University of Information Technology Warsaw

Final project
“Hotel”
Database relational model

Subject: Data Bases
Professor: PhD Huber Szczepaniuk
Student: Yevgeniya Li, album number 4591
Group: IDE-1
Date: 14-05-2019

Hotel is a business database relational model which represents the simple model of reserving hotel rooms by customers. It contains information about how many rooms are available for booking (number of the room), personal data about clients (first and last names and country), history of reservations per each room and client with dates of the stay and history of payment transactions which is tied to reservations so it could be easy to follow the flow of financial operations per each reservation. Also there is the table that describes room categories with price for each category (for example prices are the same for all rooms that are in the same category Double) so we could avoid repeating the prices per each particular room so we could keep our table in the third normal form. Also the table with types of payment prevents repeating the same payment types for table payments. All columns store single attributes and do not store multiple not related with each other values from business perspective. Columns are independent from each other and have unique names in the context of particular table. There are also two views:

1. The first was created on the base of rooms and reservations table and contains information about how many times each room was reserved
2. The second was created on the base of clients and reservations and contains information about how many times each client reserved a room

```
CREATE SCHEMA HOTEL;
USE HOTEL;
CREATE TABLE IF NOT EXISTS payment_types (
  `id` INT NOT NULL,
  `name` VARCHAR(100) NOT NULL,
  PRIMARY KEY(`id`),
  CONSTRAINT `u_payment_types` UNIQUE (`name`))
  ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS room_categories (
  `id` INT NOT NULL,
  `name` VARCHAR(100) NOT NULL,
  `price` DOUBLE NOT NULL,
  PRIMARY KEY(`id`),
  CONSTRAINT `u_room_categories` UNIQUE (`name`))
  ENGINE=InnoDB;
```

Here we added possibility to modify records with cascade because these tables contain foreign keys to other tables and it is convenient to perform deleting or updating operations and populate the changes across child and parent tables together.

```
CREATE TABLE IF NOT EXISTS rooms (
  `id` INT NOT NULL,
  `number` INT NOT NULL CHECK (`number` > 0),
  `room_category_id` INT NOT NULL,
  PRIMARY KEY(`id`),
  CONSTRAINT `fk_rooms_room_categories` FOREIGN KEY (`room_category_id`) REFERENCES  room_categories (`id`)
  ON DELETE CASCADE ON UPDATE CASCADE)
  ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS clients (
  `id` BIGINT NOT NULL,
  `first_name` VARCHAR(100),
  `last_name` VARCHAR(100),
  `country` VARCHAR(100),
  PRIMARY KEY(`id`))
  ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS reservations (
  `id` BIGINT NOT NULL,
  `date_from` DATE NOT NULL,
  `date_to` DATE NOT NULL,
  `room_id` INT NOT NULL,
  `client_id` BIGINT NOT NULL,
  PRIMARY KEY(`id`),
  CONSTRAINT `fk_reservations_rooms` FOREIGN KEY (`room_id`) REFERENCES  rooms (`id`),
```

```
  CONSTRAINT `fk_reservations_clients` FOREIGN KEY (`client_id`) REFERENCES  clients (`id`)
  ON DELETE CASCADE ON UPDATE CASCADE)
  ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS payments (
  `id` BIGINT NOT NULL,
  `transaction_date` DATETIME NOT NULL,
  `reservation_id` BIGINT NOT NULL,
  `payment_size` DOUBLE NOT NULL,
  `payment_type_id` INT NOT NULL,
  PRIMARY KEY(`id`),
  CONSTRAINT `fk_payments_reservations` FOREIGN KEY (`reservation_id`) REFERENCES  reservations (`id`),
  CONSTRAINT `fk_payments_payment_types` FOREIGN KEY (`payment_type_id`) REFERENCES  payment_types (`id`)
  ON DELETE CASCADE ON UPDATE CASCADE)
  ENGINE=InnoDB;
```

