Hotel

Si presuppone una base di dati per la gestione di un hotel.

Camere, clienti, prenotazioni.

L'hotel è formato da un certo numero di camere, che vengono riservate ai clienti tramite prenotazione. Una prenotazione prevede una data di arrivo e una data di partenza, e una camera può avere una e una sola prenotazione associata in una certo lasso di tempo. Un cliente effettua più prenotazioni, una prenotazione è effettuata da un solo cliente. Per ogni prenotazione vengono specificati il numero di ospiti (il numero di ospiti deve essere minore o uguale del numero di posti letto della camera associata).
In ogni prenotazione si registra anche il costo totale (composto da costo per notte, costi extra e durata del soggiorno meno sconti o promozione giornaliera).
Una promozione giornaliera è uno sconto su tutte le camere che vengono prenotate in una certa giornata.
La camera è identificata dal numero della stanza, altri attributi sono il costo per notte e il numero di posti letto. Un cliente è determinato dai soliti dati anagrafici. A ogni cliente spettano sconti fedeltà dopo che ha prenotato un certo numero di camere.
Tra questi due sconti (promozione giornaliera e sconto fedeltà) viene applicato solo lo sconto maggiore su una certa prenotazione.

Dipendenti, turni.

L'hotel ha anche una serie di dipendenti, identificati da codice fiscale più i soliti dati anagrafici. I dipendenti si dividono in personale pulizie, receptionist e dirigenti. Ogni lavoratore del personale pulizie in un dato giorno pulisce diverse camere, mentre ogni camera, in un singolo giorno, è pulita da un solo lavoratore. (Data una camera e un turno, ho un solo dipendente assegnato. Dato un dipendente e un giorno, ho più turni associati a più camere.) Ogni turno dura un' ora, e in ogni turno viene pulita una sola camera. Il valore della retribuzione oraria varia a seconda della camera pulita. Ogni dipendente effettua al massimo sei turni una stessa giornata.
Anche i receptionist sono organizzati con dei turni. Ogni receptionist può effettuare più turni(composti da data e fascia giornaliera), mentre un turno in reception è effettuato da un solo receptionist, che può fare al massimo un turno per ogni giorno. Ogni turno è da 6 ore.
Personale pulizie e receptionist hanno la possibilità di avere un sostituto fra i colleghi di lavoro per coprire un determinato turno.
Lo stipendio dei dirigenti è un dato fisso, mentre quello di personale pulizie e receptionist è un dato derivato calcolato dalla retribuzione oraria per il numero di turni .

Servizi extra.

Ad una prenotazione si possono collegare servizi extra. L'aggiunta di ogni servizio extra implica un costo aggiuntivo.

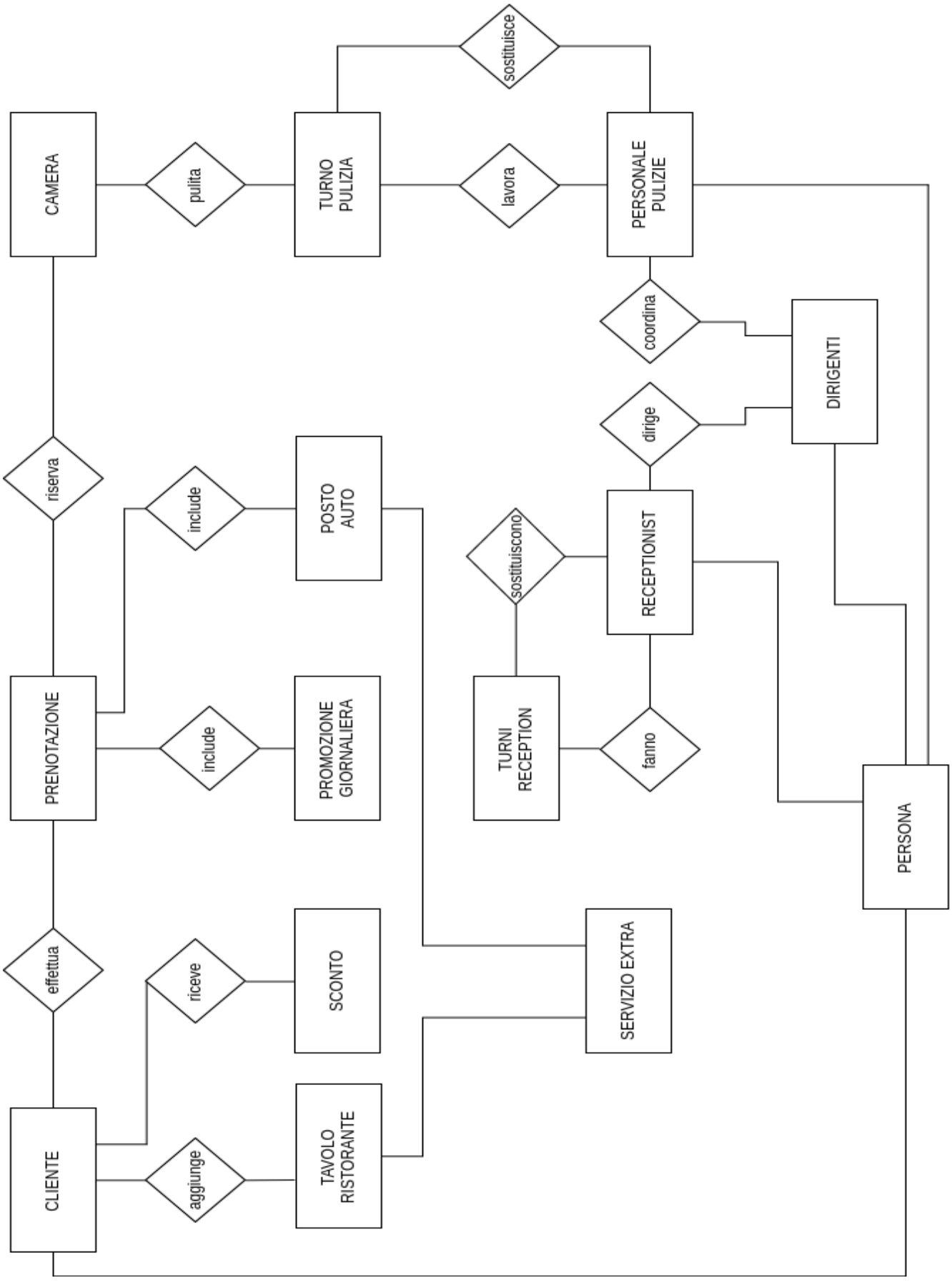
Viene data la possibilità di riservare uno o più tavoli al ristorante, data la camera (prenotata). Un tavolo viene riservato in una certa data e ora per una certa prenotazione, è inoltre definito un costo a persona e il numero di posti prenotati.
Fra i servizi extra viene data la possibilità di prenotare un certo numero di posti auto. Il posto auto è identificato dal numero del posto e dai dati della prenotazione a cui si riferisce, ed ha un costo giornaliero.

Glossario

Termine	Descrizione	Sinonimi	Legami
Persona	• Cod. Fiscale • Nome • Cognome	• Soggetto	• Dipendente • Cliente
Dipendente	• Stipendio		• Pers. Pulizie • Receptionist • Dirigente
Cliente		• Ospite	• Prenotazione • Sconto
Prenotazione	• Data_inizio • Data_fine • Cauzione • Num_persone • Costo_totale	• Soggiorno	• Cliente • Camera • Promozione Giornaliera • Posto auto • Tavolo ristorante
Camera	• Numero • Costo_notte • Num_posti	• Stanza	• Prenotazione • Turno pulizie
Turno pulizie	• Giorno • Retribuzione oraria	• Turno di lavoro settimanale	• Camera • Personale pulizie
Personale pulizie			• Dipendente • Turno pulizie • Dirigente
Dirigente		• Supervisore	• Dipendente • Personale pulizie • Receptionist
Receptionist			• Dipendente • Dirigente • Turno reception
Turno reception	• Giorno • Fascia giornaliera • Retribuzione oraria	• Turno di lavoro settimanale	• Receptionist

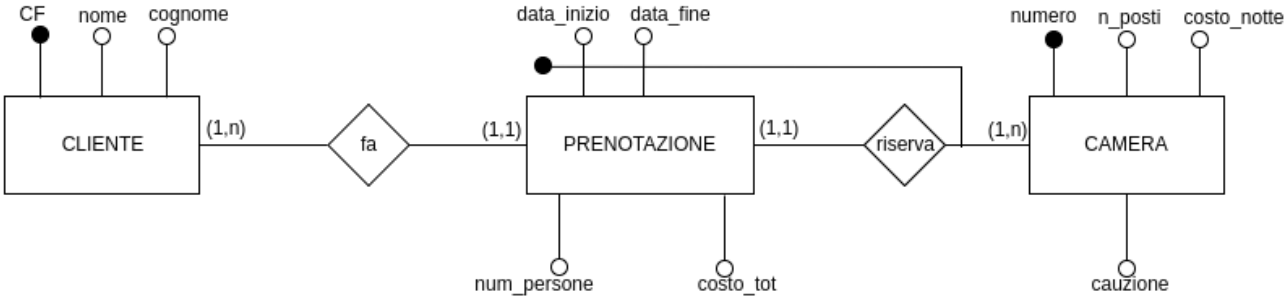
Promozione giornaliera	• Percent. sconto • Titolo • Data	• Sconto • Promozione per una prenotazione in una certa data	• Prenotazione
Sconto	• Cod. offerta • Descrizione • Percentuale	• Sconto clienti abituali	• Cliente
Posto auto	• Posto • Costo giornaliero	• Parcheggio	• Prenotazione • Servizio extra
Tavolo ristorante	• N_tavolo		• Servizio extra • Riservare
Riservare	• Data • Ora • Posti • Costo persona	• Prenotare	• Clienti • Tavolo ristorante

Schema Scheletro



Progettazione concettuale

Camere, clienti, prenotazioni.



Progettazione di camere, clienti e prenotazioni. L'entità prenotazione viene pensata come un "riservare" una certa camera, per cui si è scelto di utilizzare un'identificazione esterna con la chiave di camera, per rafforzare l'idea di dipendenza fra le due entità.

Nell'entità *Cliente* sono esplicitati solamente nome e cognome fra gli attributi, ma ovviamente fanno parte del cliente gli usuali dati anagrafici, telefono, etc. Si è scelto di non inserirli tutti per non rendere eccessivamente incomprensibile la rappresentazione dell'ER.

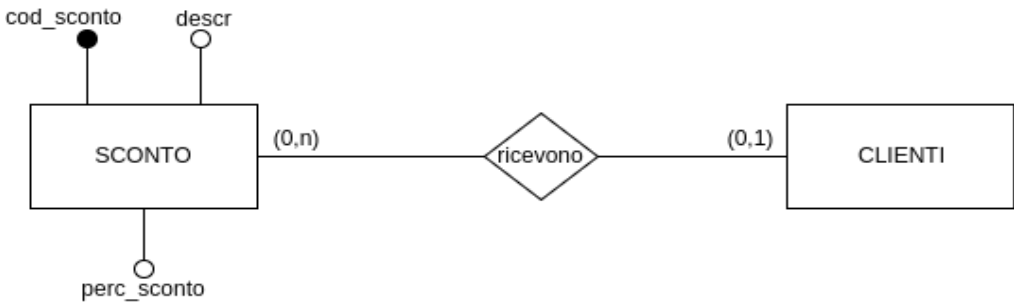
La prenotazione è identificata da numero di camera prenotata e il periodo scelto per il soggiorno. L'utilizzo dei due attributi *DataInizio* e *DataFine* per definire la chiave primaria fa nascere un potenziale problema per quanto riguarda la possibile sovrapposizione della durata di una prenotazione, relativamente ad una certa camera.
Ad esempio, si potrebbe avere una prenotazione per la camera X che vada dalla data A alla data B, ma per la stessa camera sarebbe accettata una prenotazione dalla data A alla data C. Questo problema verrà successivamente risolto con uno specifico trigger.

Per ogni prenotazione, il numero di persone incluse nella prenotazione non deve superare il numero di persone che una camera può contenere. Questo controllo verrà effettuato con uno specifico trigger.

Si è scelto di rappresentare un dato derivato all'interno dell'entità *Prenotazione* (CostoTot), il cui costo ed utilità sarà valutato successivamente. Il costo per una camera viene calcolato come il prodotto fra *Costo per notte* e *Numero di notti* (contenute nell'entità Prenotazioni, calcolate come *DataFine – DataInizio*), a cui si aggiunge la somma di eventuali sconti, mente il costo totale di una prenotazione è più articolato, e comprende la somma di tutte le spese effettuate durante il soggiorno (ossia tutti i *Servizi Extra*).

