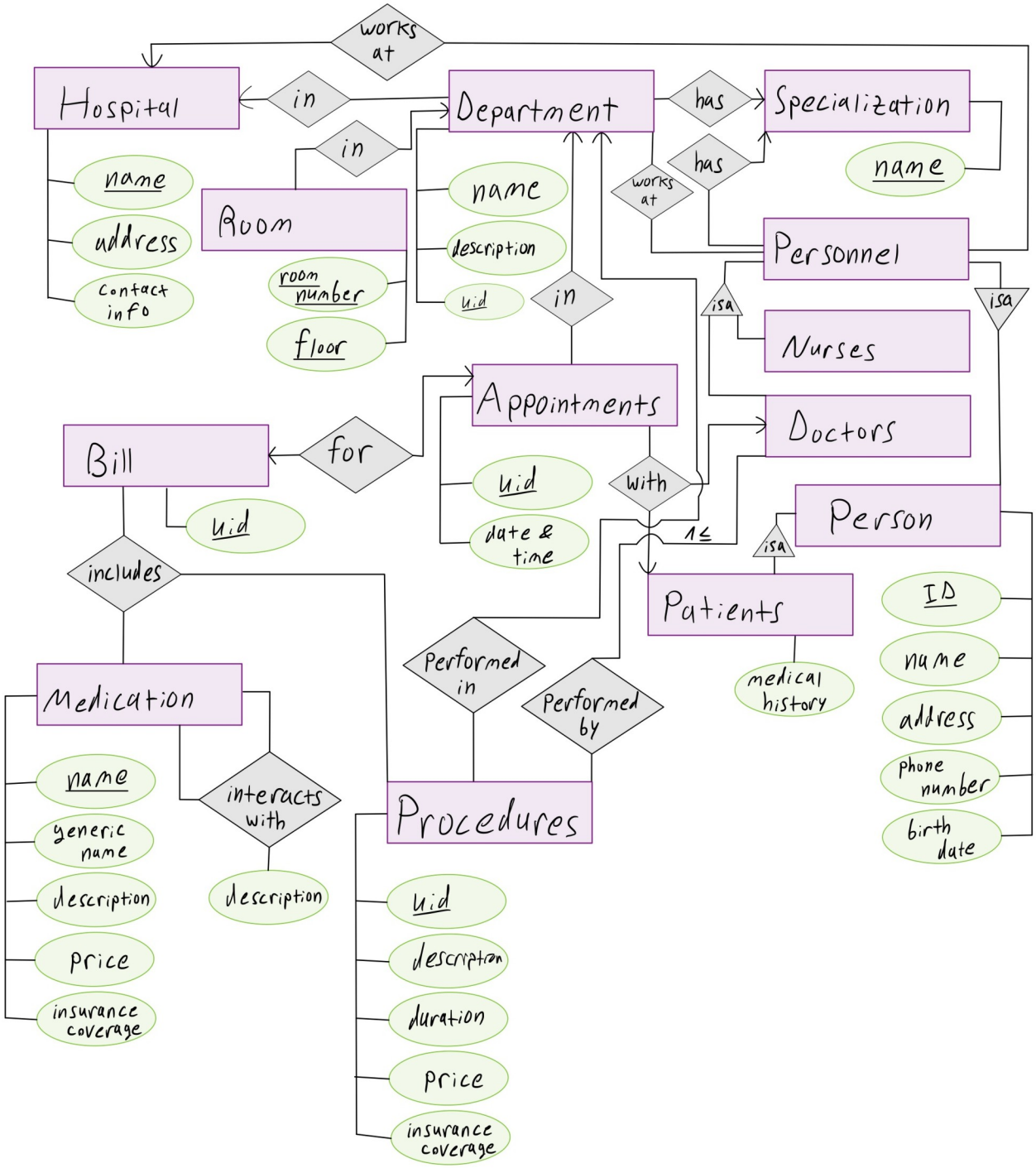
DBMS Assignment 2

ER Diagram:



