UNIVERSITATEA TEHNICĂ “GH ASACHI” IAŞI

FACULTATEA AUTOMATICĂ ŞI CALCULATOARE

SPECIALIZAREA CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI

DISCIPLINA BAZE DE DATE

Gestiunea unui sistem de management hotelier

Coordonator,

Prof. Mironeanu Cătălin

Student,

Aeloaiei Denisa-Valentina

Grupa 1307A

Iași, 2024

Titlul proiectului

Gestiunea unui sistem de management hotelier

Se analizează modul de funcționare al unui hotel cu scopul de a implementa o bază de date care să se plieze pe nevoile sale.

Descrierea proiectului

În cadrul gestionării unui hotel intervin numeroase operațiuni ce țin de organizarea serviciilor oferite, cât și a resurselor și a clienților, cum ar fi: administrarea camerelor disponibile, a angajaților, gestionarea clienților, managementul serviciilor, recepția etc.

Astfel, se dorește crearea unei evidențe a camerelor (după disponibilitate, preț, capacitate șamd), dar și a datelor clienților, care au posibilitatea de a-și rezerva camera, alegând fie să efectueze plata cu cardul, fie în numerar. De asemenea, clienții au acces și la diferite facilități oferite la nivelul întregului hotel. La final sunt generate facturi în funcție de rezervările efectuate și de serviciile furnizate.

Pentru o bună administrare a acțiunilor organizatorice din cadrul unui hotel, trebuie să reținem informații despre:

- **Camere**: ele constituite principalul serviciu oferit de hotel, de aceea este important să păstrăm o bună evidență a lor;
- **Clienți**: întrucât ei sunt cei care solicită serviciile puse la dispoziție, va fi necesar să reținem date despre ei;
- **Angajați**: facturile vor fi generate prin intermediul recepționerilor, de aceea va trebui să avem la dispoziție informațiile lor;
- **Servicii**: pe lângă posibilitatea de a rezerva o cameră pe un număr de zile ales de client, acesta are acces la câteva servicii suplimentare (al căror preț nu intră în prețul camerei);
- **Rezervări**: ne dorim să ținem evidența rezervărilor, fiind important în momentul în care un client nou ar dori să rezerve o cameră și nu poate, deoarece alege o perioadă în care acea cameră nu este disponibilă, asigurând astfel o bună organizare a serviciilor hoteliere oferite.
- **Facturi**: o rezervare poate fi anulată, în curs de derulare sau onorată. Generarea facturii reprezintă ultimul pas în cadrul închirierii unei camere

și, deci, vom ști astfel că o rezervare a fost onorată în momentul în care plata a fost efectuată.

Structura și relațiile dintre entități

Entitățile din această aplicație sunt:

- Room:
  - room_id: NUMERIC(4), obligatoriu (ia valori între 1000-9999);
  - room_number: NUMERIC(4), unic, obligatoriu (ia valori doar între anumite intervale);
  - floor: NUMERIC(2), obligatoriu (ia valori între 1 și 10);
  - capacity: NUMERIC(1), obligatoriu (ia valori între 1 și 5);
  - price: NUMERIC(4), obligatoriu.
- Client:
  - client_id: NUMERIC(5), obligatoriu (ia valori între 10000-99999);
  - name: VARCHAR2(40), obligatoriu;
  - birth_date: DATE;
  - phone_number: VARCHAR2(10), unic, obligatoriu;
  - email: VARCHAR2(25), unic.
- Employee:
  - employee_id: NUMBER(3), obligatoriu (ia valori între 100 și 999);
  - name: VARCHAR2(40), obligatoriu;
  - phone_number: VARCHAR2(10) unic, obligatoriu;
  - job: VARCHAR2(15), obligatoriu („chelner”, „manager”, „recepție”, „instalator”, „menajer”);
  - hire_date: DATE, obligatoriu;
  - salary: NUMBER(5), obligatoriu.
- Service:
  - service_id: NUMBER(2), obligatoriu (ia valori între 10 și 99);
  - service_name: VARCHAR2(15), obligatoriu;
  - price: NUMBER(4), obligatoriu;
  - description: VARCHAR2(15).
- Booking:
  - reservation_id: NUMBER(6), obligatoriu (ia valori între 100000-999999);

- check_in_date: DATE, obligatoriu;
- check_out_date: DATE, obligatoriu;
- room_total_price: NUMBER(5);
- services_price: NUMBER(5);
- status: VARCHAR2(10) (“derulare”, “onorată” sau “anulată”);
- reservation_date: DATE.