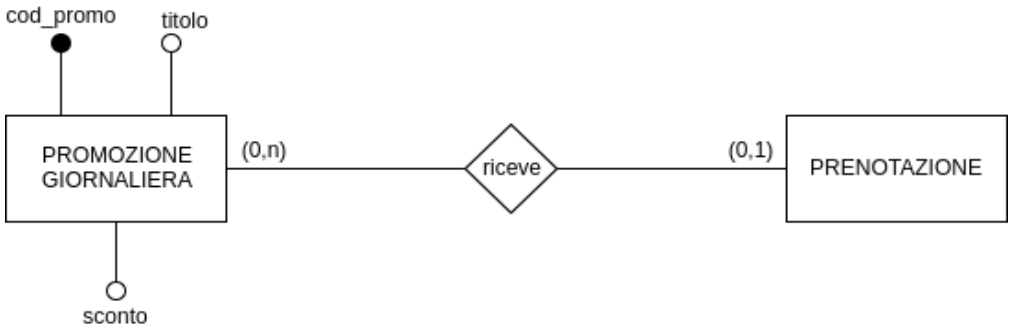
Sconti e promozioni giornaliere.

Sconti:



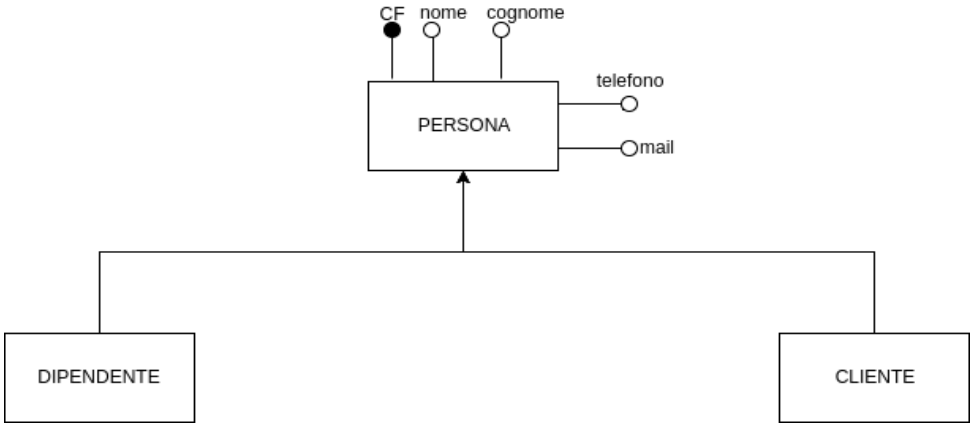
Essendo una sorta di "sconto fedeltà" l'entità *Sconto* è direttamente collegata a *Clienti*. L'idea è che all'aumentare del numero di prenotazioni effettuate da un dato cliente, maggiore sarà lo sconto associato per la prenotazione successiva. La percentuale di sconto viene sottratta al costo della camera.

Promozioni giornaliere:



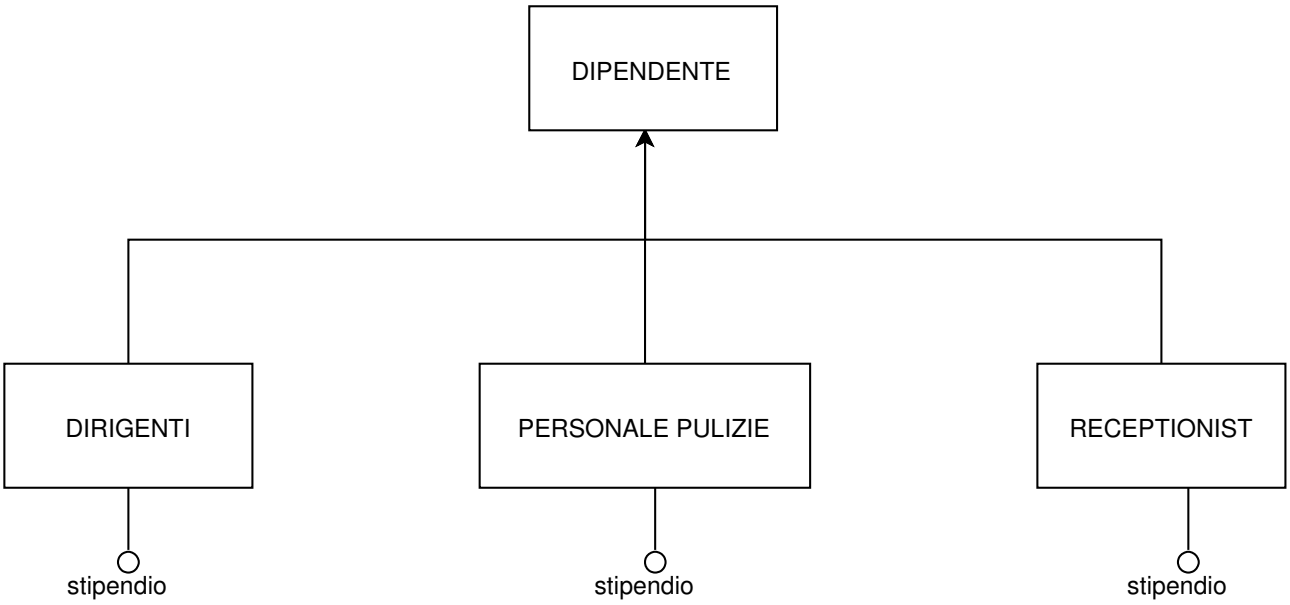
La natura della promozione giornaliera è leggermente diversa: si tratta di un'offerta per prenotazioni fatte in un determinato giorno, e conseguentemente si è scelto di creare una nuova entità, la quale è stata collegata a *Prenotazione*.
Le cardinalità sono state decise supponendo che una prenotazione possa avere al massimo uno sconto per volta, mentre una singola promozione può essere associata a più prenotazioni.

Personale.



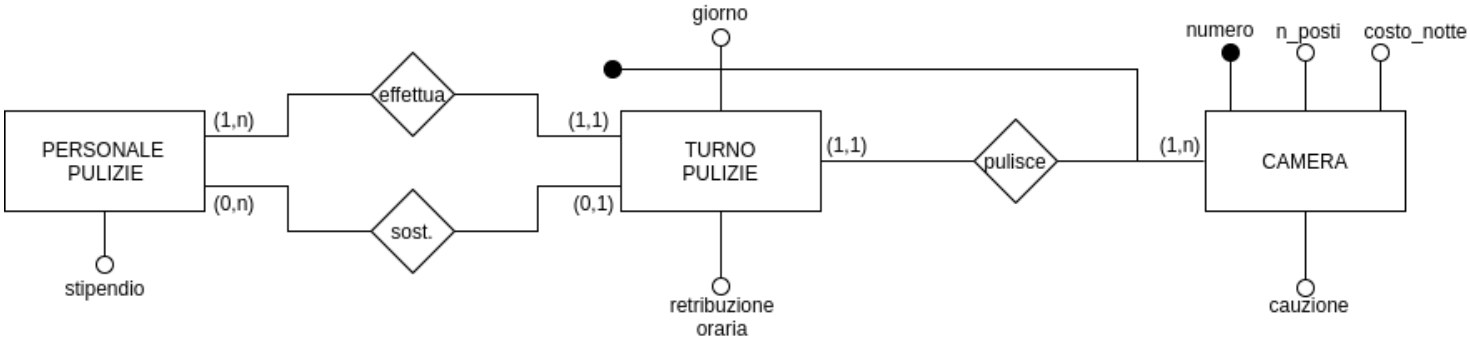
La struttura ISA identifica *Dipendente* e *Cliente* come entità figlie di *Persona,* la quale ha come attributi i soliti dati anagrafici (alcuni vengono omessi per brevità).

Dipendenti:



Si suppongono 3 diverse categorie di dipendenti. Le differenze fra queste categorie sono prevalentemente concettuali, non vi è una vera e propria diversità da un punto di vista strutturale, tuttavia è sembrato opportuno distinguerle in tre entità diverse, date le notevoli differenze nello scopo. Si è pensato di racchiudere tutti i dati anagrafici in *Dipendente*.

Personale pulizie, turni, camere:

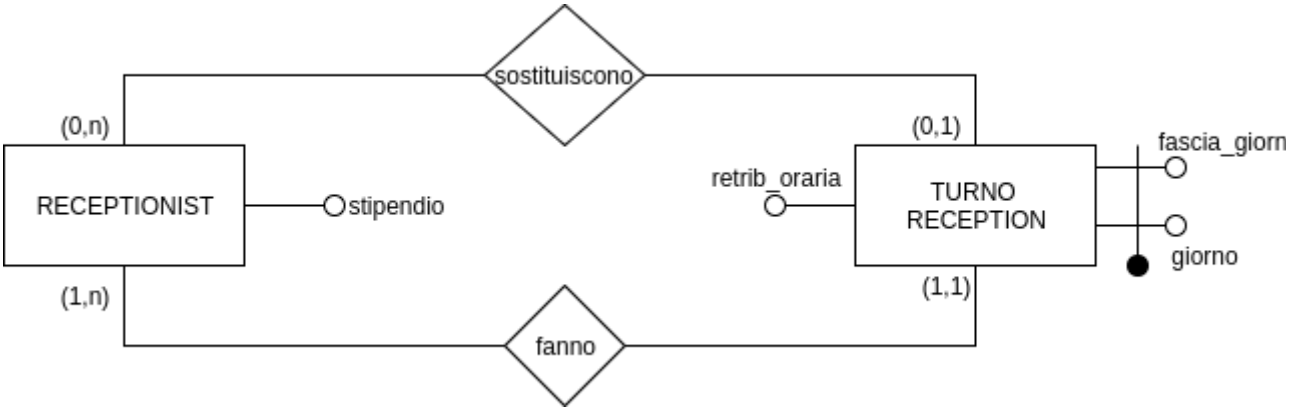


Si è scelto di rappresentare il *Turno pulizie* come un'entità subordinata a *Camera*, questo per enfatizzare la dipendenza fra le due dovuta al fatto che una camera, in un determinato giorno, è pulita una volta soltanto, mentre un membro del personale pulizie pulisce più camere. Ogni turno dura un'ora, e ogni dipendente esegue più turni in una singola giornata.

Lo stipendio di un dipendente è calcolato tramite un dato derivato (*stipendio)* . *Stipendio* è calcolato come somma di tutte le *retribuzione_oraria* di tutti i turni effettuati da un certo dipendente. Qui l'idea è che il dato derivato rappresenti lo stipendio mensile, totalizzato come somma di tutti i turni realizzati in un mese, che possono così essere visualizzati a fine mese. Si suppone che all'inizio di ogni mese, tutti gli stipendi siano azzerati tramite una query.

Il fatto che un singolo dipendente possa effettuare al massimo sei turni in una stessa giornata verrà controllato da un trigger opportuno.
Gli attributi di Personale Pulizie non sono elencati in quanto si suppone che questa entità sia figlia dell'entità padre Dipendenti, dalla quale si ereditano i dati.

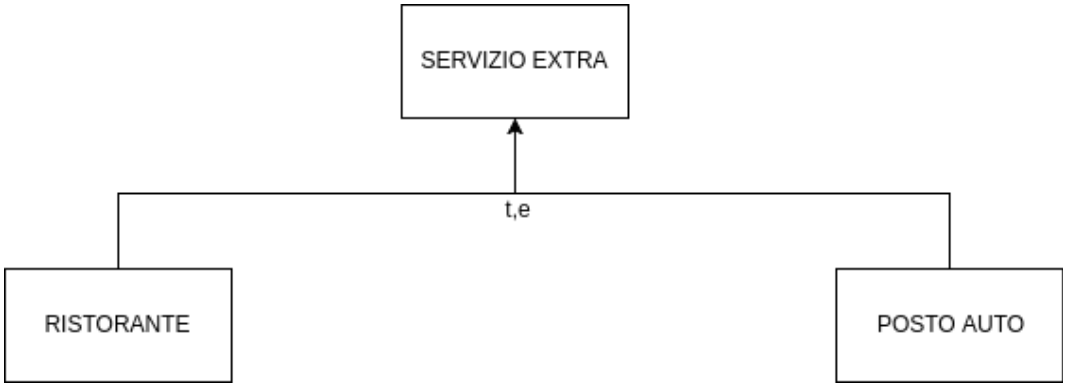
Receptionist e turni:



La situazione è simile a *Personale pulizie,* ogni receptionist effettua un turno in una determinata fascia giornaliera. Ogni receptionist può effettuare al massimo un turno al giorno. Le cardinalità sono scelte in modo che un receptionist venga sostituito da al

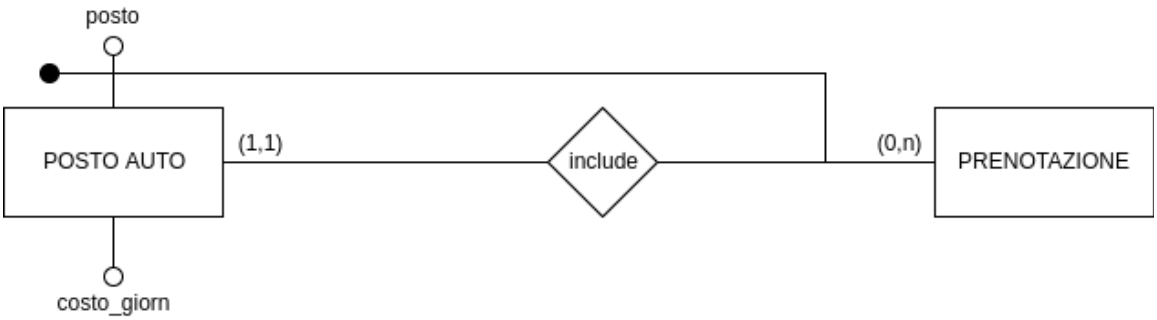
massimo un suo collega, e viceversa un receptionist può coprire più turni (o anche nessuno) di suoi colleghi.

