Projeto Bases de Dados

2016/2017

**4ª feira 8h30 - Grupo 9**

**81900** – Nuno Anselmo

**81936** – Liliana Oliveira

**82047** – André Mendes

Parte 3

Esforço dedicado:

**81900 –**

**81936 –**

**82047 –**

Schema

drop table if exists estado;

drop table if exists paga;

drop table if exists aluga;

drop table if exists reserva;

drop table if exists oferta;

drop table if exists posto;

drop table if exists espaco;

drop table if exists fiscaliza;

drop table if exists arrenda;

drop table if exists alugavel;

drop table if exists edificio;

drop table if exists fiscal;

drop table if exists user;

create table user (

nif varchar(9) not null unique,

nome varchar(80) not null,

telefone varchar(26) not null,

primary key(nif));

create table fiscal (

id int not null unique,

empresa varchar(255) not null,

primary key(id));

create table edificio (

morada varchar(255) not null unique,

primary key(morada));

create table alugavel (

morada varchar(255) not null,

codigo varchar(255) not null,

foto varchar(255) not null,

primary key(morada, codigo),

foreign key(morada) references edificio(morada) ON UPDATE CASCADE ON DELETE CASCADE);

create table arrenda (

morada varchar(255) not null,

codigo varchar(255) not null,

nif varchar(9) not null,

primary key(morada, codigo),

foreign key(morada, codigo) references alugavel(morada, codigo) ON UPDATE CASCADE ON DELETE CASCADE,

foreign key(nif) references user(nif) ON UPDATE CASCADE ON DELETE CASCADE);

create table fiscaliza (

id int not null,

morada varchar(255) not null ,

codigo varchar(255) not null ,

primary key(id, morada, codigo),

foreign key(morada, codigo) references arrenda(morada, codigo) ON UPDATE CASCADE ON DELETE CASCADE,

foreign key(id) references fiscal(id) ON UPDATE CASCADE ON DELETE CASCADE);

create table espaco (

morada varchar(255) not null,

codigo varchar(255) not null,

primary key(morada, codigo),

foreign key(morada, codigo) references alugavel(morada, codigo) ON UPDATE CASCADE ON DELETE CASCADE);

create table posto (

morada varchar(255) not null,

codigo varchar(255) not null,

codigo\_espaco varchar(255) not null,

primary key(morada, codigo),

foreign key(morada, codigo) references alugavel(morada, codigo) ON UPDATE CASCADE ON DELETE CASCADE,

foreign key(morada, codigo\_espaco) references espaco(morada, codigo) ON UPDATE CASCADE ON DELETE CASCADE);

create table oferta (

morada varchar(255) not null,

codigo varchar(255) not null,

data\_inicio date not null,

data\_fim date not null,

tarifa numeric(19,4) not null,

primary key(morada, codigo, data\_inicio),

foreign key(morada, codigo) references alugavel(morada, codigo) ON UPDATE CASCADE ON DELETE CASCADE);

create table reserva (

numero varchar(255) not null unique,

primary key(numero));

create table aluga (

morada varchar(255) not null,

codigo varchar(255) not null,

data\_inicio date not null,

nif varchar(9) not null,

numero varchar(255) not null,

primary key(morada, codigo, data\_inicio, nif, numero),

foreign key(morada, codigo, data\_inicio) references oferta(morada, codigo, data\_inicio) ON UPDATE CASCADE ON DELETE CASCADE,

foreign key(nif) references user(nif) ON UPDATE CASCADE ON DELETE CASCADE,

foreign key(numero) references reserva(numero) ON UPDATE CASCADE ON DELETE CASCADE);

create table paga (

numero varchar(255) not null unique,

data timestamp not null,

metodo varchar(255) not null,

primary key(numero),

foreign key(numero) references reserva(numero) ON UPDATE CASCADE ON DELETE CASCADE);

create table estado (

numero varchar(255) not null,

time\_stamp timestamp not null,

estado varchar(255) not null,

primary key(numero, time\_stamp),

foreign key(numero) references reserva(numero) ON UPDATE CASCADE ON DELETE CASCADE);

Queries

a) Quais os espaços com postos que nunca foram alugados?