By modelling the database was used two triggers for checking the price for room category so that price can not be negative.
The first triggering is performed both in case of updating or inserting a record into the table.

```
DELIMITER $
CREATE TRIGGER `chk_room_categories_price_insert` BEFORE INSERT ON `room_categories`
FOR EACH ROW
BEGIN
IF NEW.price < 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Price cannot be negative';
END IF;
END$
DELIMITER ;

DELIMITER $
CREATE TRIGGER `chk_room_categories_price_update` BEFORE UPDATE ON `room_categories`
FOR EACH ROW
BEGIN
IF NEW.price < 0
THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Price cannot be negative';
END IF;
END$
DELIMITER ;
```

The second triggering is performed when updating or inserting a record into the payment table to prevent size of the payment not to be less then the price of the room category.

```
DELIMITER $
CREATE TRIGGER `chk_payments_payment_size_insert` BEFORE INSERT ON `payments`
FOR EACH ROW
BEGIN
DECLARE room_price DOUBLE;
SET room_price = (SELECT rc.price FROM room_categories rc
  INNER JOIN rooms r ON r.room_category_id = rc.id
  INNER JOIN reservations res ON res.room_id = r.id
  WHERE res.id = NEW.reservation_id);
IF NEW.`payment_size` < room_price
THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Payment size cannot be less then room price';
END IF;
END$
DELIMITER ;

DELIMITER $
CREATE TRIGGER `chk_payments_payment_size_update` BEFORE UPDATE ON `payments`
FOR EACH ROW
BEGIN
```

```
DECLARE room_price DOUBLE;
SET room_price = (SELECT rc.price FROM room_categories rc
  INNER JOIN rooms r ON r.room_category_id = rc.id
  INNER JOIN reservations res ON res.room_id = r.id
  WHERE res.id = OLD.reservation_id);
IF NEW.`payment_size` < room_price
THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Payment size cannot be less then room price';
END IF;
END$
DELIMITER ;
```

Insertion sample data into tables.

```
INSERT INTO payment_types (`id`, `name`) VALUES
(1, 'Credit card'),
(2, 'Cash'),
(3, 'Bank transfer'),
(4, 'Voucher');

INSERT INTO room_categories (`id`, `name`, `price`) VALUES
(1, 'President Suite', 3000.00),
(2, 'Penthouse', 1750.50),
(3, 'Single', 200.74),
(4, 'Twin', 325.99),
(5, 'Double', 300.45),
(6, 'Lux Appartment',700.45);

INSERT INTO rooms (`id`, `number`, `room_category_id`) VALUES
(1, 501, 1),
(2, 601, 2),
(3, 401, 5),
(4, 301, 3),
(5, 302, 3),
(6, 303, 3),
(7, 304, 3),
(8, 305, 4),
(9, 201, 4),
(10, 202, 4),
(11, 402, 5);

INSERT INTO clients (`id`, `first_name`, `last_name`, `country`) VALUES
(1, 'Manuel', 'Neuer', 'Germany'),
(2, 'Joshua', 'Kimmich', 'Germany'),
(3, 'Thomas', 'Muller', 'Germany'),
(4, 'Jerome', 'Boateng', 'Germany'),
(5, 'Mats', 'Hummels', 'Germany'),
(6, 'Thiago', NULL, 'Spain'),
(7, 'Franck', 'Ribery', 'France'),
(8, 'Javi', 'Martinez', 'Spain'),
(9, 'Robert', 'Lewandowski', 'Poland'),
(10, 'Arjen', 'Robben', 'Netherland'),
(11, 'James', 'Rodriguez', 'Columbia'),
(12, 'Kingsley', 'Coman', 'France'),
(13, 'Rafinha', NULL, 'Spain'),
(14, 'David', 'Alaba', 'Austria');

INSERT INTO reservations (`id`, `date_from`, `date_to`, `room_id`, `client_id`) VALUES
(1, '2018-07-05','2018-07-11', 5, 11),
(2, '2018-07-08', '2018-07-15', 6, 9),
(3, '2018-08-12', '2018-08-15', 4, 2),
(4, '2018-08-24', '2018-08-30', 5, 10),
```

```
(5, '2018-07-07', '2018-07-14', 8, 2),
(6, '2018-07-01', '2018-07-03', 1, 1),
(7, '2018-07-08', '2018-07-12', 11, 7),
(8, '2018-08-09', '2018-08-16', 5, 3),
(9, '2018-05-01', '2018-05-03', 1, 9),
(10, '2018-06-04', '2018-06-28', 2, 9);

INSERT INTO payments (`id`, `transaction_date`, `reservation_id`, `payment_size`, `payment_type_id`) VALUES
(1, '2018-07-01 03:14:07', 1,  210.00, 1),
(2, '2018-06-30 12:24:23', 2, 250.50, 3),
(3, '2018-08-11 13:35:17', 3, 220.10, 1),
(4, '2018-08-14 10:54:46', 4, 300.20, 1),
(5, '2018-07-05 19:48:33', 5, 354.90, 2),
(6, '2018-06-25 10:31:01', 6, 3300.00, 2),
(7, '2018-12-20 12:00:00', 7, 310.20, 4),
(8, '2018-07-04 05:06:07', 8, 203.00, 2),
(9, '2018-02-14 13:25:19', 9, 3003.00, 4),
(10, '2018-05-03 15:26:24', 10, 1800.00, 3);
```