Servizi extra.



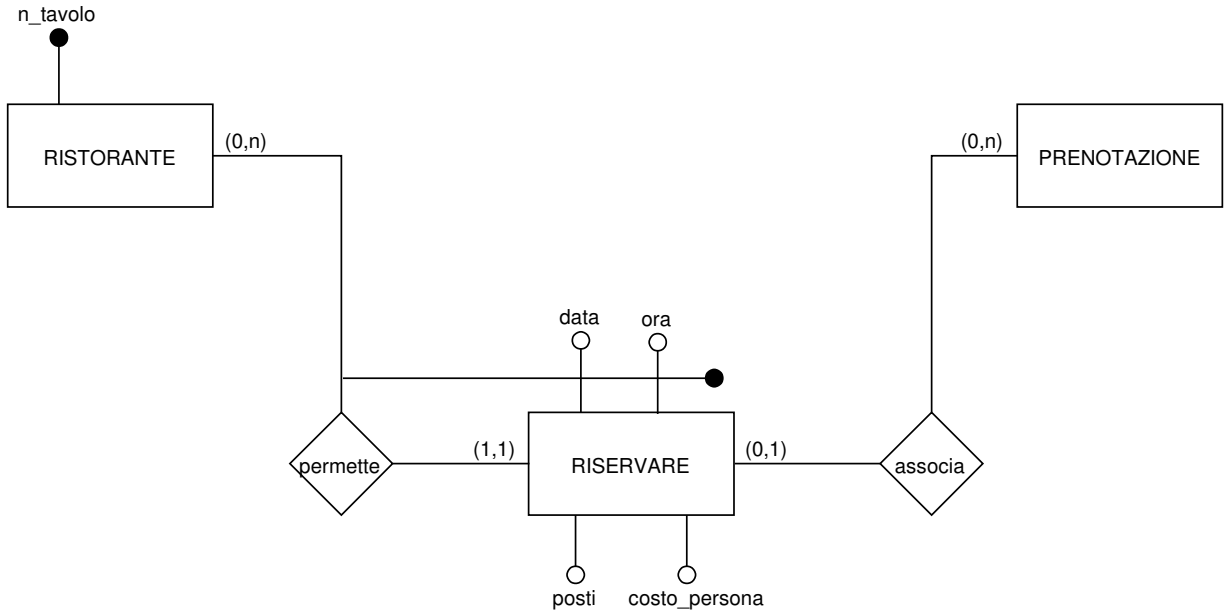
Trattandosi di un hotel, ogni ospite ha la possibilità di usufruire di *Servizi Extra,* quali l'utilizzo di un *Posto auto* o del *Ristorante*. Si presuppone che tali servizi siano a pagamento e il loro costo venga aggiunto al costo totale memorizzato nella *Prenotazione* di un dato cliente, il che implica un collegamento fra ogni servizio con l'entità *Prenotazione*. Per ogni servizio utilizzato vi sarà quindi uno specifico incremento del dato derivato contenuto in *Prenotazione*.
In questo progetto vengono esplicitati solo due possibili servizi extra, in quanto erano sembrati i più coerenti e adatti, tuttavia la struttura del database ammette facilmente l'immissione di nuovi servizi, relativi alle possibili necessità di specifici hotel.

Posto Auto:



Il servizio rappresentato definisce un posto auto che, se esistente, si suppone sia occupato e pagato dall'inizio alla fine della prenotazione. Un certo posto auto è associato a una singola prenotazione (ossia in un certo periodo, definito nella prenotazione, un certo posto auto può essere associato a una sola persona). In questo modo, quindi, assicurandoci che *Posto auto* sia rappresentata come un'entità dipendente da *Prenotazione* ci assicuriamo che in un certo periodo un posto auto non possa essere associato a due clienti diversi.

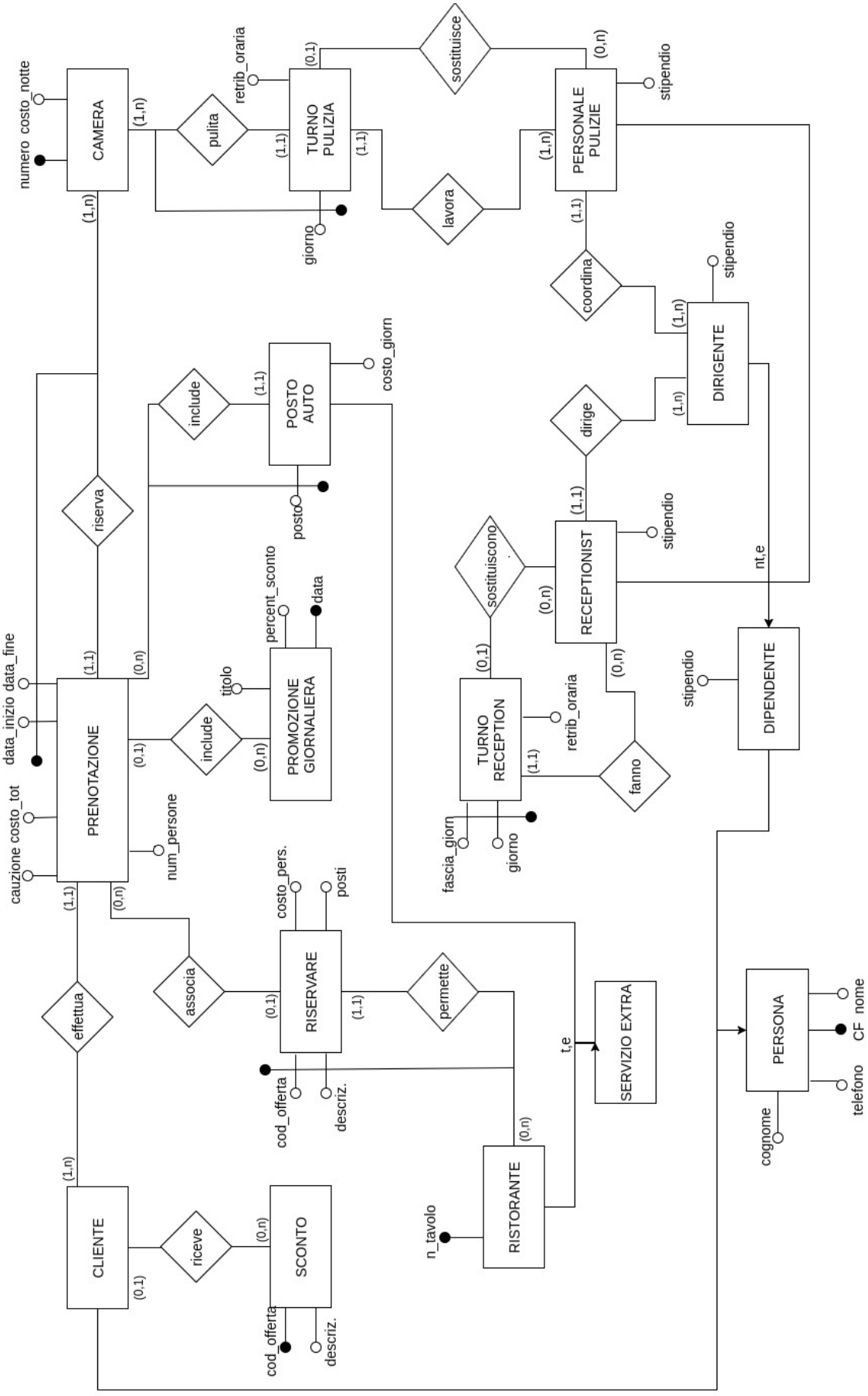
Ristorante:



Si presuppone il ristorante come un servizio a pagamento disponibile solo ed esclusivamente ai clienti dell'hotel. In questo modo solamente clienti che si trovano all'interno dell'hotel, e che quindi dispongono di una prenotazione possono effettivamente prenotare un tavolo al ristorante (ossia avere un istanza associata nell'entità *Riservare*). Questo spiega perché *Riservare* e *Prenotazione* sono state collegate, al posto di *Riservare* e *Cliente*.
Ricapitolando, *Prenotazione* esprime le prenotazioni effettuate da clienti su una certa camera, mentre *Riservare* contiene le prenotazioni di un certo tavolo di ristorante effettuate da un certo cliente.
Si è deciso di reificare l'associazione fra *Ristorante – Prenotazione* per gestire con una maggiore flessibilità la prenotazione di un certo tavolo del ristorante, meglio gestendo il numero di persone, il costo a persona, ecc. *Data* e *ora* sono scelte per creare la chiave primaria, relativamente al tavolo del ristorante, nel senso che a una certa data, in un certo orario, solamente un tavolo (*Ristorante*) può essere prenotato. Quindi un'istanza di *Riservare (*cioè un tavolo,ad una certa data, in una certa ora*)* è associata ad un singolo cliente (ossia ad una specifica *Prenotazione*), mentre un singolo cliente può riservare più tavoli, questo giustifica le cardinalità utilizzate.

Si presuppone che dato un tavolo prenotato in una certa data e ora, una nuova prenotazione non possa essere effettuata per lo stesso tavolo a meno che non siano passate almeno 3 ore dalla precedente prenotazione.

