REPORT

1. INTRODUCTION:
2. Introduction and database description:

The database was compiled on the basis of the hotel industry, all the data is fake and was filled with mackaroo.com . Information about hotel chains has been collected, which is grouped by country and city and has a stable database structure. The main purpose of our database is to optimize the system of hotel chains and their payment method.

1. What functions should the system perform? For example, inventory control, billing, ordering, etc.

The hotel as a segment of the service sector. The hotel business represents the services of hotels, inns and guest houses, as well as apartment rentals and, in general, refers to hospitality and catering services.

1. Who are the end users? Remember that the DBA is NOT an end user:
2. Casual End Users – who are managers in the given hotel
3. Naïve end users: It can be person who manages the payment system, reservation clerks
4. Specialised users and Standalone users those who maintain personal information of each customer

IDK

1. How will data obsolescence be handled?

Data obsolescence is an increase in data loss due to obligations to access digital assets due to required hardware or software. As mentioned earlier, our database has a stable data structure that prevents data obsolescence. Normalization reduces the organizational complexity of archival institutions by reducing the number of similar file types by conversion. All our tables are in BCNF form, which gives a guarantee. We also use triggers and transactions. Durability transaction gives guarantee that once a transaction is committed, it will remain in the system – even if there’s a system crash immediately following the transaction.

Steps:

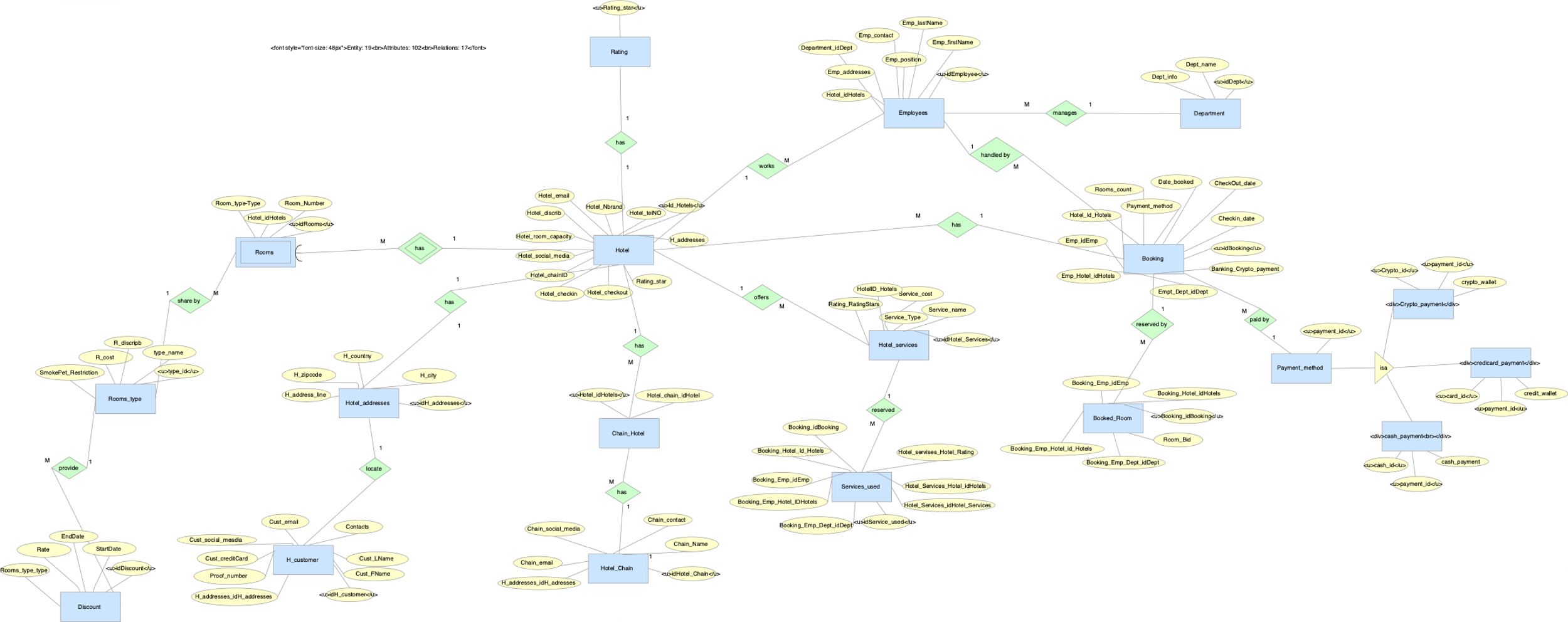
* Create a workable taxonomy for your data. Work with key stakeholders to establish a uniform set of definitions, labels and groupings, so you can clearly understand what data you have.
* Establish a policy and a set of best practices for handling ROT data. For example, establish procedures for purging obsolete records and trivial data.