Query with multi JOIN

Displays which rooms are the most popular and benefit from their reservation per given period

```
SELECT rc.name, rc.price, COUNT(rc.name) AS 'reservation_count'
FROM room_categories rc JOIN rooms r ON r.room_category_id = rc.id
INNER JOIN reservations res ON res.room_id = r.id
WHERE res.date_from >= '2018-07-01'
GROUP BY rc.name
ORDER BY 'reservation_count'  DESC;
```

Displays which rooms are in the idle state as well as costs per each room

```
SELECT r.number, rc.price
FROM rooms r
LEFT OUTER JOIN reservations res ON res.room_id = r.id
INNER JOIN room_categories rc ON r.room_category_id = rc.id
WHERE res.room_id IS NULL;
```

Query with subquery

Displays in which dates payment type Voucher is popular

```
SELECT p.transaction_date FROM payments p
WHERE p.payment_type_id IN
(SELECT pt.id FROM payment_types pt WHERE pt.name = 'Voucher');
```

Displays which room categories are more popular among tourists of specific country

```
SELECT rc.name FROM room_categories rc WHERE rc.id IN
(SELECT r.room_category_id FROM rooms r
INNER JOIN reservations res ON r.id = res.room_id
INNER JOIN clients c ON c.id = res.client_id WHERE c.country = 'Germany');
```

Query with GROUP BY and HAVING

Shows that the maximum money transactions were performed with a cash

```
SELECT MAX(`payment_size`) AS `max_payment`,
pt.name FROM payments p
INNER JOIN payment_types pt ON p.payment_type_id = pt.id
WHERE CAST(p.transaction_date AS DATE) > '2018-01-01'
GROUP BY pt.name
```

```
HAVING `max_payment`> 200
ORDER BY `max_payment` DESC;
```

Query with aggregate function

Displays contact data of guests whose amount of stays is less than normal frequency

```
CREATE VIEW `reservation_amount_clients` AS
SELECT c.first_name, c.last_name, COUNT(res.client_id) AS reservations_amount
FROM clients c INNER JOIN reservations res
ON c.id = res.client_id GROUP BY res.client_id;

SELECT c.first_name, c.last_name, reservations_amount
FROM reservation_amount_clients c
WHERE reservations_amount < (SELECT AVG(reservations_amount) FROM reservation_amount_clients);
```