- Bill:
  - total_price: NUMBER(5), obligatoriu;
  - payment_date: DATE;
  - payment_method: VARCHAR2(10), obligatoriu („cash” sau „numerar”);
  - card_number: NUMBER(16).

Normalizarea bazei de date

Normalizarea reprezintă procesul de organizare (fără a pierde din informații) a atributelor și tabelelor dintr-o bază de date relațională, cu scopul de a minimiza redundanța datelor și implicit de a minimiza potențialele erori care pot apărea în manipularea datelor. În cadrul acestui proiect apar 3 forme normale.

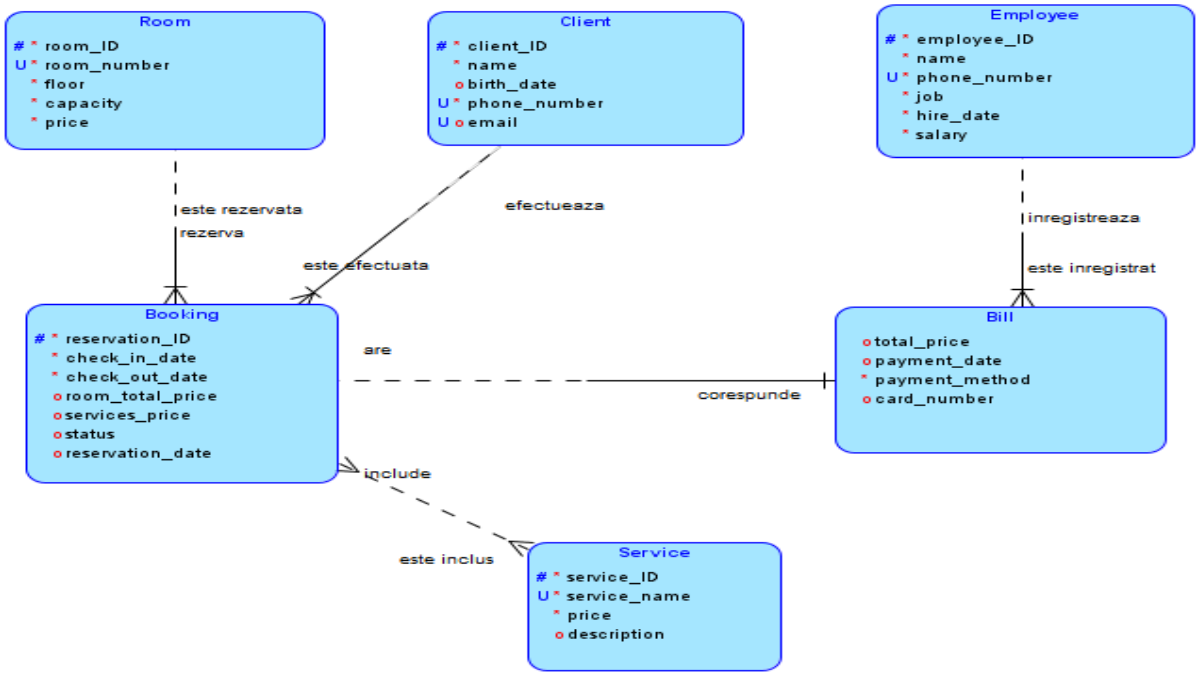
Prima formă normală reprezintă o relație care se definește prin faptul că un atribut conține valori atomice din domeniul său (și nu grupuri de astfel de valori) și prin faptul că nu conține grupuri care să se repete. Această formă apare în cazul entității **Client**, unde clienții sunt diferențiați după număr de telefon și email (și reținem un singur email și un singur număr de telefon pentru fiecare), dar și-n cazul entităților **Room**, **Employee** și **Service** (unde există câte un câmp care acceptă valori unice, singulare).