1. Idea for project.

The main idea of the project is to create hotel management system for hotel chain. To optimize the whole system, to provide billing, reservation, control system and automize it. Its been improved by adding crypto paying system. So, it will solve some data loss and reduncdacy problems .

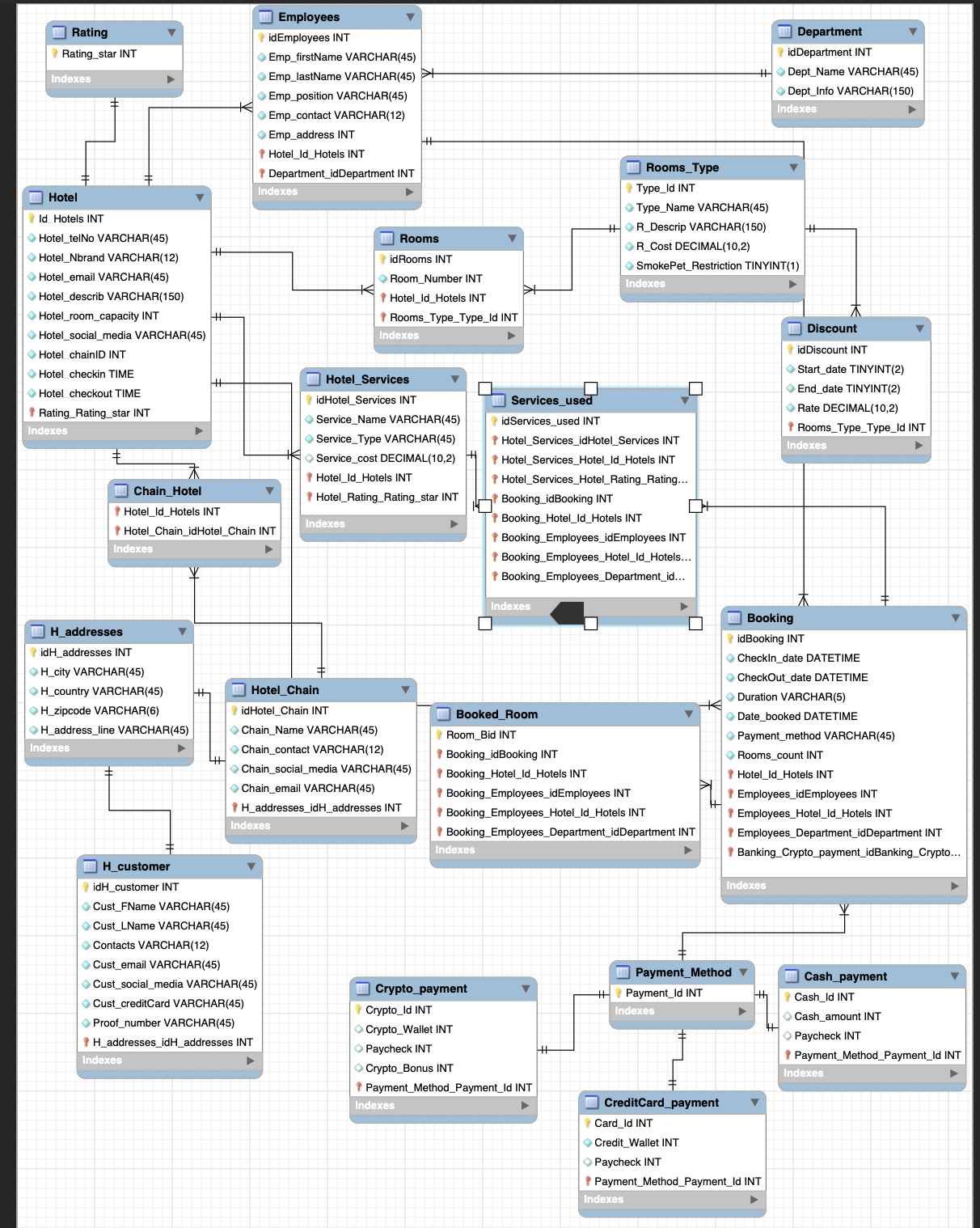
1. ENTITY RELATIONSHIP DESIGN:

ERD:

