (piso, estante e prateleira) dos exemplares de um livro com determinado título?

Para resposta a esta transação foi criado o método sp\_localizacao\_exemplares() que pode ser visto de seguida:

CREATE PROCEDURE `sp\_localizacao\_exemplares`(IN p\_titulo VARCHAR(250))

BEGIN

SELECT DISTINCT Loc.\*

FROM (SELECT \*

FROM livro

WHERE Titulo = p\_titulo) L, Exemplar E, localizacao Loc

WHERE L.idLivro = E.Livro AND E.Localizacao = Loc.idLocal;

END

### 4.3.2 Efectuar requisição

Para dar resposta a esta transação foi criado um método com a seguinte assinatura:

CREATE PROCEDURE sp\_efectuar\_requisicao(IN p\_IdExemplar INT, IN p\_Utilizador INT, IN p\_DataRequisicao DATE, IN p\_DataEntrega DATE, IN p\_NrMaxRenovacoes INT)

Visto se tratar de um método algo complexo, o código será apresentado de forma faseada com respectivos comentários. Em primeiro lugar, visto o método se tratar de uma transação, além das variáveis “normais” inicializa-se também um handler para controlar a transação:

DECLARE mustRollback INT DEFAULT 0;

DECLARE v\_numeroReservasOutrosUtilizadores INT DEFAULT 0;

DECLARE v\_disponibilidadeExemplar INT;

DECLARE Erro BOOL DEFAULT 0;

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET Erro = 1;

SET autocommit = 0;

SET SQL\_SAFE\_UPDATES = 0;

START TRANSACTION;

Primeiro, é necessário verificar se o exemplar esta disponivel para reserva e ativar a flag *mustRollback* caso isso não se verifique:

SELECT E.Disponibilidade INTO v\_disponibilidadeExemplar

FROM Exemplar E

WHERE E.idExemplar = p\_IdExemplar;

IF v\_disponibilidadeExemplar != 2 THEN SET mustRollback = 1; END IF;

É ainda necessário assegurar que nenhum utilizador efectuou reserva sobre o mesmo exemplar:

SELECT COUNT(Exemplar) INTO v\_numeroReservasOutrosUtilizadores

FROM `exemplar-reservado-utilizador` ERU

WHERE Exemplar = p\_IdExemplar AND Utilizador != p\_Utilizador AND Estado = 1;

IF v\_numeroReservasOutrosUtilizadores > 0 OR Erro THEN SET mustRollback = 1; END IF;

Caso a requisição tenha sido feita a partir de uma reserva, actualizar o estado da reserva para “livro entregue”:

UPDATE `exemplar-reservado-utilizador`

SET Estado = 2

WHERE Estado = 1 AND Exemplar = p\_IdExemplar AND Utilizador = p\_Utilizador;

Por último, actualizar o estado do exemplar para “requisitado”:

UPDATE exemplar

SET Disponibilidade = 1

WHERE idExemplar = p\_IdExemplar;

IF Erro OR mustRollback = 1 THEN ROLLBACK; ELSE COMMIT; END IF;

SET SQL\_SAFE\_UPDATES = 1;

END;

### 4.3.3 Efectuar uma reserva

Para dar resposta a esta transação foi criado um método com a seguinte assinatura:

CREATE PROCEDURE sp\_efectuar\_reserva(IN p\_idExemplar INT, IN p\_idUtilizador INT, IN p\_DataReserva DATE)

Visto se tratar de um método algo complexo, o código será apresentado de forma faseada com respectivos comentários. Em primeiro lugar, visto o método se tratar de uma transação, além das variáveis “normais” inicializa-se também um handler para controlar a transação:

DECLARE mustRollback INT DEFAULT 0;

DECLARE v\_numReservas INT DEFAULT 0;

DECLARE v\_disponibilidade INT;

DECLARE v\_estadoAInserir INT;

DECLARE Erro BOOL DEFAULT 0;

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET Erro = 1;

SET autocommit = 0;

SET SQL\_SAFE\_UPDATES = 0;

START TRANSACTION;

Começa-se por verificar se o exemplar é requisitável e quantas reservas existem sobre o mesmo:

SELECT E.Disponibilidade INTO v\_disponibilidade

FROM exemplar E

WHERE E.idExemplar = p\_idExemplar;

SELECT COUNT(Exemplar) INTO v\_numReservas

FROM `exemplar-reservado-utilizador`

WHERE Exemplar = p\_idExemplar AND (Estado = 0 OR Estado = 1);