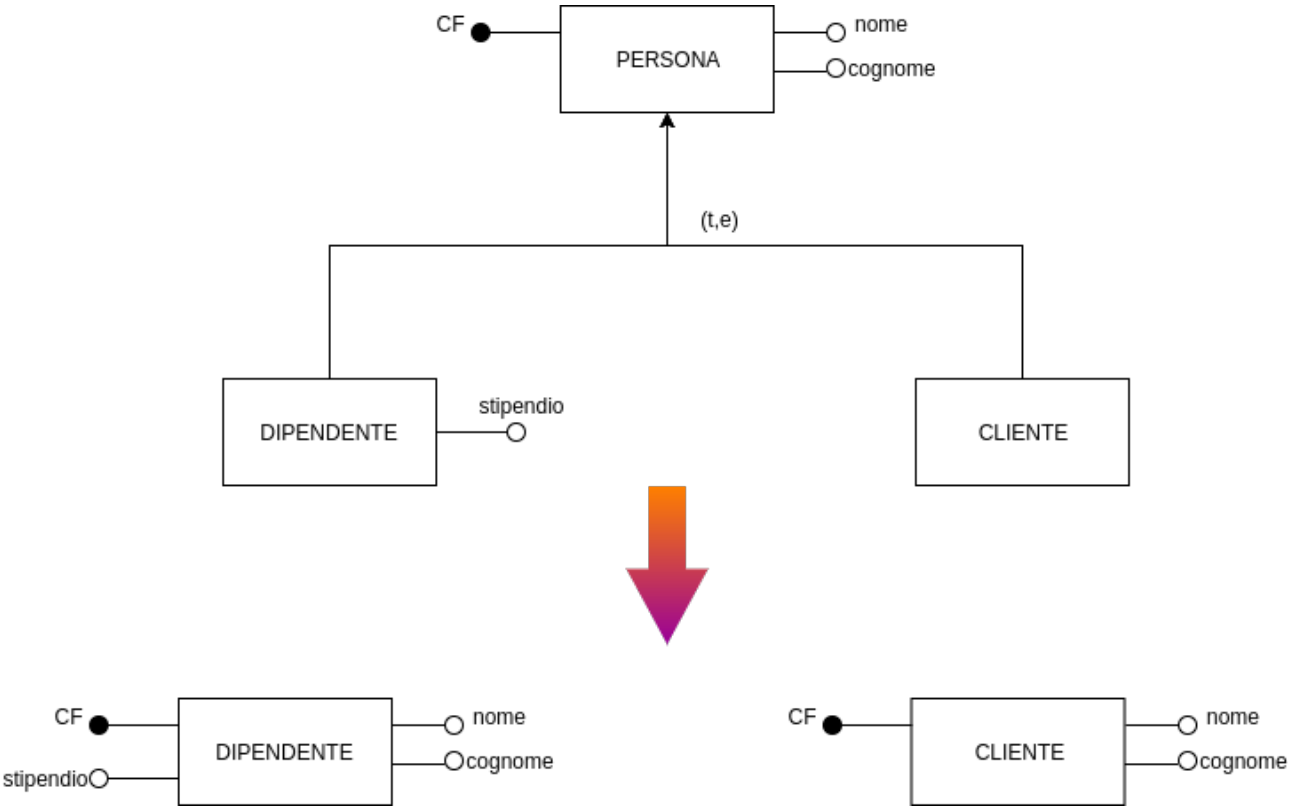
Schema Entità-Relazioni completo



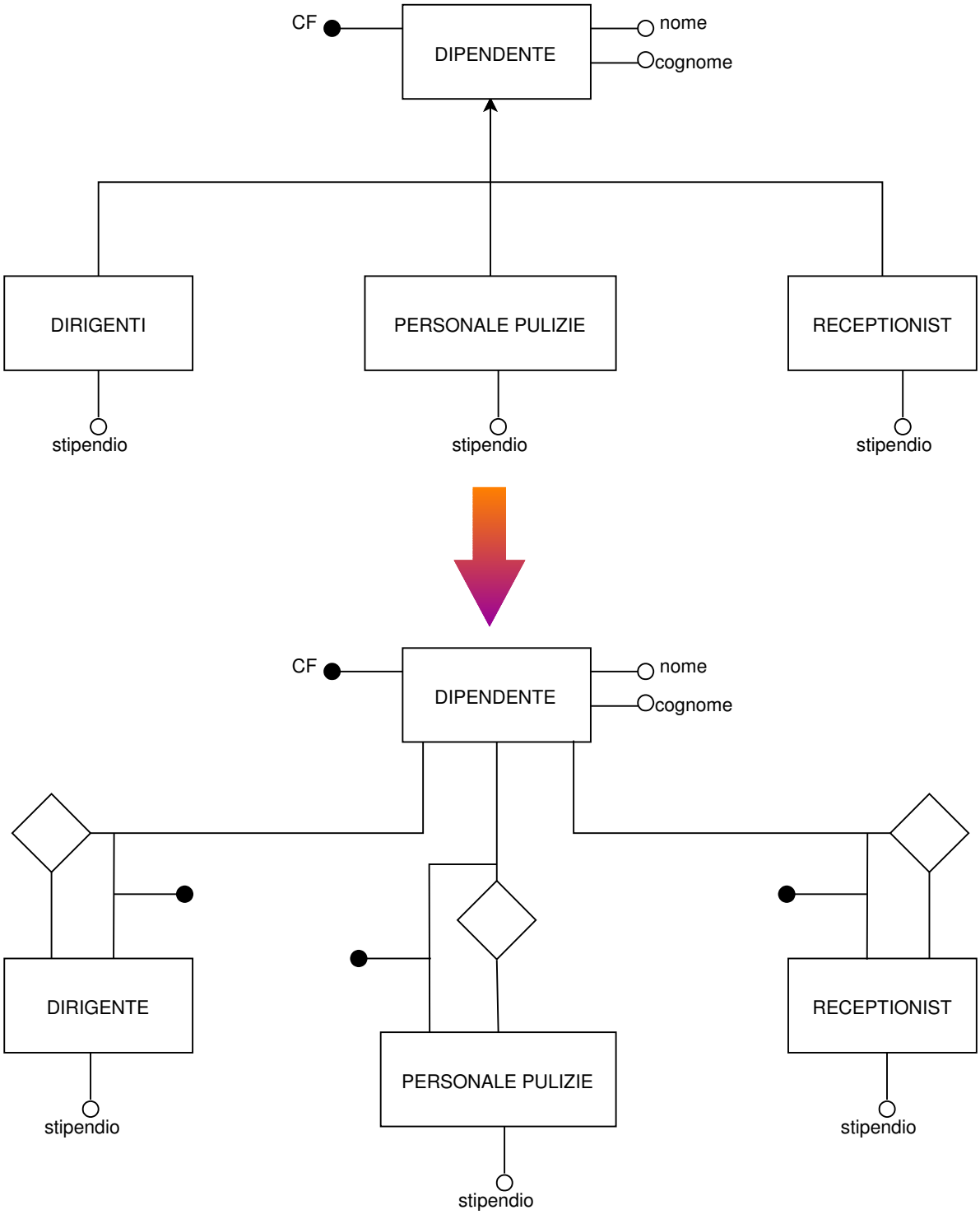
Da schema ER a Progetto logico

- *Eliminazione gerarchie ISA*

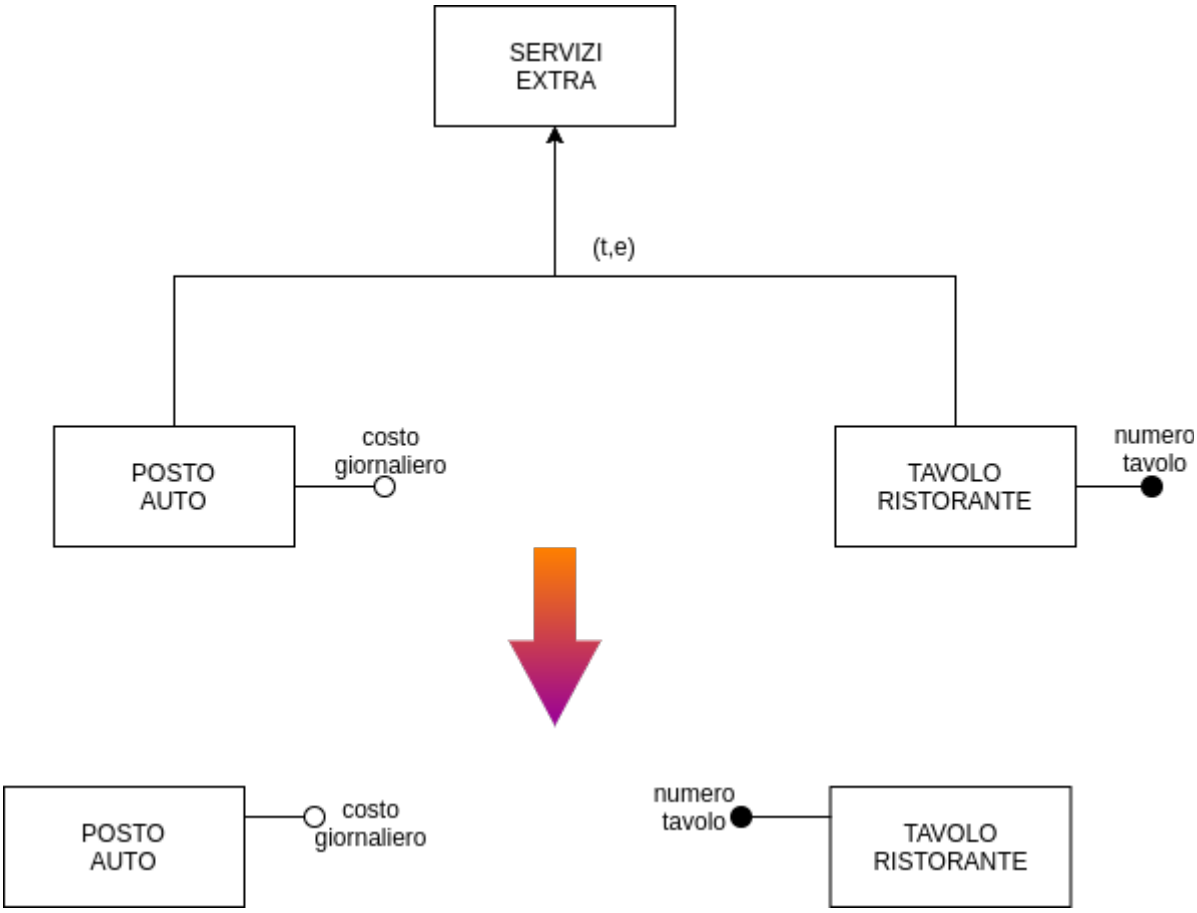
Nella base di dati sono presenti 3 gerarchie ISA: la prima è l'entità *Persona* che si divide in *Dipendente* o *Cliente*. La gerarchia è di tipo totale ed esclusivo, ovvero una persona deve essere un dipendente oppure un cliente, ma non entrambe. Si effettua quindi un collasso verso il basso (reso possibile in quanto si tratta di una relazione totale ed esclusiva), si elimina l'entità padre mantenendo i due figli.
Nel caso in cui un dipendente voglia soggiornare nell'Hotel come cliente, si duplicherà la persona dipendente dentro l'entità *Cliente*, questo perché si suppone che una situazione del genere possa accadere molto di rado, inoltre risulterà molto più comodo accedere solo all'entità *Cliente*.
Questa possibile ridondanza giustifica l'esclusività della gerarchia.



La seconda gerarchia è quella tra l'entità padre Dipendente e le entità figlie Personale Pulizie, Dirigente e Receptionist. In questo caso la gerarchia è di tipo non totale ed esclusivo, poiché si suppone che possano esistere alcuni dipendenti "speciali" che non facciano parte di nessuna delle entità figlie (per esempio un dipendente esterno, uno studente in stage ecc..) e per i quali non interessa creare un'entità dedicata. Si opta per il mantenimento di tutte le entità al posto del collasso verso l'alto (che sarebbe più funzionale per accedere all'attributo stipendio) per evitare il numero di valori nulli nel selettore e per mantenere la comodità nelle relazioni tra *Personale Pulizie* - *Turno Pulizie* e *Receptionist – Turno Reception*.



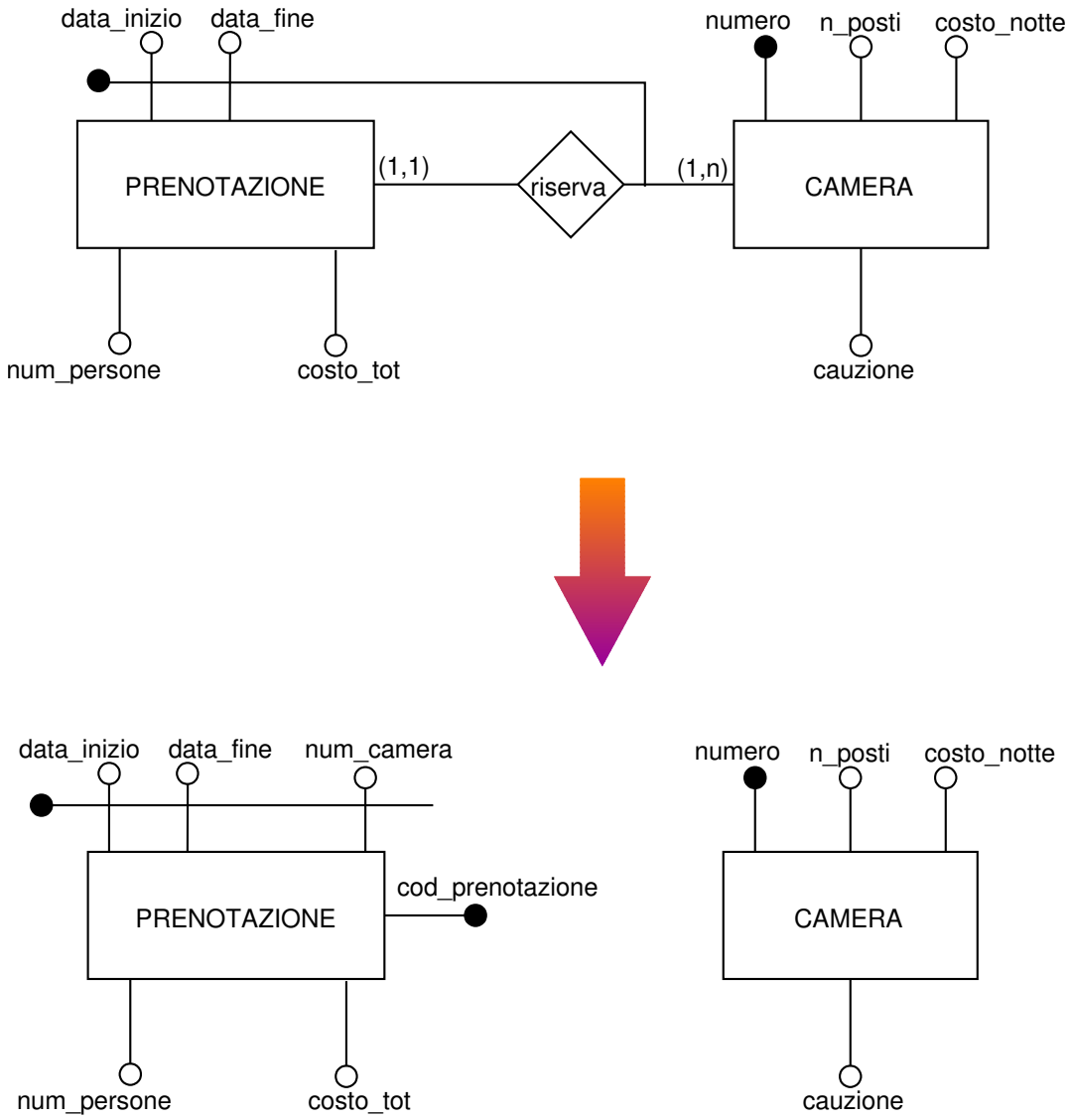
L'ultima gerarchia da eliminare è quella con padre *Servizio Extra* e figli *Ristorante* e *Posto Auto*. Anche in questo caso la gerarchia è di tipo totale ed esclusiva, perciò si effettua un collasso verso il basso tramite l'eliminazione dell'entità *Servizio Extra*. In questo modo sarà facilitato l'accesso separato alle entità figlie.