A doua formă normală reprezintă o relație care, pe lângă faptul că este în prima formă normală, are toate atributele non-cheie dependente în totalitate de toate cheile candidat. În acest caz aparține entitatea **Bill**, ale căror atribute depind de ID-ul rezervării din tabela **Booking**, căci nu se poate discuta de generarea unei facturi în lipsa unei rezervări, dar și de ID-ul angajatului din **Employee**.

A treia formă normală este caracterizată de aspectele definitorii ale relației de a doua formă normală, adăugând faptul că toate atributele non-cheie sunt direct (non-tranzitiv) dependente de toate cheile candidat. Tabela **Booking** creează

legătura între ID-urile specifice entității sale și entitățile Room și Client, pentru validarea unei rezervări efectuate cu succes.

Modelul logic



Tipurile de relații

În proiectarea acestei baze de date s-au identificat următoarele tipuri de relații: 1:1 (one-to-one), 1:n (one-to-many), n:m (many-to-many).

Între entitatea **Room** și entitatea **Booking** se stabilește o relație de 1:n. O cameră poate apărea în mai multe rezervări (ținând cont de faptul că perioadele dintre check_in și check_out ale celor două rezervări să nu se intersecteze. O rezervare nu poate include mai multe camere. Dacă un același client dorește să rezerve mai multe camere, va fi nevoit să creeze câte o rezervare pentru fiecare dintre ele. Legătura între cele două entități se face prin atributul **room_ID**.

Între entitatea **Client** și entitatea **Booking** se stabilește o relație de 1:n. Un același client are posibilitatea de a avea mai multe rezervări, însă o rezervare va fi efectuată de un singur client. Legătura între cele două entități se face prin atributul

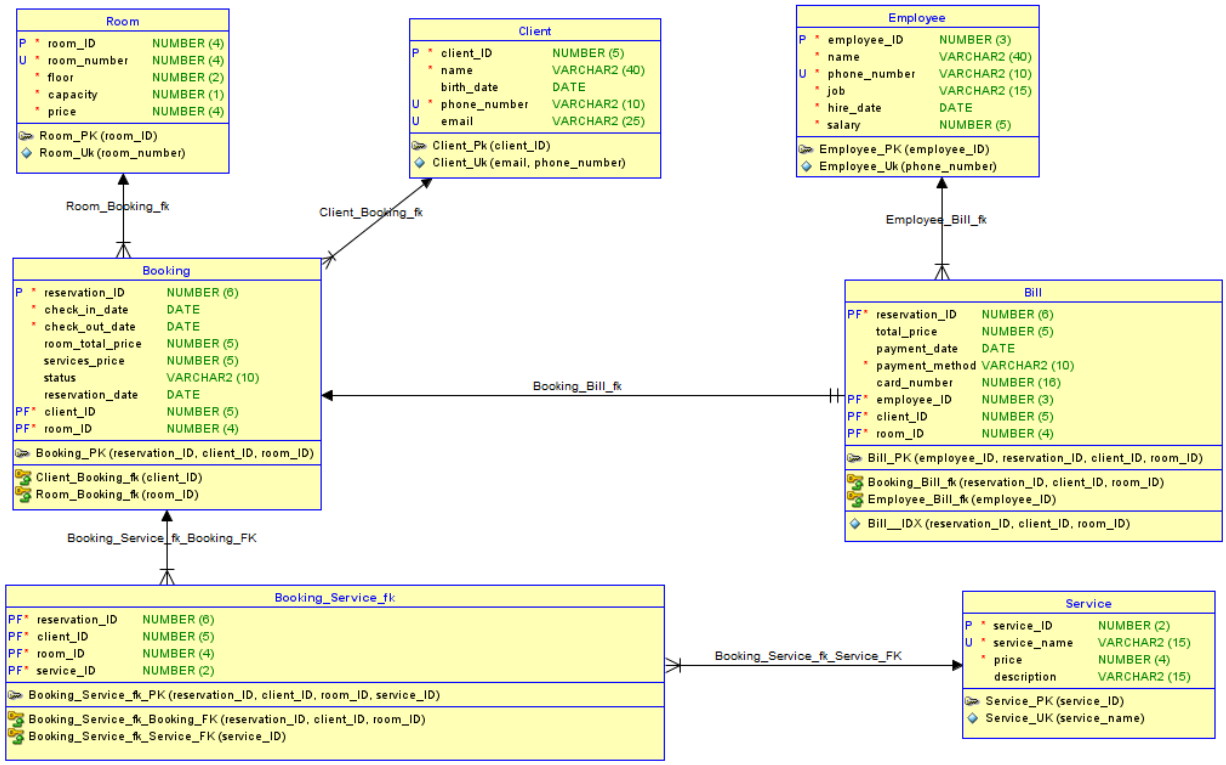
client_ID.

