**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**