- Eliminazione delle Foreign Keys e selezione delle Private Keys

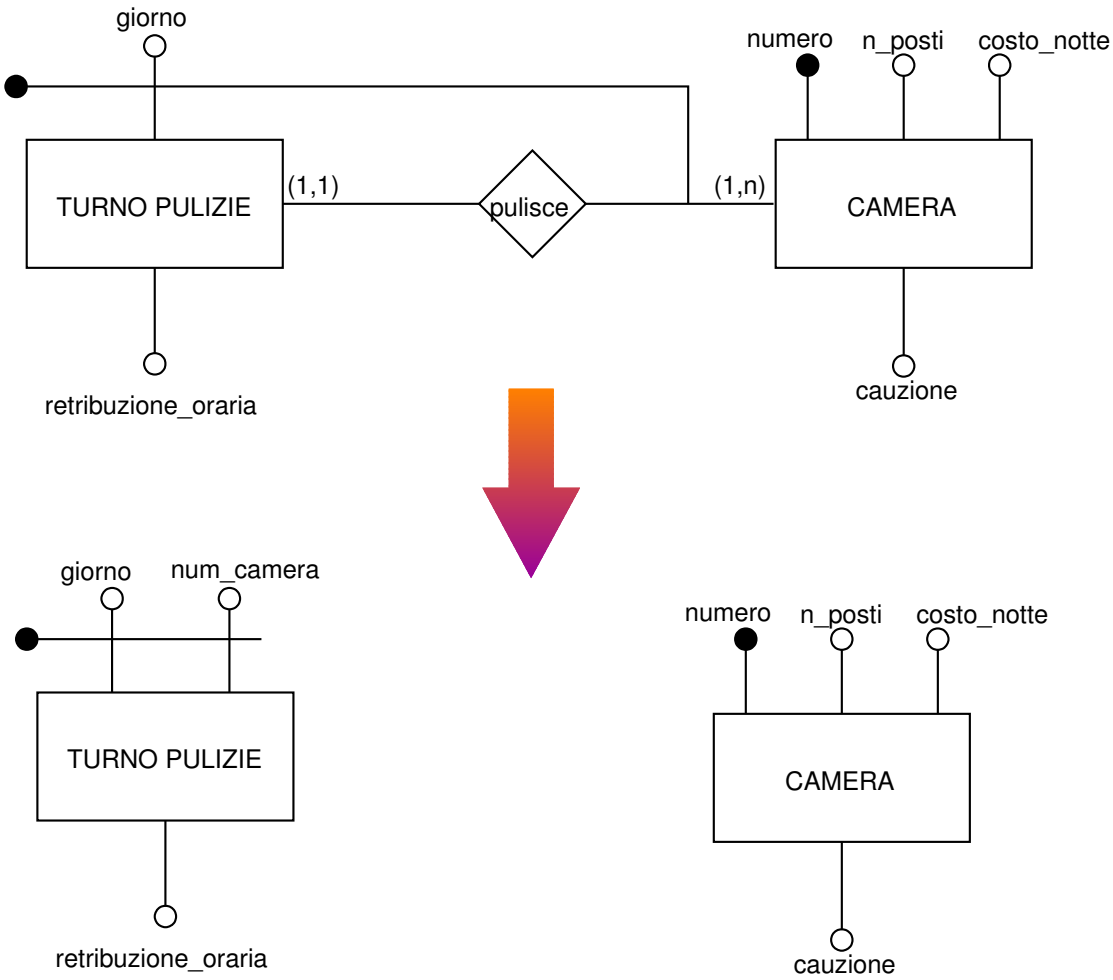
Nello schema ER sono presenti diverse entità caratterizzate da una componente di identificazione esterna, ossia entità dove la chiave primaria è composta da attributi dell'entità stessa e da uno o più attributi di un'altra entità. Seppure questa realizzazione sia molto flessibile, per poter realizzare il progetto logico vero e proprio sono necessarie alcune modifiche.

Prenotazione – Camera



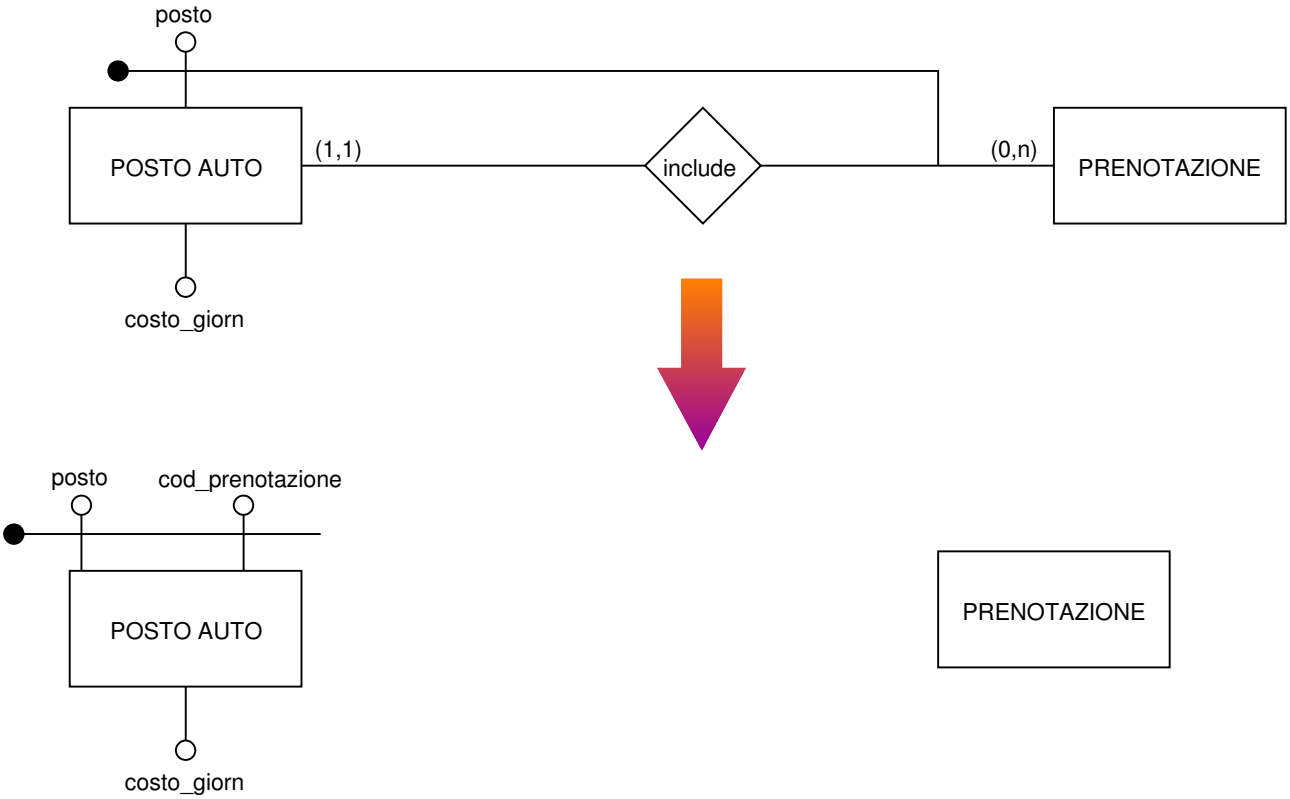
In questo caso viene creato un nuovo attributo in *Prenotazione,* il quale serve sia per comporre la chiave primaria, sia per stabilire il collegamento con la camera.
Essenzialmente *num_camera* funge sia da parte di chiave primaria, sia da chiave esterna.
Un trigger si occuperà di gestire che una certa camera, in un certo periodo non sia già stata riservata da un altro cliente.
Viene anche inserita una sorta di chiave secondaria fittizia (cod_prenotazione). Questo per rendere più semplice ed leggibile il collegamento con altre entità subordinate.
Ogni valore assunto dalla chiave secondaria è univoco.

Camera - Turno



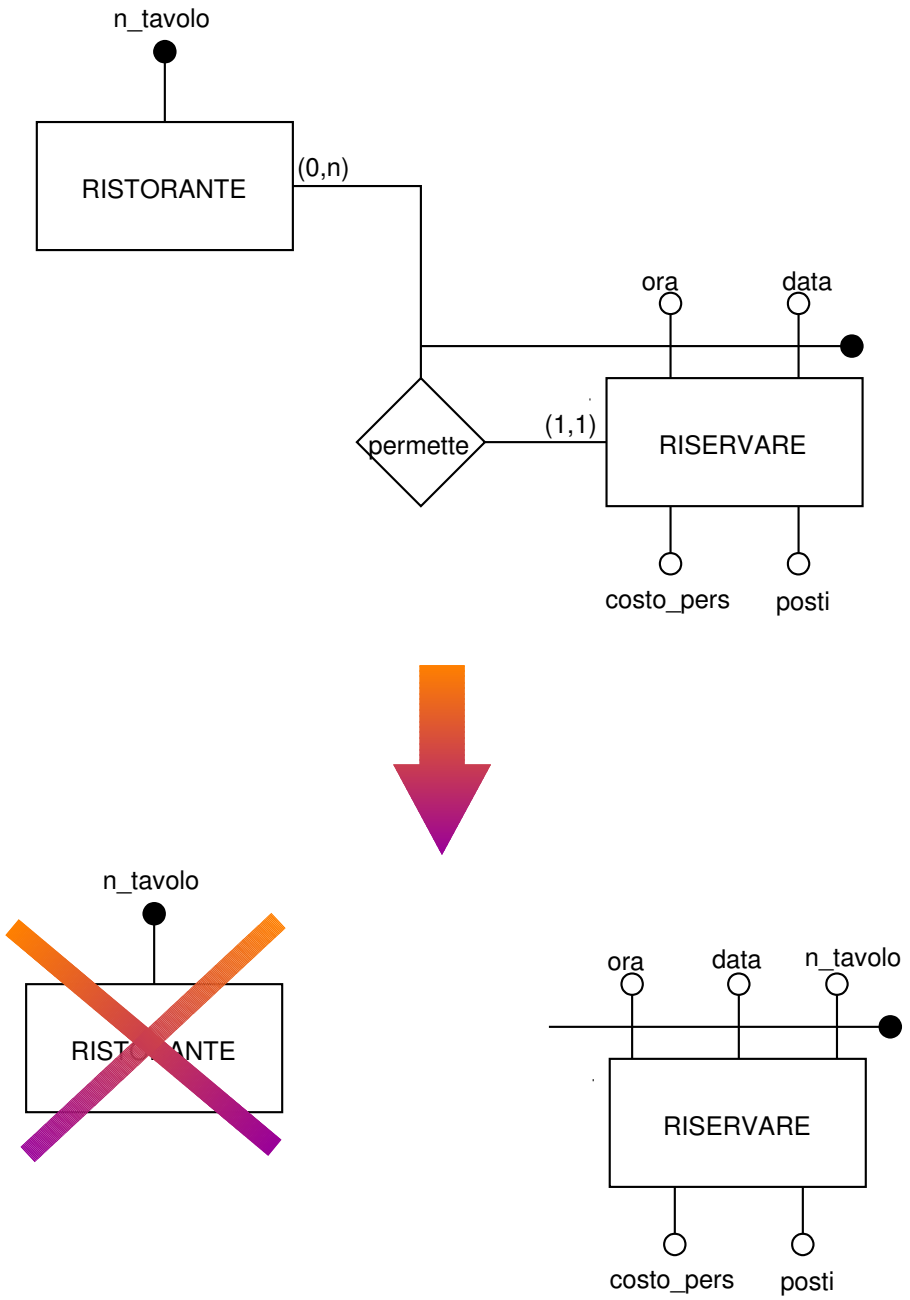
In maniera simile viene gestita la situazione fra queste due entità. Anche qui viene creato in *Turno pulizie* un nuovo attributo utilizzato sia per la composizione della private key, sia come foreign key, per il collegamento a *Camera*.