SELECT DISTINCT p.morada,

p.codigo\_espaco

FROM posto p

LEFT OUTER JOIN aluga a

ON p.morada = a.morada

AND p.codigo = a.codigo

WHERE a.numero IS NULL;

b) Quais edifícios com um número de reservas superior à média?

SELECT morada

FROM aluga

GROUP BY morada

HAVING Count(\*) > ( (SELECT Count(\*)

FROM aluga) / (SELECT Count(\*)

FROM edificio) );

c) Quais utilizadores cujos alugáveis foram fiscalizados sempre pelo mesmo fiscal?

SELECT nif

FROM fiscaliza

NATURAL JOIN arrenda

GROUP BY nif

HAVING Count(DISTINCT id) = 1;

d) Qual o montante total realizado (pago) por cada espaço durante o ano de 2016? Assuma que a tarifa indicada na oferta é diária. Deve considerar os casos em que o espaço foi alugado totalmente ou por postos.

SELECT morada,

codigo,

Sum(montante)

FROM ((SELECT morada,

codigo\_espaco AS codigo,

( Datediff(data\_fim, data\_inicio) + 1 ) \* tarifa AS montante

FROM aluga

NATURAL JOIN oferta

NATURAL JOIN posto

NATURAL JOIN paga

WHERE Year(data) = 2016)

UNION

(SELECT morada,

codigo,

( Datediff(data\_fim, data\_inicio) + 1 ) \* tarifa AS montante

FROM aluga

NATURAL JOIN oferta

NATURAL JOIN espaco

NATURAL JOIN paga

WHERE Year(data) = 2016)) t

GROUP BY morada,

codigo;

e) Quais os espaços de trabalho cujos postos nele contidos foram todos alugados? (Por alugado entende-se um posto de trabalho que tenha pelo menos uma oferta aceite, independentemente das suas datas.)

SELECT morada,

codigo\_espaco

FROM (SELECT morada,

codigo\_espaco,

Count(\*) AS count

FROM posto

GROUP BY morada,

codigo\_espaco) r1

NATURAL JOIN (SELECT morada,

codigo\_espaco,

Count(\*) AS count

FROM (SELECT morada,

codigo\_espaco

FROM posto

NATURAL JOIN aluga

NATURAL JOIN estado

WHERE estado = 'Aceite') p

GROUP BY morada,

codigo\_espaco) r2;

x) Qual o montante total realizado (pago) por um dado espaço?

SELECT morada,

codigo,

Sum(montante)

FROM ((SELECT morada,

codigo\_espaco AS codigo,

( Datediff(data\_fim, data\_inicio) + 1 ) \* tarifa AS montante

FROM aluga

NATURAL JOIN oferta

NATURAL JOIN posto

NATURAL JOIN paga

WHERE codigo\_espaco = 'Central'

AND morada = 'ISEL')

UNION

(SELECT morada,

codigo,

( Datediff(data\_fim, data\_inicio) + 1 ) \* tarifa AS montante

FROM aluga

NATURAL JOIN oferta

NATURAL JOIN espaco

NATURAL JOIN paga

WHERE codigo = 'Central'

AND morada = 'ISEL')) t

GROUP BY morada,

codigo;

y) Qual o montante total realizado (pago) por cada espaço do utilizador de nif '143856248'?

SELECT morada,

codigo,

Sum(montante)

FROM ((SELECT morada,

codigo\_espaco AS codigo,

( Datediff(data\_fim, data\_inicio) + 1 ) \* tarifa AS montante

FROM aluga

NATURAL JOIN oferta

NATURAL JOIN posto

NATURAL JOIN paga

NATURAL JOIN (SELECT morada,

codigo

FROM arrenda

WHERE nif = '113056729') u1)

UNION

(SELECT morada,

codigo,

( Datediff(data\_fim, data\_inicio) + 1 ) \* tarifa AS montante

FROM aluga

NATURAL JOIN oferta

NATURAL JOIN espaco

NATURAL JOIN paga

NATURAL JOIN (SELECT morada,

codigo

FROM arrenda

WHERE nif = '113056729') u2)) t

GROUP BY morada,

codigo;

z) Quais as ofertas que não têm reservas associadas cujos estados sejam Paga ou Aceite?