Caso o exemplar não seja sequer requisitável, a variável *mustRollback* toma o valor 1. Caso alguém tenha o exemplar requisitado actualmente ou já existam reservas sobre o mesmo, definir estado da reserva a adicionar como “reserva pedida”. Se o exemplar estiver disponível e não houver reservas sobre ele, definir estado da reserva a adicionar como “livro pronto a levantar”:

IF v\_disponibilidade = 0 THEN SET mustRollback = 1; END IF;

IF v\_disponibilidade = 1 OR v\_numReservas > 0 THEN SET v\_estadoAInserir = 0; END IF;

IF v\_disponibilidade = 2 AND v\_numReservas = 0 THEN SET v\_estadoAInserir = 1; END IF;

Por fim, inserir a reserva:

INSERT INTO `exemplar-reservado-utilizador`

(Exemplar,Utilizador, DataReserva, Estado)

VALUES

(p\_idExemplar, p\_idUtilizador, p\_DataReserva, v\_estadoAInserir);

IF Erro OR mustRollback = 1 THEN ROLLBACK; ELSE COMMIT; END IF;

SET SQL\_SAFE\_UPDATES = 1;

END;

## 4.4 Estimativa dos Requisitos de Espaço em Disco

### 4.4.2 Povoamento e tamanho inicial

Para melhor análise dos requisitos em termos de espaço foi efectuado um povoamento da base de dados de tamanho semelhante ao indicado anteriormente na secção 2.5. Desta forma é possível ter uma ideia do espaço inicial da base de dados. Depois de efectuado o povoamento, executou-se a seguite querie SQL para conhecer os tamanhos das tabelas:

SELECT TABLE\_NAME 'Nome Tabela', table\_rows 'Nº Registos', round(data\_length/1024,2) 'Dados (KB)', round(data\_length/1024 /table\_rows,2) 'KB/Registo',round(index\_length/1024,2) 'Indices (KB)', round(((data\_length + index\_length)/1024/1024),3) 'Total MB'

FROM information\_schema.TABLES WHERE table\_schema = 'bgum';

Cujos resultados se apresentam de seguida:

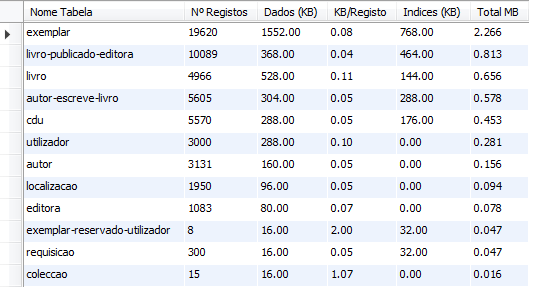


Figura 26 – Nº de registos e tamanho inicial das tabelas

De seguida foi efectuada a seguinte querie para saber o espaço total ocupado pela base de dados:

SELECT table\_schema 'DB Name', SUM(data\_length + index\_length) / 1024 / 1024 'DB Size in MB'

FROM information\_schema.tables

WHERE table\_schema = 'bgum';

Cujo resultado se apresenta de seguida:

espacoOcupadoTotal

Conclui-se assim que a base de dados necessita de cerca de 5.5 MB inicialmente.

### 4.4.3 Crescimento Futuro

Tal como referido na secção 3.5, as requisições e reservas serão o principal factor de aumento do espaço em disco da base de dados. Por dia há cerca de 10 reservas. Ao fim de um ano há 10\*365 = 3650 reservas. Há 50 requisições por dia, o que ao fim de um ano perfaz 50\*65= 18.250 requisições. Por ano há ainda cerca de 3.000 novos utilizadores. Usando os dados da figura 33 pode-se então prever o aumento aproximado da base de dados ao fim de um ano:

3650\*2+18.250\*0,05 + 3000\*0,1= 8512.5 KB = 8.3 MB.