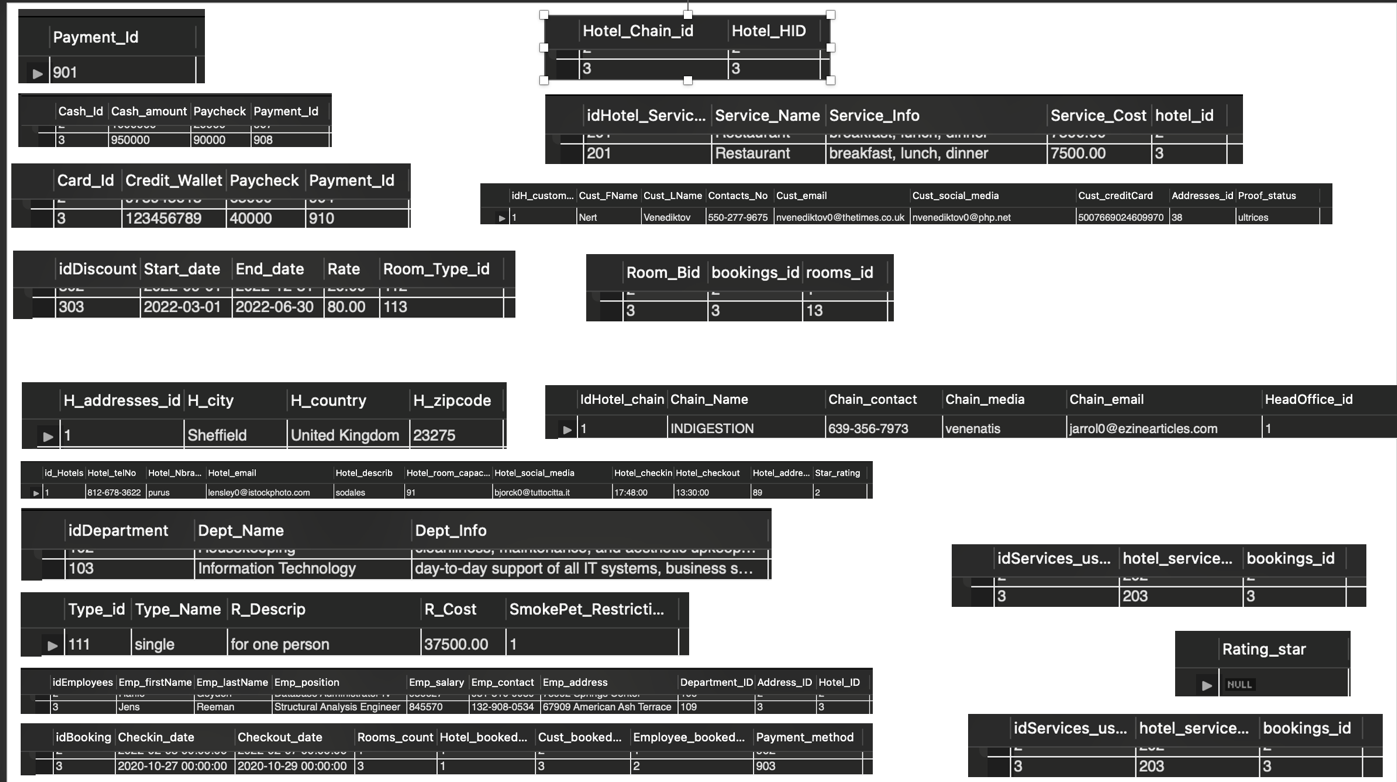


Schema design

:



Relational schema:



1. NORMALIZATION:
2. H\_addresses:

H\_addresses\_id -> H\_zipcode, H\_country, H\_city

1. Hotel\_Chain:

idHotel\_chain -> Chain\_Name, Chain\_contact, Chain\_media, Chain\_email, HeadOffice\_id

HeadOffice\_id -> Chain\_name, chain\_contact, chain\_media, chain\_email

1. Hotel:

idHotels -> Hotel\_telNo, Hotel\_NBrand, Hotel\_email, Hotel\_describ, Hotel\_room\_capacity, hotel\_checkin, Hotel\_checkout, hotel\_address, star\_rating