Prenotazione – Posto auto



Anche in questo caso la chiave primaria di *Posto auto* viene modificata, componendola con quella dell'entità *Prenotazione*.
In questo caso, la chiave primaria di *Posto auto* è generata sfruttando la chiave secondaria di *Prenotazione*. Questo per favorire una miglior chiarezza e leggibilità delle entità: si sarebbe potuto usare allo stesso modo le entità che compongono la chiave primaria di *Prenotazione*, ottenendo lo stesso risultato a livello pratico, tuttavia la leggibilità dell'immagine e la gestione delle chiave esterne ne avrebbe sofferto molto.

Ristorante - Riservare

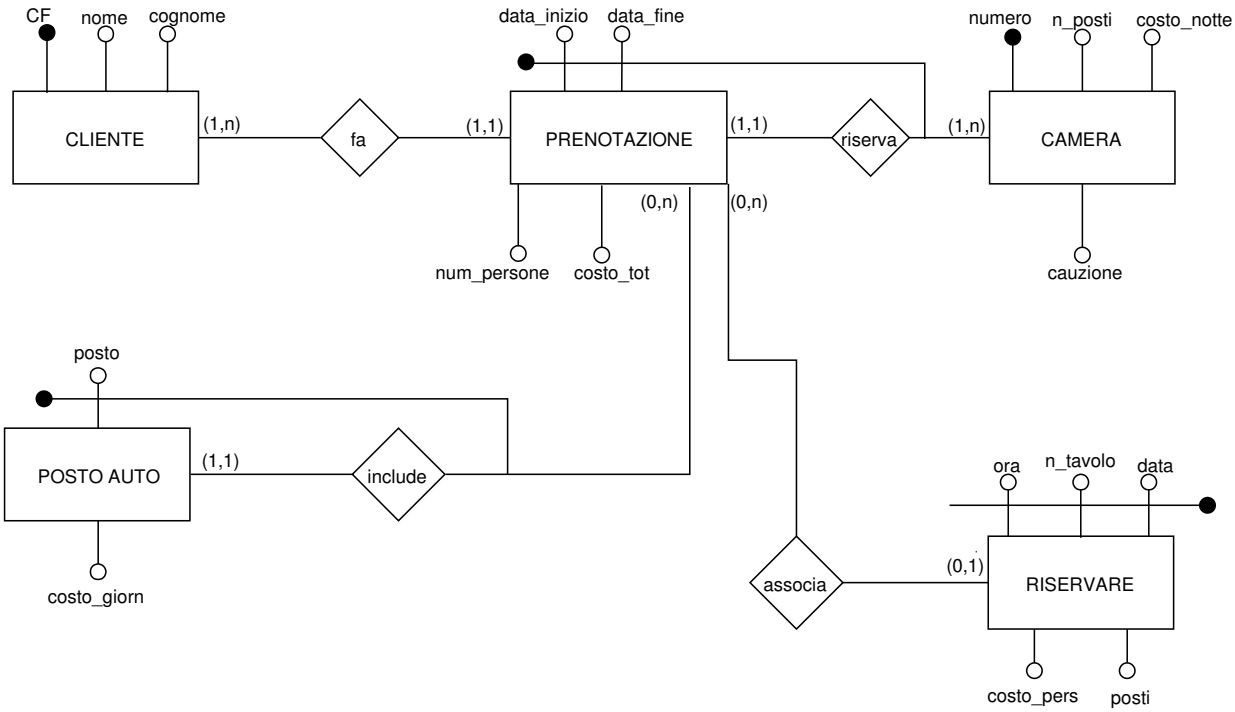


Anche qui viene aggiunto un nuovo attributo in *Riservare,* usato come chiave primaria e chiave esterna.
La presenza del numero del tavolo in *Riservare* rende *Ristorante* alquanto ridondante, in quanto non contiene alcuna altra informazione, si è quindi optato per eliminare quest'ultima entità.

- Valutazione dei dati derivati

Come precedentemente accennato, nello schema logico sono presenti dati derivati.
Come sempre **il costo di una Scrittura è uguale al costo di 2 Letture.**

Costo_tot (Prenotazione)



Il dato derivato è ***costo_tot*** in *Prenotazione*. Rappresenta il costo complessivo di tutti i servizi utilizzati da un cliente durante il soggiorno in albergo, e il costo della stanza. Senza di esso, i costi possono sempre essere calcolati singolarmente, ad esempio: **costo camera** = **costo_notte * (data_fine) – (data_inizio).**

Operazioni di cui tenere conto:

1. Aggiunta di una prenotazione, fatta da un cliente (che si suppone sia già stato inserito nel database) per una certa camera (si suppone che la camera non sia occupata e il trigger di creazione vada a buon fine). (**Op. 1**)
2. Aggiunta di un posto auto (si supponga **nota** e valida la prenotazione). (**Op. 2**)
3. Aggiunta di un tavolo al ristorante (si supponga **nota** e valida la prenotazione). (**Op. 3**)
4. Recuperare il costo totale di una prenotazione, che si suppone sia valida. (**Op. 4**)

I valori associati a volumi e frequenze sono **valori arbitrari**, da noi forniti, scelti in modo speculativo ma ragionevole per procedere a un calcolo del costo del dato derivato che sia sensato e vicino alla realtà.
Per la selezione dei valori si è tenuto conto di certi criteri:

- Si tiene conto delle **cardinalità**.
- Si è deciso che il numero di prenotazioni a un tavolo è maggiore del numero di posti auto prenotati.
- Si presuppone che le operazioni giornaliere effettuate con maggiore frequenza siano, nell'ordine: **Op. 4**, **Op. 3**, **Op. 1**, **Op. 2**.

Tabella dei Volumi:

CONCETTO	TIPO	VOLUME
Cliente	E	150
Prenotazione	E	100
Camera	E	120
P. Auto	E	300
Riservare	E	610
Fa	R	100
Riserva	R	100
Include	R	300
Associa	R	600

Tabella delle Operazioni:

OPERAZIONI	TIPO	FREQUENZA
Op 1	1	50/g
Op 2	1	40/g
Op 3	1	80/g
Op 4	1	100/g

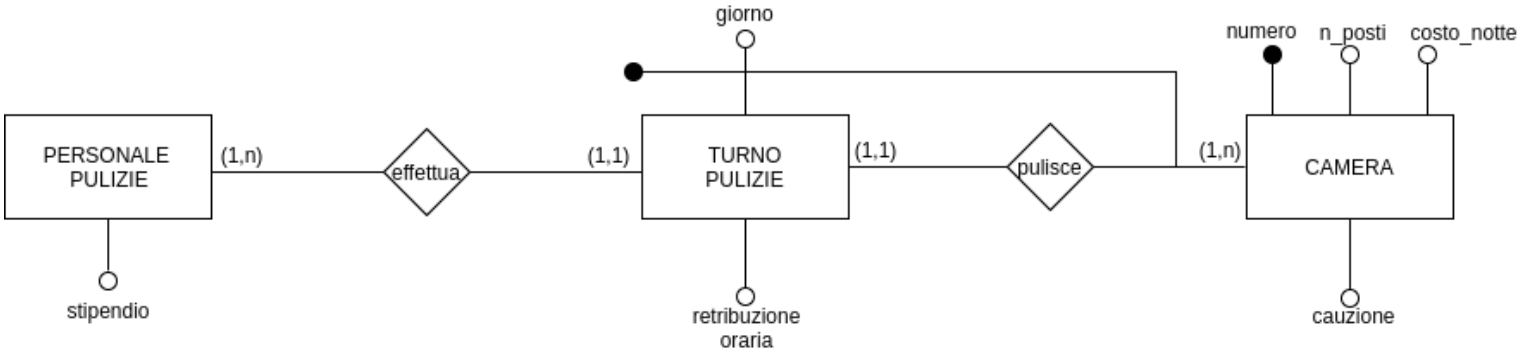
Calcolo dei Costi di Accesso:

Con il dato derivato:				
	Concetto	**Accesso**	**Tipo**	**Costo /g**
Op. 1	Camera	1	Lettura	**400**
	Riserva	1	Scrittura	
	Prenotazione	1	Scrittura	
	Cliente	1	Lettura	
	Fa	1	Scrittura	
Op. 2	Posto auto	1	Scrittura	**280**
	Include	1	Scrittura	
	Prenotazione	1	Lettura	
	Prenotazione	1	Scrittura	
Op. 3	Riservare	1	Scrittura	**560**
	Associa	1	Scrittura	
	Prenotazione	1	Lettura	
	Prenotazione	1	Scrittura	
Op. 4	Prenotazione	1	Lettura	**100**
Costo totale /g :				**1340**

Senza il dato derivato:				
	Concetto	**Accesso**	**Tipo**	**Costo /g**
Op. 1	Camera	1	Lettura	**400**
	Riserva	1	Scrittura	
	Prenotazione	1	Scrittura	
	Cliente	1	Lettura	
	Fa	1	Scrittura	
Op. 2	Posto auto	1	Scrittura	**160**
	Include	1	Scrittura	
Op. 3	Riservare	1	Scrittura	**320**
	Associa	1	Scrittura	
Op. 4	Prenotazione	1	Lettura	**2100**
	Riserva	1	Lettura	
	Camera	1	Lettura	
	Include	3	Lettura	
	Posto auto	3	Lettura	
	Associa	6	Lettura	
	Riserva	6	Lettura	
Costo totale /g :				**2980**

Come possiamo vedere, con i dati da noi forniti risulta che i costi di accesso giornalieri siano notevolmente minori con la presenza del dato derivato, che verrà quindi lasciato nel progetto finale. Ovviamente non si deve dimenticare che sono stati usati dati puramente speculativi, non è impossibile che con dati estratti da un database reale, la presenza di un dato derivato possa risultare ridondante e costosa. Nel caso mostrato il dato derivato sarà mantenuto.

Stipendio (Personale Pulizie)



Stipendio è il dato derivato di cui valutare il costo. Anche qui i dati utilizzati per volumi e frequenze sono **valori arbitrari**, da noi forniti, scelti in modo speculativo ma ragionevole per procedere a un calcolo del costo del dato derivato che sia sensato e vicino alla realtà.

Operazioni di cui tenere conto:

1. Aggiunta di un turno, essendo noto il dipendente (si suppone nota e valida la camera associata al turno). (**Op. 1**)
2. Visualizzazione dello stipendio mensile finale, noto il dipendente. (**Op. 2**)

Criteri di cui tenere conto nella scelta dei volumi e delle frequenze di accesso:

- I turni di pulizia sono sicuramente molto **maggiori** del personale che li esegue.
- L'operazione 1 è sicuramente eseguita più spesso dell'operazione 2, che in teoria viene soprattutto eseguita a fine mese.