SELECT o.morada,

o.codigo,

o.data\_inicio,

o.data\_fim,

o.tarifa

FROM oferta o

LEFT OUTER JOIN (SELECT morada,

codigo

FROM aluga

NATURAL JOIN (SELECT numero

FROM estado e

NATURAL JOIN (SELECT

numero,

Max(time\_stamp) AS time\_stamp

FROM estado

GROUP BY numero) f

WHERE estado = "aceite"

OR estado = "paga") z) s

ON o.morada = s.morada

AND o.codigo = s.codigo

WHERE s.codigo IS NULL;

Triggers

a) Não podem existir ofertas com datas sobrepostas

DROP TRIGGER IF EXISTS insertOffer;

DELIMITER //

CREATE TRIGGER insertOffer BEFORE INSERT ON oferta

FOR EACH row

begin

DECLARE registers INTEGER;

SELECT Count(\*)

INTO registers

FROM oferta

WHERE codigo = new.codigo

AND morada = new.morada

AND new.data\_inicio <= data\_fim

AND new.data\_fim >= data\_inicio;

IF registers > 0 THEN

CALL raise\_error;

END IF;

END//

delimiter ;

b) A data de pagamento de uma reserva paga tem de ser superior ao timestamp do último estado dessa reserva

DROP TRIGGER IF EXISTS insertPay;

DELIMITER //

CREATE TRIGGER insertPay BEFORE INSERT ON paga

FOR EACH row

begin

DECLARE last TIMESTAMP;

SELECT time\_stamp

INTO last

FROM estado

WHERE numero = new.numero

ORDER BY time\_stamp DESC

LIMIT 1;

IF (last >= new.data) THEN

CALL raise\_error;

END IF;

END//

delimiter ;

PHP

// User.php

class User extends Model{

// ...

public static function find($nif){

try {

$stmt = self::$connection->prepare('SELECT \* FROM user WHERE user.nif = :nif');

$stmt->bindValue(':nif', $nif);

$stmt->execute();

$stmt->execute();

if ($stmt->rowCount() == 0) return null;

$row = $stmt->fetch();

return new User($row['nif'], $row['nome'], $row['telefone']);

} catch (PDOException $e) {}

return null;

}

}

//Building.php

class Building extends Model{

// ..

public static function find($address){

try {

$stmt = self::$connection->prepare('SELECT \* FROM edificio WHERE morada = :morada');

$stmt->bindValue(':morada', $address);

$stmt->execute();

if ($stmt->rowCount() == 0) return null;

$row = $stmt->fetch();

return new Building($row['morada']);

} catch (PDOException $e) {}

return null;

}

public static function all(){

$buildings = [];

try {

$stmt = self::$connection->prepare('SELECT \* FROM edificio ORDER BY morada');

$stmt->execute();

foreach ($stmt->fetchAll() as $row) {

array\_push($buildings, new Building($row['morada']));

}

} catch (PDOException $e) {}

return $buildings;

}

public static function create($address){

try {

$stmt = self::$connection->prepare('INSERT INTO edificio(morada) VALUES(:morada)');

$stmt->bindParam(':morada', $address);

if ($stmt->execute())

return new Building($address);

} catch (PDOException $e) {}

return null;

}

public function delete(){

try {

// Remove the building

$stmt = self::$connection->prepare('DELETE FROM edificio WHERE morada = :morada');

$stmt->bindValue(':morada', $this->getAddress());

return $stmt->execute();

} catch (PDOException $e) {}

return false;

}

public function getWorkspaces(){

$workspaces = [];

try {

require\_once 'Workspace.php';