1. Rooms\_type:

Type\_id -> Type\_name, R\_descrip, R\_cost, SmokePet\_Restriction

Type\_id, type\_name -> R\_cost, SmokePet\_Restriction

1. Rooms:

idRooms -> Room\_Number, TypeRoom\_id, Hotel\_id

Hotel\_id, idRooms -> Room\_Number, TypeRoom\_id

Hotel\_id -> idRooms

1. Customer:

idH\_customer -> cust\_FName, cust\_LName, Contacts\_No, Cust\_email, Cust\_social\_media, cust\_creditCard, addresses\_id, proof\_status

idH\_Customer, Addresses-id -> proof\_status

1. Department:

idDepartment -> Dept\_name, Dept\_info

1. Employees:

idEmployees -> Emp\_firstName, Emp\_lastName, Emp\_position, Emp\_salary, Emp\_contact, Emp\_address, Department\_ID, Address\_ID, Hotel\_ID

Department\_Id -> idEmployees

Hotel\_id, address\_Id -> department\_id

1. Crypto\_payment:

Crypto\_id -> crypto\_wallet, paycheck, payment\_id, crypto\_bonus, paycheck

1. Card\_payment:

Card\_id -> credit\_amount, paycheck, payment\_id, paycheck

1. Cash\_payment:

Cash\_id -> cash\_amount, paycheck, payment\_id, paycheck

1. Booking:

idBooking -> checkin\_date, checkout\_date, duration, rooms\_count, hotel\_booked\_id, cust\_booked\_id, employee\_booked\_id, payment\_method

1. Discount :

idDiscount -> start\_date, end\_date, rate, room\_type\_id

1. Rooms\_booking:

Room\_Bid -> bookings\_id, rooms\_id

Room\_Bid, rooms\_id -> bookings\_id

Bookings\_id -> rooms\_id, room\_Bid

1. Hotel\_service:

idHotel\_Service -> service\_name, service\_info, service\_cost, hotel\_id

hotel\_id -> idHotel\_services

1. Services\_used:

idServices\_used -> hotel\_service\_id, bookings\_id

bookings\_id -> idServices\_used, hotel\_service\_id

1. PHYSICAL DESIGN:

19 tables

102 entities

7 tables with 100 rows

All data fake, we use mockaroo.com

1. QUERY:
2. LEFT JOIN, count, group by : total number of employees in each department:

select d.Dept\_name as departmentName,

count(e.Department\_ID) as totalEmployees

from department d left join

employees e

on d.idDepartment = e.Department\_ID

group by d.Dept\_Name;

R.A:

join := (department d)⋈ (d.idDepartment = e.Department\_ID) (employees e)

agg := ɣ (d.dept\_name, count(e.department\_id));

print := (d.dept\_name, agg) (join)

1. TRIPLE INNER JOIN :

Specify the price and category of all rooms in the hotel "Purus".

select rooms\_type.R\_cost, rooms\_type.type\_name, hotel.hotel\_Nbrand from rooms\_type

inner join rooms on rooms\_type.type\_id = rooms.TypeRoom\_id

inner join hotel on rooms.hotel\_id = hotel.id\_Hotels

where hotel.hotel\_Nbrand like 'purus';

R.A:

join := (rooms\_type ) ⋈ (rooms\_type.type\_id = rooms.TypeRoom\_id) (rooms)

join2 := (hotel) ⋈ (rooms.hotel\_id = hotel.id\_Hotels)(join)

select := σ (hotel.hotel\_Nbrand like 'purus') (join)

print := (rooms\_type.R\_cost, rooms\_type.type\_name, hotel.hotel\_Nbrand) (select)

1. view with join:

list of guests staying in hotels in kazakhstan

create view guests\_list as

select c.cust\_Fname, c.cust\_LName, h.h\_country from h\_customer as c

join h\_addresses as h on c.addresses\_id = h.h\_addresses\_id

where h.h\_country like 'Kazakhstan';

select \* from guests\_list;

R.A:

join := (h\_customer as c) ⋈ (c.addresses\_id = h.h\_addresses\_id) (h\_addresses)

select := σ (h.h\_country like 'Kazakhstan') (join)

print := (c.cust\_Fname, c.cust\_LName, h.h\_country) (select)

1. VIEW with GROUP BY, COUNT, ORDER BY:

create view room\_rating as

SELECT count(hotel\_NBrand) as totalRoom, star\_rating

FROM hotel

GROUP BY star\_rating

HAVING count(hotel\_NBrand) > 5

ORDER BY star\_rating DESC;

select \* from room\_rating;

R.A:

Agg := ɣ (count(hotel\_NBrand), star\_rating)

Order := τ (star\_rating DESC)

Select = σ (agg > 5) (hotel)

Print := (agg, star\_rating) (select)

1. DOUBLE SUBQUERY + aggregations:

output the number of countries in which the number of rooms in one hotel is more than 100