Tabella dei Volumi:

CONCETTO	TIPO	VOLUME
Personale Pul.	E	50
Turno Pul.	E	250
Effettua	R	250

Tabella delle Frequenze:

OPERAZIONI	TIPO	FREQUENZA
Op 1	1	60/g
Op 2	1	20/g

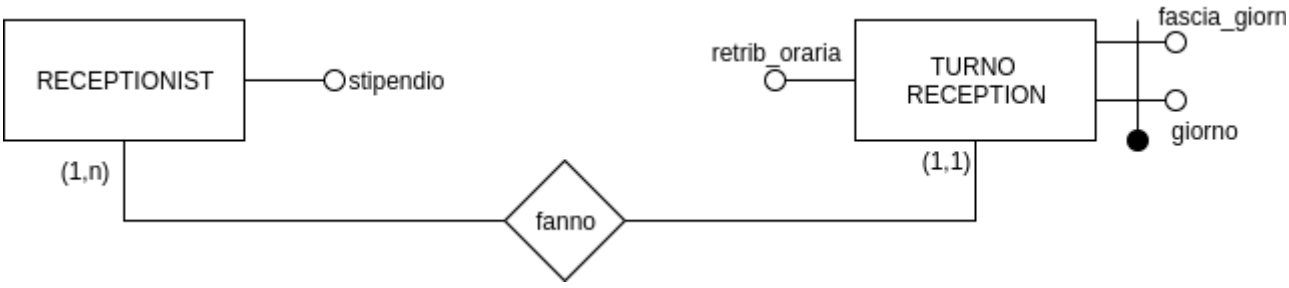
Calcolo dei Costi di Accesso:

Con il dato derivato:				
	Concetto	**Accesso**	**Tipo**	**Costo /g**
Op. 1	Turno Pulizie	1	Scrittura	**420**
	Effettua	1	Scrittura	
	Personale Pulizie	1	Lettura	
	Personale Pulizie	1	Scrittura	
Op. 2	Personale Pulizie	1	Lettura	**20**
Costo totale /g:				**440**

Senza dato derivato:				
	Concetto	**Accesso**	**Tipo**	**Costo /g**
Op. 1	Turno Reception	1	Scrittura	**240**
	Fanno	1	Scrittura	
Op. 2	Receptionist	1	Lettura	**220**
	Fanno	5	Lettura	
	Turno Reception	5	Lettura	
Costo totale /g:				**460**

Anche in questo caso i costi di accesso sono inferiori con il dato derivato, che quindi verrà mantenuto.

Stipendio (Reception)



Stipendio è il dato derivato di cui valutare il costo. Anche qui i dati utilizzati per volumi e frequenze sono **valori speculativi**, da noi forniti, scelti in maniera ragionevole per procedere a un calcolo del costo del dato derivato che sia sensato e vicino alla realtà.
Si presuppone che lo stipendio sia mensile, e che venga azzerato all'inizio di ogni mese tramite un'opportuna query. Uno stipendio è dato dalla somma delle retribuzioni di tutti i turni fatti in un mese.

Operazioni di cui tenere conto:

1. Inserimento di un nuovo turno, essendo **noto** il receptionist.
2. Visualizzazione dello stipendio, essendo **noto** il receptionist.

Criteri di cui si è tenuto conto:

- Il numero di receptionist è sicuramente **minore** del numero di turni.

Tabella dei Volumi:

CONCETTO	TIPO	VOLUME
Receptionist	E	20
Turno Pul.	E	100
Fanno	R	100

Tabella delle Frequenze:

OPERAZIONI	TIPO	FREQUENZA
Op 1	1	20/g
Op 2	1	50/g

Calcolo dei costi di accesso:

Con il dato derivato:				
	Concetto	**Accesso**	**Tipo**	**Costo /g**
Op. 1	Turno Reception	1	Scrittura	**140**
	Fanno	1	Scrittura	
	Receptionist	1	Lettura	
	Receptionist	1	Scrittura	
Op. 2	Receptionist	1	Lettura	**50**
Costo totale /g:				**190**

Senza dato derivato:				
	Concetto	**Accesso**	**Tipo**	**Costo /g**
Op. 1	Turno Reception	1	Scrittura	**80**
	Fanno	1	Scrittura	
Op. 2	Receptionist	1	Lettura	**550**
	Fanno	5	Lettura	
	Turno Reception	5	Lettura	
Costo totale /g:				**630**

Come possiamo vedere, anche in questo caso la presenza di un dato derivato risulta vantaggiosa.

- Verifica di normalizzazione.

A seguito di un'attenta analisi non sono state reputate necessarie alcune modifiche di normalizzazione.

Schema Logico finale

Dipendente (CF, Nome, Cognome)

Sconto (CodOfferta, descrizione, percentuale)

Cliente (CF, Nome, Cognome, CodOfferta)
FK: CodOfferta references Sconto

Dirigente (CF, Nome, Cognome, Stipendio)
FK: CF references Dipendente

PersonalePulizie (CF, Nome, Cognome, Stipendio, CFDirigente)
FK: CF references Dipendente
FK : CFDirigente references Dirigente

Receptionist (CF, Nome, Cognome, Stipendio, CFDirigente)
FK: CF references Dipendente
FK : CFDirigente references Dirigente

TurnoReception (Giorno, FasciaGiornaliera, RetribuzioneOraria, CF, CFSostituto)
FK: CF references Receptionist
FK: CFSostituto references Receptionist

Camera(NumeroCamera, NPosti, CostoNotte, Cauzione)

TurnoPulizie (Giorno, NumeroCamera, RetribuzioneOraria, CF, CFSostituto)
FK: NumeroCamera references Camera
FK: CF references PersonalePulizie
FK: CFSostituto references PersonalePulizie

PromozioneGiornaliera (DataPromozione, Percentuale, Titolo)

Prenotazione (CodPrenotazione, DataInizio, DataFine, NumeroCamera,
NumPersone, CostoTot, DataPromozione, CF)
FK: NumeroCamera references Camera
FK: DataPromozione references PromozioneGiornaliera
FK: CF references Cliente
AK: CodPrenotazione

PostoAuto (Posto, CodPrenotazione, CostoGiornaliero)
FK: CodPrenotazione references Prenotazione

Riservare (Data, Ora, NumeroTavolo, Posti, CostoPersona, CodPrenotazione)
FK: NumeroTavolo references Ristorante
FK: CodPrenotazione references Prenotazione

Vincoli Aggiuntivi

Di seguito vengono riportati tutti quei vincoli che non possono essere gestiti direttamente dal DataBase o tramite la progettazione concettuale. I più semplici vengono gestiti tramite appositi *check* nelle query di creazione, i più complessi invece verranno gestiti attraverso trigger.

Check:

- Le sezioni della giornata per i turni di pulizia e/o reception possono essere: (Mattino, Pomeriggio, Sera, Notte).
- Lo stipendio di un qualsiasi dipendente deve essere compreso tra 0 e 999.999.
- La percentuale dello sconto o della promozione giornaliera deve essere compresa tra 1 e 99.
- La retribuzione oraria nei turni per la reception e per le pulizie deve essere maggiore o uguale a zero.
- Il numero di ogni camera deve essere un numero positivo.
- I posti letto di una camera devono essere maggiori di zero.
- Il costo per notte di una camera non può essere minore di zero.
- La cauzione per una camera deve essere compresa tra 0 e 1.000.
- La data iniziale di una prenotazione deve essere minore della data finale.
- Il numero di persone relativo ad una prenotazione deve essere maggiore di zero.
- Il costo totale di una prenotazione deve essere maggiore o uguale a zero.
- Il numero di ogni posto auto deve essere positivo.
- Il costo giornaliero di un posto auto deve essere maggiore o uguale a zero.
- Il numero di ogni tavolo del ristorante deve essere positivo.
- Il numero dei posti per riservare un tavolo al ristorante deve essere positivo.
- Il costo a persona nel riservare un tavolo al ristorante maggiore o uguale a zero.

Trigger:

1) Ogni camera può avere al massimo una prenotazione nello stesso lasso di tempo, ovvero non posso prenotare una camera che risulta già occupata.

2) Nel momento della prenotazione vengono specificati il numero di ospiti. Questo numero non può superare la "capienza" della camera, ovvero i posti letto disponibili.

3) Sia lo sconto associato al cliente che lo sconto derivante dalla promozione giornaliera vengono calcolati solamente sul prezzo della camera, escludendo i costi extra del soggiorno.
Ai clienti che hanno effettuato la prenotazione in una certa data (corrispondente al giorno in cui la prenotazione viene inserita nel DataBase) verrà applicata la relativa percentuale, se questa data è associata ad una promozione giornaliera.
Ogni prenotazione può avere al massimo uno dei due sconti (si considera la percentuale più alta).

4) Un dipendente della reception può effettuare massimo un turno al giorno.

5) Aggiornare il dato derivato "CostoTot" ogni volta che viene prenotato un posto auto.

6) Aggiornare il dato derivato "CostoTot" ogni volta che viene prenotato un tavolo al ristorante.

7) Aggiornare lo stipendio del personale pulizie ogni volta che viene aggiunto un turno.

8) Aggiornare lo stipendio dei Receptionist ogni volta che viene aggiunto un turno

9) Ogni volta che viene prenotato un tavolo, controllare che lo stesso tavolo non sia prenotato nella stessa fascia oraria (Devono passare almeno 3 ore per poter prenotare di nuovo lo stesso tavolo).

10) Ogni volta che viene prenotato un posto auto, assicurarsi che tale posto sia libero per tutta la durata della prenotazione relativa.

11) Accertarsi che un dipendente pulisca al massimo n camere in una giornata (ad esempio 6, dando per scontato che ogni camera occupi un'ora)

Di seguito verranno presentate tutte le parti di codice necessarie alla creazione e popolazione del DataBase. Inoltre vengono presentate una serie di operazioni, a titolo esemplificativo, di interrogazioni al DataBase, alcune delle quali sono operazioni citate sopra, altre sono state ritenute utili al fine di simulare un utilizzo realistico del DataBase stesso.
Per effettuare l'implementazione fisica fare riferimento al file **README.txt**.

Query di Creazione

Creazione Dominio:

```
create domain SezioniGiornata
	as varchar(10)
check
(
	value='Mattino' OR
	value='Pomeriggio' OR
	value='Sera' OR
	value='Notte'
/*
	Mattino = 7:00 - 13:00
	Pomeriggio = 13:00 -19:00
	Sera = 19:00 - 1:00
	Notte = 1:00 - 7:00
*/
);
```

Creazione Vista:

```
create view Stipendi as
	(select Nome,Cognome,Stipendio
	from Receptionist)
union
	(select Nome,Cognome,Stipendio
	from PersonalePulizie)
order by Cognome
```

Creazione Tabelle:

```
create table Dipendente
(
	CF char(16) primary key,
	Nome varchar(30) not null,
	Cognome varchar(30) not null
);



create table Sconto
(
```

```
	CodOfferta char(5) primary key,
	Descrizione varchar(100),
	Percentuale smallint not null,

	check ((Percentuale>0) and (Percentuale<100))
);



create table Cliente
(
	CF char(16) primary key,
	Nome varchar(30) not null,
	Cognome varchar(30) not null,
	CodOfferta char(5) unique,

	foreign key (CodOfferta) references Sconto(CodOfferta)
		on delete no action on update cascade
);



create table Dirigente
(
	CF char(16) primary key,
	Nome varchar(30) not null,
	Cognome varchar(30) not null,
	Stipendio numeric(8,2) not null,

	check ((Stipendio>=0) and (Stipendio<=999999)),

	foreign key (CF) references Dipendente(CF)
		on delete no action on update cascade
);



create table PersonalePulizie
(
	CF char(16) primary key,
	Nome varchar(30) not null,
	Cognome varchar(30) not null,
	Stipendio numeric(8,2) not null,
	CFDirigente char(16) not null,

	check ((Stipendio>=0) and (Stipendio<=999999)),

	foreign key (CFDirigente) references Dirigente(CF)
		on delete no action on update cascade,
```

```
	foreign key (CF) references Dipendente(CF)
		on delete no action on update cascade
);



create table Receptionist
(
	CF char(16) primary key,
	Nome varchar(30) not null,
	Cognome varchar(30) not null,
	Stipendio numeric(8,2) not null,
	CFDirigente char(16) not null,

	foreign key (CFDirigente) references Dirigente(CF)
		on delete no action on update cascade,

	foreign key (CF) references Dipendente(CF)
		on delete no action on update cascade
);



create table TurnoReception
(
	Giorno date,
	FasciaGiornaliera SezioniGiornata,
	RetribuzioneOraria numeric(4,2),
	CF char(16) not null,
	CFSostituto char(16),

	check(RetribuzioneOraria >=0),

	primary key (Giorno, FasciaGiornaliera),

	foreign key (CF) references Receptionist(CF)
		on delete no action on update cascade,

	foreign key (CFSostituto) references Receptionist(CF)
		on delete no action on update cascade
);



create table Camera
(
	NumeroCamera smallint primary key,
	NPosti smallint not null,
	CostoNotte numeric(6,2) not null,
	Cauzione smallint not null,
```

```
        check ( (NumeroCamera>0) and (NPosti>0) and (CostoNotte>=0) and
        (Cauzione>=0) and (Cauzione<=1000) )
);


create table TurnoPulizie
(
        Giorno date,
        NumeroCamera smallint,
        RetribuzioneOraria numeric(4,2) not null,
        CF char(16) not null,
        CFSostituto char(16),

        check(RetribuzioneOraria >=0),

        primary key (Giorno,NumeroCamera),

        foreign key (NumeroCamera) references Camera(NumeroCamera)
                on delete no action on update cascade,

        foreign key (CF) references PersonalePulizie(CF)
                on delete no action on update cascade,

        foreign key (CFSostituto) references PersonalePulizie(CF)
                on delete no action on update cascade
);


create table PromozioneGiornaliera
(
        DataPromozione date primary key,
        Percentuale smallint not null,
        Titolo varchar(50) not null,

        check ((Percentuale>0) and (Percentuale<100))
);


create table Prenotazione
(
        CodPrenotazione char(8) not null unique,
        /*
        L'attributo CodPrenotazione è chiave secondaria. La chiave
        composta rimane primaria poiché contiene informazioni
        importanti, mentre questa chiave secondaria viene utilizzata per comodità
nelle Query di accesso al DataBase.
```

```
        */
        DataInizio date not null,
        DataFine date not null,
        Numerocamera smallint not null,
        NumPersone smallint not null,
        CostoTot numeric (7,2) not null,
        DataPromozione date,
        CF char(16) not null,

        check ( (DataInizio<DataFine) and (NumPersone>0) and (CostoTot>=0) ),

        primary key (DataInizio, DataFine, NumeroCamera),

        unique (CodPrenotazione),

        foreign key (NumeroCamera) references Camera(NumeroCamera)
                on delete no action on update cascade,

        foreign key (DataPromozione) references
        PromozioneGiornaliera(DataPromozione)
                on delete no action on update cascade,

        foreign key (CF) references Cliente(CF)
                on delete no action on update cascade
);



create table PostoAuto
(
        Posto smallint,
        CodPrenotazione char(8),
        CostoGiornaliero numeric(4,2),

        check( (Posto>0) and (CostoGiornaliero>=0) ),

        primary key (Posto, CodPrenotazione),

        foreign key (CodPrenotazione) references Prenotazione(CodPrenotazione)
                on delete no action on update cascade
);


create table Riservare
(
        Data date,
        Ora time,
        NumeroTavolo smallint,
        Posti smallint not null,
        CostoPersona numeric(5,2),
```

```
	CodPrenotazione char (8) not null,

	check( (Posti>0) and (CostoPersona>=0) ),

	primary key(Data, Ora, NumeroTavolo),

	foreign key (NumeroTavolo) references Ristorante(NumeroTavolo)
		on delete no action on update cascade,

	foreign key (CodPrenotazione) references Prenotazione(CodPrenotazione)
		on delete no action on update cascade
);
```