Assumptions:

- A department must be in a certain hospital.
- A department's name isn't unique, not even in the same hospital (added uid).
- A room must be assigned to a single department.
- All rooms in the same floor have different numbers.
- Each department and personnel can have 1 specialization at most.
- There can be multiple specialists.
- Personnel can work at multiple departments.
- An appointment must have 1 patient, 1 primary doctor and 1 associated department.
- A procedure can be performed by multiple doctors (e.g. complicated surgery), but at least 1.
- A procedure must be performed in a specific department.
- A bill can only be, and must be, produced for a specific appointment (1 to 1)

Entities
Hospital(name, address, contact info)
Specialization(name)
Department(uid, name, description, hospital_name, specialization) FK(hospital_name → Hospital.name) FK(specialization → Specialization.name)
Room(room_number, floor, department_uid) FK(department_uid → Department.uid)
Person(id, name, address, phone_number, birth_date)
Patients(id, medical_history) FK(id → Person.id)
Personnel(id, hospital_name, specialization) FK(id → Person.id) FK(hospital_name → Hospital.name) FK(specialization → Specialization.name)
Doctors(id) FK(id → Personnel.id)
Nurses(id) FK(id → Personnel.id)
Appointments(uid, date_time, department_uid, primary_doctor_id, patient_id, bill_id) FK(department_uid → Department.uid) FK(primary_doctor_id → Doctors.id) FK(patient_id → Patients.id) FK(bill_id → Bills.id)
Procedures(uid, description, duration, price, insurance_coverage, department_uid) FK(department_uid → Department.uid)
Medications(name, generic_name, description, price, insurance_coverage)
Bills(uid, appointment_uid) FK(appointment_uid → Appointments.uid)

Relations
Department_Personnel(personnel_id, department_uid) FK(personnel_id → Personnel.id) FK(department_uid → Department.uid)
Procedures_Doctors(procedure_uid, personnel_id) FK(procedure_uid → Procedures.uid) FK(personnel_id → Personnel.id)
Medication_Interactions(medication1_name, medication2_name, description) FK(medication1_name → Medication.name) FK(medication2_name → Medication.name)
Bill_Medication(bill_uid, medication_name) FK(bill_uid → Bill.uid) FK(medication_name → Medication.name)
Bill_Procedure(bill_uid, procedure_uid) FK(bill_uid → Bill.uid) FK(procedure_uid → Procedure.uid)

Notes:

- Patients can be admitted to multiple hospitals because there is no limitation on the amount or type of appointments a patient can have. A patient can have multiple appointments, in different departments, which are in different hospitals.
- There is no need for a relation between bills and patients, as this can be achieved via the appointments a patient had (which are connected to the bills in a 1-to-1 fashion).
- There is no need for a relation between procedures and patients, as this can be achieved by looking at the bills a patient received, and the associated entry in *Bills_Procedures*.
- A description was added to *Medication_Interactions*, so doctors (and if we're being honest – mostly nurses – which do most of the medication work anyway) can actually do their jobs.

Views and Query containment:

First lets analyze the view. We are joining *account* and *customer*. Since every account only has 1 owner (*account.id* is a key and account has 1 field for *owner*), we are simply adding *customer.name* and *customer.credit_rating*, to each row in *account*. We do not filter any rows becaues we assume there are no NULL values. So our view is equivalent to simply filtering the *account.owner* from *account*.

1. B. First let's observe the 2nd condition. We are joining the *account* and *currency* tables in the same fashion we did for the accounts in the view, meaning, we simply add the *currency* columns to the *account* table. This does not filter or add any results, as *currency.id* is a key and each account has a single currency in which it operates.
   Now lets observe the 1st condition. We are joining the *account* and *saving_account* tables on their ids. This will give us all the rows present in *saving_account* (with the currency columns added), but will not include *current_account* (or accounts not associated with either current or saving). This means we will only get partial information as compared to the view.