## 4.5 Definição das Vistas de Utilização e Regras de Acesso

Não foram identificadas vistas de utilizador durante a recolha de requisitos ou durante o desenvolvimento do modelo conceptual. No entanto, poderiam ser considerados dois tipos de utilizador da base se dados, os funcionários e os utentes da biblioteca. Os funcionários teriam todas as permissões, exceto a manipulação (inserção, remoção ou alteração) de utilizadores da base de dados, estando essa função a cargo dos Serviços Académicos. Para retratar esta restrição seriam estipuladas regras de acesso para os funcionários. Os utentes da biblioteca apenas poderiam efetuar consultas sobre livros, e as restantes entidades com eles relacionadas, sendo por isso essas consultas feitas sobre uma vista. Na tabela abaixo estão especificadas as permissões de cada utilizador:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Perfil de Utilzador/Tabela | Utilizador | | | |
| INSERT | UPDATE | DELECT | SELECT |
| Cliente | X | X | X | X |
| Reserva | X | X | X | X |
| ReservaBilhete | X | X | X | X |
| Percurso | X | X | X | X |
| Itinerario | X | X | X | X |
| Comboio | X | X | X | X |
| ComboioLugar | X | X | X | X |

Tabela - Permissões de cada tipo de utilizador

(I – Insert, U – Update, D- Delete, S - Select)

Para simular estas permissões, foram realizadas as seguintes queries:

CREATE VIEW vwUserLivro AS

SELECT

Titulo AS "Título",

A.PrimeirosNomes AS "Nome Autor",

A.apelido AS "Apelido",

C.Designacao AS "Coleção",

ED.Designacao AS "Editora",

LE.Edicao AS "Número da Edição",

LE.Ano AS "Ano de Edição",

COUNT(E.livro) AS "Número de Exemplares",

CodBArras AS "Código de Barras",

ISBN,

ISSN

FROM livro AS L INNER JOIN coleccao AS C

ON L.Coleccao = C.idColeccao

LEFT JOIN exemplar as E

ON (L.idlivro = E.livro)

LEFT OUTER JOIN `autor-escreve-livro` AS AL

ON L.idLivro = AL.livro

INNER JOIN autor AS A

ON A.idAutor = AL.autor

LEFT JOIN `livro-publicado-editora` as LE

ON (L.idlivro = LE.livro)

INNER JOIN editora as ED

ON ED.idEditora = LE.editora

GROUP BY L.titulo;

A vista ‘vwUserLivro’ simula a visão de um utente, que assim tem acesso a toda a informação referente aos livros, inclusive o número de exemplares de cada um.

CREATE VIEW vwExemplares AS

SELECT

L.Titulo AS "Título",

condicao AS "Condição",

Loc.Piso,

Loc.Estante,

Loc.Prateleira

FROM exemplar AS E INNER JOIN localizacao AS Loc

ON E.localizacao = Loc.idLocal

INNER JOIN livro AS L

ON L.idLivro = E.Livro

GROUP BY L.titulo;

A vista ‘vwExemplares’ permite aos utentes da biblioteca acesso a informações sobre o exemplar de cada livro, como o seu estado de conservação e localização.

No caso dos funcionários, para garantir que estes não têm permissões para manipular os utilizadores, bastaria apenas retirar-lhes esses privilégios:

* Exemplo da criação de um Funcionário:

CREATE USER 'Funcionario'@'localhost'

IDENTIFIED BY 'bgum2016';

* Concedem-se todos os privilégios:

GRANT ALL PRIVILEGES

ON bgum.\*

TO 'Funcionario'@'localhost';

* E retiram-se os privilégios de inserção, alteração e remoção na tabela Utilizador :

REVOKE INSERT, UPDATE, DELETE

ON bgum.utilizador

FROM 'Funcionario'@'localhost';

.

1. Conclusões e Trabalho Futuro

<<Elaborar uma apreciação crítica sobre o trabalho realizado, apontando os seus pontos fortes e fracos. Adicionalmente, caso se aplique, enunciar eventuais tarefas a realizar futuramente ou novas opções para estender o trabalho realizado.>>

# Referências

Connolly, T. & Begg, C., 2005. *Database Systems - A practical approach to design, implementation and management.* s.l.:Pearson Education Limited.

# Lista de Siglas e Acrónimos

<<Apresentar uma lista com todas as siglas e acrónimos utilizados durante a realização do trabalho. O formato base para esta lista deverá ser da forma como abaixo se apresenta.>>

**BD** Base de Dados

DW Data Warehouse

OLTP On-Line Analytical Processing

Anexos

<<Os anexos deverão ser utilizados para a inclusão de informação adicional necessária para uma melhor compreensão do relatório o para complementar tópicos, secções ou assuntos abordados. Os anexos criados deverão ser numerados e possuir uma designação. Estes dados permitirão complementar o Índice geral do relatório relativamente à enumeração e apresentação dos diversos anexos.>>