Între entitatea **Employee** și entitatea **Bill** se stabilește o relație de 1:n. Un același angajat se poate ocupa de efectuarea mai multor plăți, însă pentru generarea aceleiași facturi nu e necesar să fie prezenți mai mulți angajați. Legătura între cele două entități se face prin atributul **employee_ID**.

Între entitatea **Booking** și entitatea **Bill** se stabilitește o relație de 1:1. Fiecărei rezervări îi poate corespunde o factură, în momentul în care clientul decide să onoreze rezervarea. Acestuia i se alocă o perioadă limitată de la data rezervării de a plăti comanda. Dacă nu este plătită până la expirarea termenului limită, rezervarea va fi anulată. Deci, pot exista rezervări a căror plată să nu fie încă efectuată la momentul curent, însă nu pot exista facturi fără rezervarea aferentă. Legătura între cele două entități se face prin atributul **reservation_ID**.

Între entitatea **Booking** și entitatea **Service** se stabilește o relație de n:m. Fiecare rezervare poate conține mai multe servicii suplimentare solicitate, iar un același serviciu poate exista în mai multe rezervări, neexistând restricție pentru numărul de clienți care au cerut în rezervarea lor un același serviciu. Această relație este modelată prin tabela intermediară **Booking_Service**, care conține ID-urile pentru rezervare, client, cameră și serviciu.

Modelul relațional



Descrierea constrângerilor

Constrângerile de tip check se găsesc în aproape toate tabelele. Aceastea verifică dacă anumite valori sunt pozitive, dacă textul a fost introdus în formatul corect, dar și dacă a fost aleasă o opțiune dintr-o mulțime sau a fost introdus ceva nou. În cazul în care nu este respectată condiția din check, va fi introdusă o eroare.

În toate tabelele care conțin atributul pentru preț, verificăm dacă aceste este mai mare sau egal cu 0, în funcție de context (Room, Booking, Service, Bill). De asemenea, anumite câmpuri pot lua valori doar între anumite intervale (room_number, floor, capacity). Alte câmpuri pot selecta o opțiune din o listă deja predefinită, cum ar fi cazul atributului payment_method din Bill (card, numerar), status din Booking (derulare, onorata, anulata), job din Employee (chelner, instalator, manager, menajer) sau service_name din Service (cazino, fitness, spa, restaurant). De asemenea, este verificat tiparul atribului name din entitățile Client și Employee (să fie de forma '([A-Z]+[a-z]+)([A-Z]+[a-z]+)*'), dar și a email-ului ('[a-z0-9._%-] +@[a-z0-9._%-]+\.[a-z]{2,4}') și a numărului de telefon ('^07[0-9]{8}$').

A fost necesară introducerea și a constrângerilor de tip unique, prin care se asigură că două înregistrări nu conțin aceeași dată pentru anumite atribute, astfel asigurându-se diferențierea între acele înregistrări. În cazul entităților Client și Employee, nu pot fi introduse 2 numere de telefon identice sau 2 email-uri identice. Pentru entitatea Room, trebuie să asigurăm numărul camerei unic. Service_name din cadrul entității Service trebuie să fie de asemenea unic.