2. D. Although joining *account* and *saving_account* on the type of currency doesn't filter out other types of accounts, we are still selecting *DISTINCT saving_account.id*, which will not contain any id of a current account – therefore we can rule out A and C.
   Selecting DISTINCT does not guarantee in which order we will receive our tuples. Considering this, and the fact we joined on the type of currency, and selecting *balance* from *account*, we might get a savings account with a balance of a different account (their ids don't match, but their currencies do). So we can rule out B.
3. A. We are joining account and customer in the same fashion as the view.
   Grouping by *account.id*, which is a key and can only have a single customer and currency anyway we essentially retain the structure of the joined table (at least for the columns we are interested in; other columns which we do not try and select will not be accessible but are not relevant to us anyway).
   Now selecting *MAX(account.balance)* will give the max of every single row (which is just the balance). So we receive the same exact tuples.
4. B. Using EXISTS simply checks if the table is empty. It does not add new rows to the table. Since we joined account on current_account only, we are missing all of the saving accounts. As an example, assume all account are saving accounts. The query will obviously not return any tuples (*SELECT current_account.id*) but saving account do exist.

Normalization & BCNF:

(a) Firstly, we can see that no rule derives A and D, meaning they are independent, thus must be part of the minimal key set.
Computing $\{A, D\}^+ = \{A, D, E, F\}$. Since $B, C \notin \{A, D\}^+$ we compute adding each of the attributes.
$\{A, B, D\}^+ = \{A, B, C, D, E, F\}$
$\{A, C, D\}^+ = \{A, B, C, D, E, F\}$
Minimal keys sets: $\{A, B, D\}, \{A, C, D\}$.

(b) $A, B \rightarrow C$ violates BCNF.
Decompose:
$R1(A, B, C)$
FDs:
$A, B \rightarrow C$
$C \rightarrow B$
Minimal Keys: $\{A, B\}, \{A, C\}$
$R2(A, B, D, E, F)$
FDs:
$A, B \rightarrow E$
$D \rightarrow E, F$
Minimal Keys: $\{A, B, D\}$

$C \rightarrow B$ in R1 violates BCNF.
Decompose:
$R11(C, B)$
FDs:
$C \rightarrow B$
Minimal Keys: $\{C\}$
$R12(C, A)$
Minimal Keys: $\{A, C\}$

$D \rightarrow E, F$ in R2 violates BCNF.
Decompose:
$R21(D, E, F)$
FDs:
$D \rightarrow E, F$
Minimal Keys: $\{D\}$
$R22(D, A, B)$
Minimal Keys: $\{D, A, B\}$

Final decomposition: $R11(C, B)$, $R12(C, A)$, $R21(D, E, F)$, $R22(D, A, B)$.

(c) We have 2 non-trivial FDs that are not preserved: $A, B \rightarrow C$, $C \rightarrow E$.

$A, B \rightarrow C$:

R11			R12			R1		
B	C	JOIN	A	C	=	A	B	C
b1	c1		a2	c1		a2	b1	c1
b1	c2		a2	c2		a2	b1	c2

$A, B \rightarrow C$ is violated.

$C \rightarrow E$:

R11			R12			R1		
B	C	JOIN	A	C	=	A	B	C
b1	c1		a1	c1		a1	b1	c1
			a2	c1		a2	b1	c1

JOIN

R21			JOIN	R22			=	R2				
D	E	F		A	B	D		A	B	D	E	F
d1	e2	f1		a1	b1	d1		a1	b1	d1	e2	f1
d2	e1	f1		a2	b1	d2		a2	b1	d2	e1	f1

=

R					
A	B	C	D	E	F
a1	b1	c1	d1	e2	f1
a2	b1	c1	d2	e1	f1

$C \rightarrow E$ is violated.