$stmt = self::$connection->prepare('SELECT \* FROM espaco WHERE morada = :morada ORDER BY codigo');

$stmt->bindValue(':morada', $this->getAddress());

$stmt->execute();

foreach ($stmt->fetchAll() as $row) {

$workspace = new Workspace(

$this,

$row['codigo']

);

array\_push($workspaces, $workspace);

}

} catch (PDOException $e) {}

return $workspaces;

}

}

// Rentable.php

class Rentable extends Model{

// ...

public function fetchImage(){

if ($this->image == null) {

try {

$stmt = self::$connection->prepare(

'SELECT foto FROM alugavel WHERE morada = :morada AND codigo = :codigo'

);

$stmt->bindValue(':morada', $this->getBuilding()->getAddress());

$stmt->bindValue(':codigo', $this->getCode());

$stmt->execute();

$this->image = $stmt->fetch()['foto'];

} catch (PDOException $e) {}

}

}

public function fetchUser(){

if ($this->user == null) {

try {

$stmt = self::$connection->prepare(

'SELECT nif FROM arrenda WHERE morada = :morada AND codigo = :codigo'

);

$stmt->bindValue(':morada', $this->getBuilding()->getAddress());

$stmt->bindValue(':codigo', $this->getCode());

$stmt->execute();

require\_once 'User.php';

$this->user = User::find($stmt->fetch()['nif']);

} catch (PDOException $e) {}

}

}

public static function allFrom($nif){

$rentables = [];

try {

$stmt = self::$connection->prepare(

'SELECT \* FROM alugavel NATURAL JOIN arrenda WHERE nif = :nif ORDER BY morada, codigo'

);

$stmt->bindValue(':nif', $nif);

$stmt->execute();

require\_once 'Building.php';

require\_once 'User.php';

foreach ($stmt->fetchAll() as $row) {

$rentable = new Rentable(

new Building($row['morada']),

$row['codigo'],

$row['foto'],

User::find($row['nif'])

);

array\_push($rentables, $rentable);

}

} catch (PDOException $e) {}

return $rentables;

}

public static function find($address, $code){

try {

$stmt = self::$connection->prepare(

'SELECT \* FROM alugavel NATURAL JOIN arrenda WHERE morada = :morada AND codigo = :codigo'

);

$stmt->bindValue(':morada', $address);

$stmt->bindValue(':codigo', $code);

$stmt->execute();

if ($stmt->rowCount() == 0) return null;

$row = $stmt->fetch();

require\_once 'Building.php';

require\_once 'User.php';

return new Rentable(new Building($row['morada']), $row['codigo'], $row['foto'], User::find($row['nif']));

} catch (PDOException $e) {}

return null;

}

public static function create($address, $code, $image, $nif){

try {

$stmt = self::$connection->prepare(

'INSERT INTO alugavel(morada, codigo, foto) VALUES(:morada, :codigo, :foto)'

);

$stmt->bindValue(':morada', $address);

$stmt->bindValue(':codigo', $code);

$stmt->bindValue(':foto', $image);

if (!$stmt->execute())

return null;

$stmt = self::$connection->prepare(

'INSERT INTO arrenda(morada, codigo, nif) VALUES(:morada, :codigo, :nif)'

);

$stmt->bindValue(':morada', $address);

$stmt->bindValue(':codigo', $code);

$stmt->bindValue(':nif', $nif);

if ($stmt->execute()) {

require\_once 'Building.php';

require\_once 'User.php';

return new Rentable(new Building($address), $code, $image, User::find($nif));

}

} catch (PDOException $e) {}

return null;

}

public function delete(){

try {

$stmt = self::$connection->prepare('DELETE FROM alugavel WHERE morada = :morada AND codigo = :codigo');

$stmt->bindValue(':morada', $this->getBuilding()->getAddress());

$stmt->bindValue(':codigo', $this->getCode());

return $stmt->execute();

} catch (PDOException $e) {}

return false;

}

}

// Workspace.php

class Workspace extends Rentable{

// ...