Displays how big is income from each type of room categories and analyzesif it is enough or not

```
SELECT SUM(DATEDIFF(res.date_to, res.date_from) * rc.price) AS `common_income`,
rc.name AS `room_category_name`,
CASE WHEN SUM(DATEDIFF(res.date_to, res.date_from) * rc.price) > 10000 THEN "excellent"
WHEN SUM(DATEDIFF(res.date_to, res.date_from) * rc.price) <= 10000
AND SUM(DATEDIFF(res.date_to, res.date_from) * rc.price) > 5000
THEN "satisfied" ELSE "not enough"
END AS comments
FROM reservations res
INNER JOIN rooms r ON res.room_id = r.id
INNER JOIN room_categories rc ON rc.id = r.room_category_id
GROUP BY  rc.name;
```

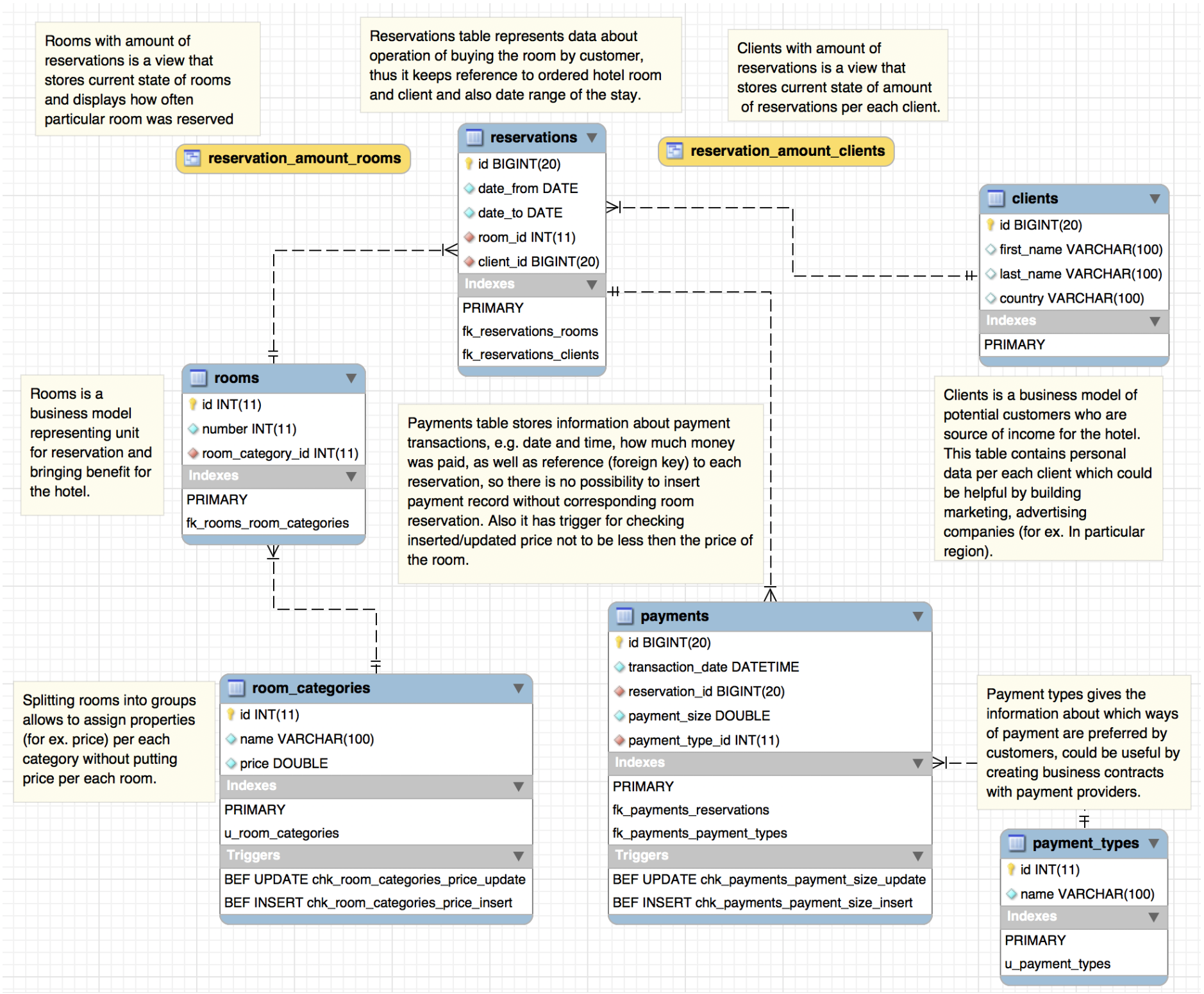
Query with UNION

Displays the most popular/unpopular, the most expensive/cheapest rooms in the hotel, so we could highlight which rooms brings most/less of the benfit, more/less preferable.

```
CREATE VIEW `reservation_amount_rooms` AS
SELECT r.number, COUNT(res.room_id) AS `reservations_amount`, rc.price
FROM reservations res INNER JOIN rooms r ON r.id = res.room_id
INNER JOIN room_categories rc ON rc.id = r.room_category_id
GROUP BY res.room_id;

(SELECT rar.number, 'most popular room'
FROM reservation_amount_rooms rar
WHERE rar.reservations_amount = (SELECT MAX(rar.reservations_amount) FROM reservation_amount_rooms rar) LIMIT
1)
UNION
(SELECT rar.number, 'most unpopular room'
FROM reservation_amount_rooms rar
WHERE rar.reservations_amount = (SELECT MIN(rar.reservations_amount) FROM reservation_amount_rooms rar) LIMIT 1)
UNION
(SELECT rar.number, 'most expensive room'
FROM reservation_amount_rooms rar
WHERE rar.price= (SELECT MAX(rar.price) FROM reservation_amount_rooms rar) LIMIT 1)
UNION
(SELECT rar.number, 'cheapest room'
FROM reservation_amount_rooms rar
WHERE rar.price = (SELECT MIN(rar.price) FROM reservation_amount_rooms rar) LIMIT 1);
```

Bellow is the E/R diagram of database model


Rooms with amount of reservations is a view that stores current state of rooms and displays how often particular room was reserved
Reservations table represents data about operation of buying the room by customer, thus it keeps reference to ordered hotel room and client and also date range of the stay.