SELECT Count(\*), h\_country

FROM H\_addresses

WHERE h\_addresses\_id in (SELECT hotel\_address

FROM hotel

WHERE EXISTS

(SELECT Count(\*) AS RoomCount

FROM hotel

WHERE Hotel\_room\_capacity > 100))

group by h\_country;

R.A:

Sub1 := σ (Hotel\_room\_capacity > 100) ( (ɣ (count(\*))) (hotel))

Sub2 := σ (sub1) ( (hotel\_address) (hotel))

Print := σ (sub2) ( (ɣ(count(\*), h\_country)) (H\_addresses))

1. except, subquery, join with 4 tables, group by:

available room with pets/smoke in 5 star hotel in Charlton

select rooms.Room\_Number, rooms\_type.Type\_Name, rooms\_type.SmokePet\_Restriction, hotel.Star\_rating, rooms\_type.R\_cost from rooms\_type, rooms, hotel, h\_addresses

where rooms\_type.Type\_id = rooms.TypeRoom\_id and rooms.Hotel\_id = Hotel\_id and hotel.Hotel\_address = h\_addresses.H\_addresses\_id

and rooms\_type.SmokePet\_Restriction = 1 and hotel.Star\_rating = 5 and h\_addresses.H\_city = 'Charlton' and rooms.idRooms in (

select idrooms from rooms

except

select room\_Bid from booked\_room )

group by room\_number;

R.A:

Joins := rooms\_type ⋈ (rooms\_type.Type\_id = rooms.TypeRoom\_id) rooms ⋈ (rooms.Hotel\_id = Hotel\_id) hotel ⋈ (hotel.Hotel\_address = h\_addresses.H\_addresses\_id) h\_addresses

Sub := (idrooms) (rooms) - (room\_Bid) (booked\_room)

Select := σ (rooms\_type.SmokePet\_Restriction = 1, hotel.Star\_rating = 5, h\_addresses.H\_city = 'Charlton', sub )

Print := ɣ(room\_number)(select)

1. exists, update, subquery:

new year sale and all room type cost equal to discounted amount

UPDATE rooms\_type

SET r\_cost = r\_cost - (SELECT rate

FROM discount

WHERE rooms\_type.type\_id = discount.room\_type\_id)

WHERE EXISTS (SELECT rate

FROM discount

WHERE rooms\_type.type\_id = discount.room\_type\_id);

select type\_name, r\_cost from rooms\_type;

1. aggregations + join :

show sum, avg, min, max price of services in the hotel 'nulla'

select hotel.hotel\_nbrand, sum(service\_cost), avg(service\_cost), min(Service\_Cost), max(Service\_Cost)

from hotel\_services, hotel

where hotel\_services.hotel\_id = hotel.id\_Hotels and hotel.Hotel\_Nbrand like 'nulla';

R.A:

Agg := ɣ (sum(service\_cost), avg(service\_cost), min(Service\_Cost), max(Service\_Cost))

Join := hotel\_service ⋈ (hotel\_services.hotel\_id = hotel.id\_Hotels) hotel

Select := σ (hotel.Hotel\_Nbrand like 'nulla')(join)

Print := (hotel.hotel\_nbrand, agg)(select)

1. union, order by:

during evacuation, a list of data of all people in the hotel, grouped by addresses

select h\_customer.Cust\_FName as first\_name, h\_customer.Cust\_LName as last\_name, h\_customer.Contacts\_No as contacts, addresses\_id as address

from h\_customer

union

select employees.Emp\_firstName, employees.Emp\_lastName, employees.Emp\_contact, address\_id from employees

order by address;

R.A:

R1 := (h\_customer.Cust\_FName, h\_customer.Cust\_LName, h\_customer.Contacts\_No, addresses\_id)(h\_customer)

R2:= (employees.Emp\_firstName, employees.Emp\_lastName, employees.Emp\_contact, address\_id)( employees)

Print:= (ɣ(address))(r1 U r2)

1. union all, subquery, in, intersect, order by:

output the data of employees and employees who have a same address id

SELECT Cust\_FName as first\_name,Cust\_LName as last\_name, Contacts\_No as contacts, address

FROM

(

SELECT h\_customer.Cust\_FName, h\_customer.Cust\_LName, h\_customer.Contacts\_No, Addresses\_id as address

FROM h\_customer

UNION ALL

SELECT employees.Emp\_firstName, employees.Emp\_lastName, employees.Emp\_contact, address\_id

FROM employees

)

derived

WHERE address IN (

SELECT address\_id FROM employees

INTERSECT

SELECT addresses\_id

FROM h\_customer)

order by (address);