public static function find($address, $code){

try {

$stmt = self::$connection->prepare('SELECT \* FROM espaco WHERE morada = :morada AND codigo = :codigo');

$stmt->bindValue(':morada', $address);

$stmt->bindValue(':codigo', $code);

$stmt->execute();

if ($stmt->rowCount() == 0) return null;

$row = $stmt->fetch();

return new Workspace(new Building($row['morada']), $row['codigo']);

} catch (PDOException $e) {}

return null;

}

public static function create($address, $code, $image, $nif){

try {

$rentable = parent::create($address, $code, $image, $nif);

if ($rentable == null)

return null;

$stmt = self::$connection->prepare(

'INSERT INTO espaco(morada, codigo) VALUES(:morada, :codigo)'

);

$stmt->bindValue(':morada', $address);

$stmt->bindValue(':codigo', $rentable->getCode());

if ($stmt->execute())

return new Workspace(new Building($address), $rentable->getCode());

} catch (PDOException $e) {}

return null;

}

public function getWorkstations(){

$workstations = [];

try {

require\_once 'Workstation.php';

$stmt = self::$connection->prepare(

'SELECT \* FROM posto WHERE morada = :morada AND codigo\_espaco = :codigo ORDER BY morada, codigo'

);

$stmt->bindValue(':morada', $this->getBuilding()->getAddress());

$stmt->bindValue(':codigo', $this->getCode());

$stmt->execute();

foreach ($stmt->fetchAll() as $row) {

$workstation = Workstation::find($row['morada'], $row['codigo']);

array\_push($workstations, $workstation);

}

} catch (PDOException $e) {}

return $workstations;

}

public function getTotal(){

try {

require\_once 'Workstation.php';

$stmt = self::$connection->prepare('

SELECT

morada,

codigo,

sum(montante)

FROM ((SELECT

morada,

codigo\_espaco AS codigo,

(datediff(data\_fim, data\_inicio) + 1) \* tarifa AS montante

FROM aluga

NATURAL JOIN oferta

NATURAL JOIN posto

NATURAL JOIN paga

WHERE codigo\_espaco = :codigo

AND morada = :morada)

UNION

(SELECT

morada,

codigo,

(datediff(data\_fim, data\_inicio) + 1) \* tarifa AS montante

FROM aluga

NATURAL JOIN oferta

NATURAL JOIN espaco

NATURAL JOIN paga

WHERE codigo = :codigo

AND morada = :morada)) t

GROUP BY morada, codigo;');

$stmt->bindValue(':morada', $this->getBuilding()->getAddress());

$stmt->bindValue(':codigo', $this->getCode());

if ($stmt->execute() && $stmt->rowCount() > 0)

return $stmt->fetch()[2];

} catch (PDOException $e) {}

return 0;

}

}

// Workstation.php

class Workstation extends Rentable{

// ...

public static function find($address, $code){

try {

$stmt = self::$connection->prepare('SELECT \* FROM posto WHERE morada = :morada AND codigo = :codigo');

$stmt->bindValue(':morada', $address);

$stmt->bindValue(':codigo', $code);

$stmt->execute();

if ($stmt->rowCount() == 0) return null;

$row = $stmt->fetch();

return new Workstation(Workspace::find($address, $row['codigo\_espaco']), $row['codigo']);

} catch (PDOException $e) {}

return null;

}

public static function create($address, $code, $image, $nif, $workspaceCode){

try {

$rentable = parent::create($address, $code, $image, $nif);

if ($rentable == null)

return null;

$stmt = self::$connection->prepare(

'INSERT INTO posto(morada, codigo, codigo\_espaco) VALUES(:morada, :codigo, :codigo\_espaco)'

);

$stmt->bindValue(':morada', $address);

$stmt->bindValue(':codigo', $rentable->getCode());

$stmt->bindValue(':codigo\_espaco', $workspaceCode);

if ($stmt->execute()) {

require\_once 'Workspace.php';

return new Workstation(Workspace::find($address, $workspaceCode), $rentable->getCode(), $image);