Clients with amount of reservations is a view that stores current state of amount of reservations per each client.
reservation_amount_rooms
reservations
id BIGINT(20)
date_from DATE
date_to DATE
room_id INT(11)
client_id BIGINT(20)
Indexes
PRIMARY
fk_reservations_rooms
fk_reservations_clients
reservation_amount_clients
clients
id BIGINT(20)
first_name VARCHAR(100)
last_name VARCHAR(100)
country VARCHAR(100)
Indexes
PRIMARY
rooms
id INT(11)
number INT(11)
room_category_id INT(11)
Indexes
PRIMARY
fk_rooms_room_categories
Rooms is a business model representing unit for reservation and bringing benefit for the hotel.
Payments table stores information about payment transactions, e.g. date and time, how much money was paid, as well as reference (foreign key) to each reservation, so there is no possibility to insert payment record without corresponding room reservation. Also it has trigger for checking inserted/updated price not to be less then the price of the room.
Clients is a business model of potential customers who are source of income for the hotel. This table contains personal data per each client which could be helpful by building marketing, advertising companies (for ex. In particular region).
payments
id BIGINT(20)
transaction_date DATETIME
reservation_id BIGINT(20)
payment_size DOUBLE
payment_type_id INT(11)
Indexes
PRIMARY
fk_payments_reservations
fk_payments_payment_types
Triggers
BEF UPDATE chk_payments_payment_size_update
BEF INSERT chk_payments_payment_size_insert
Payment types gives the information about which ways of payment are preferred by customers, could be useful by creating business contracts with payment providers.
payment_types
id INT(11)
name VARCHAR(100)
Indexes
PRIMARY
u_payment_types
Splitting rooms into groups allows to assign properties (for ex. price) per each category without putting price per each room.
room_categories
id INT(11)
name VARCHAR(100)
price DOUBLE
Indexes
PRIMARY
u_room_categories
Triggers
BEF UPDATE chk_room_categories_price_update
BEF INSERT chk_room_categories_price_insert