R.A:

r1 :=  (h\_customer.Cust\_FName, h\_customer.Cust\_LName, h\_customer.Contacts\_No, Addresses\_id )(h\_customer)

r2:= (employees.Emp\_firstName, employees.Emp\_lastName, employees.Emp\_contact, address\_id)(employees)

union := r1 U r2

Sub := ( (address\_id) (employees)) n ((addresses\_id)(h\_customer))

Select := σ (address IN sub)(union)

Print := (Cust\_FName,Cust\_LName, Contacts\_No, address)(select)

1. aggregations + subquery + group by + any + having:

наименование отеля в котором средняя зарплата сотрудников больше 500к

select hotel.hotel\_NBrand, avg(hproject.employees.emp\_salary) from hotel, hproject.employees

where id\_hotels = ANY

(select hotel\_id

from hproject.employees

having avg(emp\_salary) >= 500000)

group by (Hotel\_Nbrand);

R.A:

Sub := (hotel\_id)((σ (ɣ(avg(emp\_salary) >= 500000))( hproject.employees))

Select := σ (id\_hotels = (sub))(hproject.employees)

Print :=  (hotel.hotel\_NBrand, ɣ (avg(hproject.employees.emp\_salary)))(select)

1. aggregation + group by + join + order by:

show rating statistics hotels having more than 200 rooms

select h\_addresses.h\_country , avg(hotel.star\_rating) as avg\_rating

from hotel, h\_addresses

where hotel.hotel\_address = h\_addresses.h\_addresses\_id

and hotel.hotel\_room\_capacity >= 200

group by (h\_addresses.h\_country)

R.A:

join := hotel ⋈ (hotel.hotel\_address = h\_addresses.h\_addresses\_id) h\_addresses

select := σ (hotel.hotel\_room\_capacity >= 200) (join)

agg := ɣ( avg(hotel.star\_rating), h\_addresses.h\_country)

print := π(h\_addresses.h\_country, agg) (select)

1. join, count, group by, order by:

display the names of employees and the number of their reservations made by them

select employees.emp\_firstname, count(booking.idBooking) as totalBooking

from employees, booking

where booking.employee\_booked\_id = employees.idEmployees

group by (employees.emp\_firstname)

order by (totalBooking) DESC;

R.A:

join := employees ⋈ (booking.employee\_booked\_id = employees.idEmployees) booking

agg := ɣ(employees.emp\_firstname, totalBooking, count(booking.idBooking)(booking))

print :=  (employees.emp\_firstname, agg)(join)

1. alter table:

alter table booking

drop column duration;

select \* from booking

1. null:

delete from booking

where idBooking = null;

1. rename:

RENAME TABLE hotel TO hotels;

RENAME TABLE hotels TO hotel;

1. all, joins:

show all customers name who used spa/hammam/massage service

select h\_customer.cust\_fname from h\_customer, booking, services\_used

where services\_used.bookings\_id = booking.idBooking and booking.cust\_booked\_id = h\_customer.idH\_customer

and services\_used.hotel\_service\_id = ALL (

select idHotel\_services

from hotel\_services

where service\_name like 'Spa / Hammam / Massage');

R.A:

Sub := (idHotel\_services) (σ(service\_name like 'Spa / Hammam / Massage')(hotel\_services))

Join := h\_customer ⋈ (booking.cust\_booked\_id = h\_customer.idH\_customer) booking ⋈ (services\_used.bookings\_id = booking.idBooking) services\_used

Select := σ(services\_used.hotel\_service\_id = sub) (join)

Print := (h\_customer.cust\_fname)(select)

1. all, aggregations, group by, order by:

finds all employees whose salaries are greater than the lowest salary of every department:

select emp\_firstName, emp\_lastname, emp\_salary

from employees

where emp\_salary >= ALL (

select min(emp\_salary) from employees

group by department\_id)

order by (emp\_firstName);

R.A:

sub := (ɣ (min(emp\_salary), department\_id))(employee)

select := σ (emp\_salary >=ALL sub)(employee)

print := (emp\_firstName, emp\_lastname, emp\_salary) (select)

1. subquery in + join :

List the rate for a room ‘1’

SELECT ROUND((r\_type.R\_Cost - ((r\_dis.Rate \* r\_type.R\_Cost)/100)), 2) AS 'Room Rate'

FROM Discount r\_dis JOIN Rooms\_Type r\_type

ON r\_dis.Room\_Type\_id = r\_type.Type\_id

WHERE r\_type.Type\_id

IN ( Select TypeRoom\_id from Rooms where idRooms = 1)

R.A:

Sub := (TypeRoom\_id)( σ (idRooms = 1 )(rooms))

Join := discount r\_dis ⋈ (r\_dis.Room\_Type\_id = r\_type.Type\_id) rooms\_type r\_type

Select := σ (r\_type.type\_id IN sub)(join)

Print := (ROUND((r\_type.R\_Cost - ((r\_dis.Rate \* r\_type.R\_Cost)/100)), 2)) (select)

20. Function:

-- -----------------------------------------------------

-- Queries HProject. Function(How many rooms are available in a given hotel?)