}

} catch (PDOException $e) {}

return null;

}

}

// Offer.php

class Offer extends Model{

// ...

public static function find($address, $code, $startDate){

try {

$stmt = self::$connection->prepare(

'SELECT \* FROM oferta WHERE morada = :morada AND codigo = :codigo AND data\_inicio = :data\_inicio'

);

$stmt->bindValue(':morada', $address);

$stmt->bindValue(':codigo', $code);

$stmt->bindValue(':data\_inicio', Database::formatDate($startDate));

$stmt->execute();

if ($stmt->rowCount() == 0) return null;

$row = $stmt->fetch();

require\_once 'Rentable.php';

return new Offer(

Rentable::find($row['morada'], $row['codigo']),

Database::parseDate($row['data\_inicio']),

Database::parseDate($row['data\_fim']),

$row['tarifa']

);

} catch (PDOException $e) {}

return null;

}

public static function create($address, $code, $startDate, $endDate, $price){

try {

$stmt = self::$connection->prepare(

'INSERT INTO oferta(morada, codigo, data\_inicio, data\_fim, tarifa) VALUES (:morada, :codigo, :data\_inicio, :data\_fim, :tarifa)'

);

$stmt->bindValue(':morada', $address);

$stmt->bindValue(':codigo', $code);

$stmt->bindValue(':data\_inicio', Database::formatDate($startDate));

$stmt->bindValue(':data\_fim', Database::formatDate($endDate));

$stmt->bindValue(':tarifa', $price);

if ($stmt->execute()) {

require\_once 'Rentable.php';

return new Offer(Rentable::find($address, $code), $startDate, $endDate, $price);

}

} catch (PDOException $e) {}

return null;

}

public static function allAvailable(){

$offers = [];

try {

$stmt = self::$connection->prepare(

'SELECT o.morada, o.codigo, o.data\_inicio, o.data\_fim, o.tarifa

FROM oferta o LEFT OUTER JOIN (

SELECT morada, codigo

FROM aluga NATURAL JOIN (

SELECT numero

FROM estado e NATURAL JOIN (

SELECT numero, MAX(time\_stamp) AS time\_stamp

FROM estado

GROUP BY numero

) f

WHERE estado = \'Aceite\' OR estado = \'Paga\'

) z

) s

ON o.morada = s.morada

AND o.codigo = s.codigo

WHERE s.codigo IS NULL ORDER BY data\_inicio DESC'

);

$stmt->execute();

require\_once 'Rentable.php';

foreach ($stmt->fetchAll() as $row) {

$offer = new Offer(

Rentable::find($row['morada'], $row['codigo']),

Database::parseDate($row['data\_inicio']),

Database::parseDate($row['data\_fim']),

$row['tarifa']

);

array\_push($offers, $offer);

}

} catch (PDOException $e) {}

return $offers;

}

public static function allFrom($nif){

$offers = [];

try {

$stmt = self::$connection->prepare(

'SELECT \* FROM oferta NATURAL JOIN arrenda WHERE nif = :nif ORDER BY data\_inicio DESC'

);

$stmt->bindValue(':nif', $nif);

$stmt->execute();

require\_once 'Rentable.php';

foreach ($stmt->fetchAll() as $row) {

$offer = new Offer(

Rentable::find($row['morada'], $row['codigo']),

Database::parseDate($row['data\_inicio']),

Database::parseDate($row['data\_fim']),

$row['tarifa']

);

array\_push($offers, $offer);

}

} catch (PDOException $e) {}

return $offers;

}

public function delete(){

try {

$stmt = self::$connection->prepare(

'DELETE FROM oferta WHERE morada = :morada AND codigo = :codigo AND data\_inicio = :data\_inicio'

);

$stmt->bindValue(':morada', $this->getRentable()->getBuilding()->getAddress());

$stmt->bindValue(':codigo', $this->getRentable()->getCode());

$stmt->bindValue(':data\_inicio', Database::formatDate($this->getStartDate()));

return $stmt->execute();