Constrângerile de tip primary-key au fost și ele adăugate în cadrul atributelor care reține ID-ul (room_ID din Room, client_ID din Client, booking_ID din Booking, bill_ID din Bill, service_ID din Service, employee_ID din Employee), fiind generate de baza de date printr-un mecanism de tip autoincrement.

Constrângerile de tip not null sunt adăugate pentru majoritatea atributelor din entități. Există câteva excepții pentru care nu este necesară introducerea unor valori, cum ar fi cazul atributului card_number din entitatea **Bill**. Un client care alege ca metodă de plată "cash" nu este nevoit să mai introducă numărul cardului (card_number). De asemenea, atributul services_price din entitatea **Booking**, care reține costul serviciilor, poate fi null pentru acele rezervări care nu au solicitat servicii suplimentare. Nici pentru atributul email din entitatea **Client** nu a fost introdusă obligativitatea, întrucât deja este folosit numărul de telefon (phone_number) ca dată de contact obligatorie în caz de strictă necesitate. În cazul atributelor care reține date (reservation_date din **Booking** și payment_date din **Bill**), permitem introducerea valorii de null.

Constrângerile de tip foreign-key au fost adăugate în urma introducerii tipurilor de relații menționate mai sus. Atributul Client_Booking_Client_ID conectează entitățile **Client** și **Booking** prin relație de 1:n, Room_Booking_Room_ID conectează entitățile **Room** și **Booking** prin relație de 1:n, Employee_Bill_Employee_ID conectează entitățile **Employee** și **Bill** printr-o relație de 1:n, Booking_Service conține 2 foreign key-uri, conectând entitățile **Booking** și **Service** într-o relație de n:m, iar Booking_Bill_Reservation_ID leagă entitățile **Booking** de **Bill** într-o relație de 1:1.

De asemenea, au fost adăugate triggere, verificând ca angajatul care generează factura (Bill) să fie doar recepționer, ca data emiterii facturii (payment_date) să fie înainte de data curentă (nu pot fi generate facturi în viitor), ca check_out_date să nu fie înaintea check_in_date-ului, ca intervalul dintre

check_in_date și check_out_date al unei rezervări să nu se intersecteze cu intervalul de check_in_date și check_out_date al unei alte rezervări în cadrul aceleiași camere, ca data rezervării să nu depășească data curentă sau check_in_date-ul, ca data nașterii unui client (birth_date din Client) sau data angajării unui employee (hire_date din Employee) să nu depășească data curentă și ca etajul unei camere (floor) să coincidă cu prima cifră a numărului camerei (room_number). Un ultim trigger va monitoriza actualizarea datelor din Booking, atunci când se dorește modificarea statusului unei rezervări (derulate, onorată, anulată).

Utilizare use-case

Un use case în SQL reprezintă un scenariu specific care ilustrează modul în care o bază de date poate fi utilizată în cazurile practice.

În cazul proiectului prezentat, am introdus câteva cazuri care utilizează un use case atât pentru modificarea datelor din tabel, cât și pentru selectarea informațiilor spre a le afișa. Un prim caz de use-case actualizează entitatea Booking de fiecare dată când au fost generate facturi (datele din tabela Bill), adăugând statusul nou al rezervării (derulare, onorată, anulată). Dacă încă nu a fost generată nicio factură care să rețină ID-ul rezervării, atunci rezervarea este în curs de derulare. În momentul în care a fost efectuată o plată înainte de data de check_out, rezervarea va fi onorată, iar câmpul status va reține această informație. Depășirea datei de check_out fără a plăti rezervarea o va marca ca fiind anulată.

De asemenea, a fost adăugat un use-case care descrie accesibilitatea serviciilor puse la dispoziție de hotel în funcție de tipul lor. Sunt anumite servicii care funcționează pe timpul zilei (cum ar fi restaurantul sau fitness-ul) și servicii care funcționează pe timpul nopții (cum ar fi cazinoul).

Tot utilizând un use-case, am dorit să descriu (afișez) programul de lucru al angajaților în funcție de job-ul pe care fiecare dintre ei îl are, verificând astfel disponibilitatea lor.