-- -----------------------------------------------------

SELECT h.Hotel\_room\_capacity - SUM(b.Rooms\_count) AS 'Available Rooms'

FROM Booking b JOIN Hotel h

ON b.Hotel\_booked\_id = h.id\_Hotels

WHERE Checkin\_date LIKE '2021-12-25%' AND Hotel\_booked\_id = 1

R.A:

π h . hotel\_room\_capacity - SUM (rooms\_count)

σ checkin\_date LIKE "2021-12-25%" AND hotel\_booked\_id = 1

(ρ b booking ⋈ b . hotel\_booked\_id = h . id\_hotels

ρ h hotel)

1. Trigger: Insert into

-- -----------------------------------------------------

-- Queries HProject. Trigger №1(Insert)

-- -----------------------------------------------------

SET sql\_notes = 0;

-- DROP TABLE IF EXISTS HProject.Bookings\_Audit ;

-- CREATE TABLE IF NOT EXISTS HProject.Bookings\_Audit(

-- idBooking INT NOT NULL,

-- Checkin\_date DATETIME NULL,

-- Checkout\_date DATETIME NULL,

-- Duration VARCHAR(5) NULL,

-- Rooms\_count INT NULL,

-- Hotel\_booked\_id INT NOT NULL,

-- Cust\_booked\_id INT NOT NULL,

-- Employee\_booked\_id INT NOT NULL,

-- Payment\_method INT NOT NULL

-- );

-- DELETE FROM Booking

-- WHERE idBooking = 19;

-- Select \* from Bookings\_Audit;

-- DROP TRIGGER IF EXISTS bookings\_after\_insert;

-- DELIMITER //

-- CREATE TRIGGER Bookings\_after\_insert

-- AFTER INSERT ON Booking

-- FOR EACH ROW

-- BEGIN

-- INSERT INTO Bookings\_Audit VALUES

-- (NEW.idBooking, NEW.Checkin\_date, NEW.Checkout\_date, DATEDIFF(NEW.Checkout\_date, NEW.Checkin\_date), NEW.Rooms\_count, NEW.Hotel\_booked\_id, NEW.Cust\_booked\_id, NEW.Employee\_booked\_id, NEW.Payment\_method);

-- END//

-- DELIMITER ;

-- -----------------------------------------------------

-- Queries HProject. Trigger №1(Update)

-- -----------------------------------------------------

-- DROP TRIGGER IF EXISTS Credit\_Card\_Update;

SET sql\_notes = 0;

-- CREATE TABLE IF NOT EXISTS HProject.CreditCard\_payment\_updated (

-- Card\_Id INT NOT NULL,

-- Credit\_Wallet INT,

-- Paycheck INT,

-- Payment\_Id INT NOT NULL

-- );

-- DELIMITER $$

-- CREATE TRIGGER Credit\_Card\_Update

-- AFTER UPDATE

-- ON creditCard\_payment FOR EACH ROW

-- BEGIN

-- Insert into CreditCard\_payment\_updated Values (old.Card\_Id, old.Credit\_Wallet, new.Paycheck, old.Payment\_Id);

-- END$$

-- DELIMITER ;

-- Update creditCard\_payment set paycheck = 40000 where Payment\_Id = 902;

-- Select \* from CreditCard\_payment\_updated;

SET sql\_notes = 1;

-- -----------------------------------------------------

-- Queries HProject. Trigger №1(Delete)

-- -----------------------------------------------------

SET sql\_notes = 0;

-- DROP TABLE IF EXISTS HProject.Bookings\_Audit\_Deleted ;

-- CREATE TABLE IF NOT EXISTS HProject.Bookings\_Audit\_Deleted(

-- idBooking INT NOT NULL,

-- Checkin\_date DATETIME NULL,

-- Checkout\_date DATETIME NULL,

-- Duration VARCHAR(5) NULL,

-- Rooms\_count INT NULL,

-- Hotel\_booked\_id INT NOT NULL,

-- Cust\_booked\_id INT NOT NULL,

-- Employee\_booked\_id INT NOT NULL,

-- Payment\_method INT NOT NULL

-- );