} catch (PDOException $e) {}

return false;

}

}

// Reservation.php

class Reservation extends Model{

// ...

public static function create($address, $code, $startDate, $nif, $number)

{

try {

$stmt = self::$connection->prepare('INSERT INTO reserva(numero) VALUES (:numero)');

$stmt->bindValue(':numero', $number);

if (!$stmt->execute())

return null;

$stmt = self::$connection->prepare(

'INSERT INTO aluga(morada, codigo, data\_inicio, nif, numero) VALUES (:morada, :codigo, :data\_inicio, :nif, :numero)'

);

$stmt->bindValue(':morada', $address);

$stmt->bindValue(':codigo', $code);

$stmt->bindValue(':data\_inicio', Database::formatDate($startDate));

$stmt->bindValue(':nif', $nif);

$stmt->bindValue(':numero', $number);

if (!$stmt->execute())

return null;

require\_once 'ReservationState.php';

$state = State::create($number, new DateTime(), 'Pendente');

if ($state == null)

return null;

return new Reservation(

User::find($nif),

Offer::find($address, $code, $startDate),

$number,

null,

$state

);

} catch (PDOException $e) {}

return null;

}

public static function allFrom($nif){

$reservations = [];

try {

$stmt = self::$connection->prepare(

'SELECT \* FROM aluga WHERE nif = :nif ORDER BY numero'

);

$stmt->bindValue(':nif', $nif);

$stmt->execute();

require\_once 'ReservationState.php';

require\_once 'ReservationPayment.php';

require\_once 'Offer.php';

require\_once 'User.php';

foreach ($stmt->fetchAll() as $row) {

$reservation = new Reservation(

User::find($row['nif']),

Offer::find($row['morada'], $row['codigo'], Database::parseDate($row['data\_inicio'])),

$row['numero'],

Payment::find($row['numero']),

State::findMostRecent($row['numero'])

);

array\_push($reservations, $reservation);

}

} catch (PDOException $e) {}

return $reservations;

}

}

// ReservationPayment.php

class Payment extends Model{

// ...

public static function create($number, $timestamp, $method)

{

try {

$stmt = self::$connection->prepare('INSERT INTO paga(numero, data, metodo) VALUES (:numero, :data, :metodo)');

$stmt->bindValue(':numero', $number);

$stmt->bindValue(':data', Database::formatTimestamp($timestamp));

$stmt->bindValue(':metodo', $method);

if ($stmt->execute())

return new Payment($timestamp, $method);

} catch (PDOException $e) {}

return null;

}

public static function find($number){

try {

$stmt = self::$connection->prepare('SELECT \* FROM paga WHERE numero = :numero');

$stmt->bindValue(':numero', $number);

if ($stmt->execute() && $stmt->rowCount() > 0) {

$row = $stmt->fetch();

return new Payment(Database::parseTimestamp($row['data']), $row['metodo']);

}

} catch (PDOException $e) {}

return null;

}

}

// ReservationState.php

class State extends Model{

// ...

public static function create($number, $timestamp, $state){

try {

$stmt = self::$connection->prepare('INSERT INTO estado(numero, time\_stamp, estado) VALUES (:numero, :time\_stamp, :estado)');

$stmt->bindValue(':numero', $number);

$stmt->bindValue(':time\_stamp', Database::formatTimestamp($timestamp));

$stmt->bindValue(':estado', $state);

if ($stmt->execute())

return new State($timestamp, $state);

} catch (PDOException $e) {}

return null;

}

public static function findMostRecent($number){

try {

$stmt = self::$connection->prepare(

'SELECT \* FROM estado WHERE numero = :numero ORDER BY time\_stamp DESC LIMIT 1'

);

$stmt->bindValue(':numero', $number);

if ($stmt->execute() && $stmt->rowCount() > 0) {

$row = $stmt->fetch();

return new State(Database::parseTimestamp($row['time\_stamp']), $row['estado']);

}

} catch (PDOException $e) {}

return null;

}

}