-- DROP TRIGGER IF EXISTS bookings\_after\_delete;

-- DELIMITER //

-- CREATE TRIGGER Bookings\_after\_delete

-- AFTER DELETE ON Booking

-- FOR EACH ROW

-- BEGIN

-- INSERT INTO Bookings\_Audit\_Deleted VALUES

-- (Old.idBooking, Old.Checkin\_date, Old.Checkout\_date, Null, Old.Rooms\_count, Old.Hotel\_booked\_id, Old.Cust\_booked\_id, Old.Employee\_booked\_id, Old.Payment\_method);

-- END//

-- DELIMITER ;

-- DELETE FROM Booking

-- WHERE idBooking = 19;

SET sql\_notes = 1;

1. TRANSACTION AND INDEXES:

Transaction:

-- -----------------------------------------------------

-- Queries HProject. Transaction (Add CryptoBonus if cryptoPayBonus less than 40)

-- -----------------------------------------------------

SELECT \* FROM crypto\_payment WHERE crypto\_Bonus < 50;

START TRANSACTION;

UPDATE crypto\_payment SET crypto\_Bonus = (crypto\_Bonus + 50) WHERE crypto\_Bonus < 50;

SELECT \* FROM crypto\_payment WHERE crypto\_Bonus < 50;

ROLLBACK;

SELECT \* FROM crypto\_payment WHERE crypto\_Bonus < 50;

Indexes:

CREATE INDEX `Hotel\_services\_has\_bookings\_idx` ON `HProject`.`Services\_used` (`bookings\_id` ASC);

CREATE INDEX `Hotel\_services\_has\_service\_idx` ON `HProject`.`Services\_used` (`hotel\_service\_id` ASC);

CREATE INDEX `Hotel\_services\_idx` ON `HProject`.`Hotel\_Services` (`hotel\_id` ASC)

CREATE INDEX `Rooms\_booked\_idx` ON `HProject`.`Booked\_Room` (`bookings\_id` ASC);

CREATE INDEX `Rooms\_booked\_rooms\_idx` ON `HProject`.`Booked\_Room` (`rooms\_id` ASC);

CREATE INDEX `Room\_Discopunt\_type\_idx` ON `HProject`.`Discount` (`Room\_Type\_id` ASC);

CREATE INDEX `Hotel\_Has\_Chain\_idx` ON `HProject`.`Chain\_has\_hotel` (`Hotel\_Chain\_id` ASC);

CREATE INDEX `Hotel\_Chain\_Hotel` ON `HProject`.`Chain\_has\_hotel` (`Hotel\_HID` ASC);

CREATE INDEX `Bookings\_Hotel\_idx` ON `HProject`.`Booking` (`Hotel\_booked\_id` ASC);

CREATE INDEX `Bookings\_Cust\_idx` ON `HProject`.`Booking` (`Cust\_booked\_id` ASC);

CREATE INDEX `Bookings\_Employees\_idx` ON `HProject`.`Booking` (`Employee\_booked\_id` ASC);

CREATE INDEX `Employees\_services1\_idx` ON `HProject`.`Employees` (`Department\_ID` ASC);

CREATE INDEX `Employees\_addresses\_idx` ON `HProject`.`Employees` (`Address\_ID` ASC);

CREATE INDEX `Employees\_hotel\_idx` ON `HProject`.`Employees` (`Hotel\_ID` ASC);

CREATE INDEX `cust\_address\_idx` ON `HProject`.`H\_customer` (`Addresses\_id` ASC);

CREATE INDEX `Rooms\_type\_idx` ON `HProject`.`Rooms` (`TypeRoom\_id` ASC);

CREATE INDEX `Hotel\_rooms\_idx` ON `HProject`.`Rooms` (`Hotel\_id` ASC);

CREATE INDEX `Hotel\_address\_idx` ON `HProject`.`Hotel` (`Hotel\_address` ASC);

CREATE INDEX `Hotel\_rating\_idx` ON `HProject`.`Hotel` (`Star\_rating` ASC);

CREATE INDEX `hotel\_chain\_idx` ON `HProject`.`Hotel\_Chain` (`HeadOffice\_id` ASC);

Presentation:

<https://prezi.com/view/2Cjy0zrTYIvu4KAaeU8o/>

Git:

<https://github.com/rassulchsh/Hotel_Database_Management_System_project>