Trigger

Trigger 1)

```
/*
Controllo date camera: la camera prenotata deve essere libera in quel lasso di tempo.
*/

create function controllo_camera()
returns trigger as $controllo_camera$

	begin
		if ( select count(*)
			from (select new.DataInizio, new.DataFine
				from Prenotazione p
				where exists
					(
					select *
					from Prenotazione p2
					where (((p.DataInizio >= p2.DataInizio)
						and (p.DataInizio <= p2.DataFine))
						or
						((p.DataFine >= p2.DataInizio)
						and (p.DataFine <= p2.DataFine)))
					and p2.NumeroCamera = p.NumeroCamera
					and p2.CodPrenotazione != p.CodPrenotazione)
				) as c ) > 0
		then
			raise exception 'La Prenotazione non può essere effettuata, la
camera selezionata risulta già occupata. Si consiglia di cambiare camera o modificare le
date selezionate.';
		end if;
	return new;
	end;
```

```
$controllo_camera$ language plpgsql;

create trigger controllo_camera
after insert or update on Prenotazione
for each row
execute procedure controllo_camera();
```

Trigger 2)

```
/*
Controllo ospiti camera: il numero di ospiti deve essere <= al numero di posti della
camera.
*/

create function controllo_ospiti()
returns trigger as $controllo_ospiti$
	begin
		if new.NumPersone >
			(select NPosti
			from Camera
			where NumeroCamera = new.NumeroCamera)
		then
			raise exception 'Numero ospiti errato!';
		end if;
	return new;
	end;
$controllo_ospiti$ language plpgsql;

create trigger controllo_ospiti
before insert on Prenotazione
for each row
execute procedure controllo_ospiti();
```

Trigger 3)

```
/*
Funzione che inserisce il costo della camera nel costo totale. Se la prenotazione o il
cliente hanno uno sconto, viene applicato il massimo dei due.
*/

create function costo_camera()
returns trigger as $costo_camera$
```

```
declare
	n_notti smallint;
	data_inizio date;
	data_fine date;
	costo numeric(7,2);
	perc_prenotazione smallint;
	perc_cliente smallint;
	perc real;

begin
	data_inizio := (select DataInizio
		from Prenotazione
		where CodPrenotazione = new.CodPrenotazione);
	data_fine := (select DataFine
		from Prenotazione
		where CodPrenotazione = new.CodPrenotazione);
	n_notti := data_fine - data_inizio;
	costo := (select CostoNotte
		from Camera
		where NumeroCamera = new.NumeroCamera);
	perc_prenotazione := (select PG.Percentuale
		from Prenotazione as P, PromozioneGiornaliera as PG
		where P.CodPrenotazione = new.CodPrenotazione
		and P.DataPromozione = PG.DataPromozione);
	perc_cliente := (select S.Percentuale
		from Prenotazione as P, Cliente as C, Sconto as S
		where P.CodPrenotazione = new.CodPrenotazione
		and P.CF = C.CF
		and C.CodOfferta = S.CodOfferta);

	if ((perc_prenotazione <> 0) and (perc_cliente <> 0)) then
		if (perc_cliente > perc_prenotazione) then
			perc := perc_cliente;
		else
			perc := perc_prenotazione;
		end if;
	else
		if (perc_cliente <> 0) then
			perc := perc_cliente;
		else
			perc := perc_prenotazione;
		end if;
	end if;

	if (perc <> 0)
	then
		costo := costo * n_notti;
		perc := 1 - perc/100;
		update Prenotazione
		set CostoTot = costo * perc
```

```
                where CodPrenotazione = new.CodPrenotazione;
        else
                update Prenotazione
                set CostoTot = costo * n_notti
                where CodPrenotazione = new.CodPrenotazione;
        end if;

    return new;
    end;
$costo_camera$ language plpgsql;

create trigger costo_camera
after insert on Prenotazione
for each row
execute procedure costo_camera();
```

Trigger 4)

```
/*
Assicurarsi che ogni dipendente della reception effettui al massimo un turno al giorno.
*/

create function max_turni_reception()
returns trigger as $max_turni_reception$
    begin
        if (select count (*)
            from (select *
                from TurnoReception t
                where exists
                (
                    select *
                    from TurnoReception t2
                    where t.CF = t2.CF
                    and t.Giorno = t2.Giorno
                    and t.FasciaGiornaliera != t2.FasciaGiornaliera
                )) as c) > 0
        then
            raise exception 'Il turno inserito non è valido, un dipendente può
effettuare al più un turno al giorno.';
        end if;

    return new;
    end;
$max_turni_reception$ language plpgsql;

create trigger max_turni_reception
```

```
after insert or update on TurnoReception
for each row
execute procedure max_turni_reception();
```

Trigger 5)

```
/*
Aggiornare dato derivato CostoTot ogni volta che viene prenotato un posto auto.
*/

create function aggiorna_costo_auto()
returns trigger as $aggiorna_costo_auto$
	declare
		costo numeric(7,2);
		notti smallint;
	begin
		notti := (select DataFine
			from Prenotazione
			where CodPrenotazione = new.CodPrenotazione)
			-
			(select DataInizio
			from Prenotazione
			where CodPrenotazione = new.CodPrenotazione);

		costo:= (select CostoGiornaliero
			from PostoAuto
			where CodPrenotazione = new.CodPrenotazione
			and Posto = new.Posto);

		costo := costo * notti;

		update Prenotazione
		set CostoTot = CostoTot + costo
		where CodPrenotazione = new.CodPrenotazione;

	return new;
	end;
$aggiorna_costo_auto$ language plpgsql;

create trigger aggiorna_costo_auto
after insert or update	on PostoAuto
for each row
execute procedure aggiorna_costo_auto();
```

Trigger 6)

```
/*
Aggiornare il CostoTot ogni volta che viene prenotato un tavolo al Ristorante
*/

create function aggiorna_costo_ristorante()
returns trigger as $aggiorna_costo_ristorante$
	declare
		costo numeric(7,2);

	begin
		costo:= (select CostoPersona
			from Riservare
			where Data = new.Data
			and Ora = new.Ora
			and NumeroTavolo = new.NumeroTavolo)
			*
			(select Posti
			from Riservare
			where Data = new.Data
			and Ora = new.Ora
			and NumeroTavolo = new.NumeroTavolo);

		update Prenotazione
		set CostoTot = CostoTot + costo
		where CodPrenotazione = new.CodPrenotazione;

	return new;
	end;
$aggiorna_costo_ristorante$ language plpgsql;

create trigger aggiorna_costo_ristorante
after insert or update	on Riservare
for each row
execute procedure aggiorna_costo_ristorante();
```

Trigger 7)

```
/*
Aggiorna stipendio reception
*/
```

```
create function stipendio_reception()
returns trigger as $stipendio_reception$
	declare
		costo numeric (8,2);

	begin
		costo := (select RetribuzioneOraria
			from TurnoReception
			where Giorno = new.Giorno
			and FasciaGiornaliera = new.FasciaGiornaliera)
			* 6;

		update Receptionist
		set Stipendio = Stipendio + costo
		where CF = new.CF;
	return new;
	end;
$stipendio_reception$ language plpgsql;

create trigger stipendio_reception
after insert or update on TurnoReception
for each row
execute procedure stipendio_reception();
```

Trigger 8)

```
/*
Aggiorna stipendio personale pulizie
*/

create function stipendio_personale_pulizie()
returns trigger as $stipendio_personale_pulizie$
	declare
		costo numeric(8,2);

	begin
		costo := (select RetribuzioneOraria
			from TurnoPulizie
			where Giorno = new.Giorno
			and NumeroCamera = new.NumeroCamera);

		update PersonalePulizie
		set Stipendio = Stipendio + costo
```

```
                where CF = new.CF;

        return new;
        end;
$stipendio_personale_pulizie$ language plpgsql;

create trigger stipendio_personale_pulizie
after insert or update on TurnoPulizie
for each row
execute procedure stipendio_personale_pulizie();
```

Trigger 9)

```
/*
Controllare che un tavolo sia libero in quella fascia oraria.
*/

create function controllo_tavolo()
returns trigger as $controllo_tavolo$
        begin
                if (select count (*)
                        from (select *
                                from Riservare
                                where Data = new.Data
                                and NumeroTavolo = new.NumeroTavolo
                                and Ora != new.Ora
                                and ( select abs(extract (epoch from (Ora -
new.Ora))::real/3600)) <=3
                                ) as c
                        ) > 0
                then
                        raise exception 'Il tavolo selezionato sarà già occupato in questo
orario.';
                end if;

        return new;
        end;
$controllo_tavolo$ language plpgsql;

create trigger controllo_tavolo
after insert or update on Riservare
for each row
execute procedure controllo_tavolo();
```

Trigger 10)

```
/*
Controllare che il posto auto sia libro per tutta la durata del soggiorno
*/

create function controllo_posto_auto()
returns trigger as $controllo_posto_auto$
	declare
		data_inizio date;
		data_fine date;

	begin
		data_inizio := (select DataInizio
			from Prenotazione p
			where CodPrenotazione = new.CodPrenotazione);

		data_fine := (select DataFine
			from Prenotazione
			where CodPrenotazione = new.CodPrenotazione);

		if (select count(*)
			from (select *
				from PostoAuto pa, Prenotazione p
				where pa.CodPrenotazione = p.CodPrenotazione
				and pa.Posto = new.Posto
				and pa.CodPrenotazione != new.CodPrenotazione
				and (((p.DataInizio >= data_inizio)
					and (p.DataInizio <= data_fine))
					or
					((p.DataFine >= data_inizio)
					and (p.DataFine <= data_fine)))
				) as c) > 0
		then
			raise exception 'Il posto auto selezionato è già occupato in queste
date.';
		end if;

	return new;
	end;
$controllo_posto_auto$ language plpgsql;

create trigger controllo_posto_auto
after insert or update on PostoAuto
for each row
execute procedure controllo_posto_auto();
```

Trigger 11)

```
/*
Trigger che si assicura che ogni dipendente pulisca al massimo 6 camere al giorno.
*/

create function max_turni_pulizie()
returns trigger as $max_turni_pulizie$
	begin
		if (select count (*)
			from (select *
				from TurnoPulizie t
				where t.CF = new.CF
				and t.Giorno = new.Giorno
				) as c
			) > 6
		then
			raise exception 'Il turno inserito non è valido, un dipendente può
pulire al più 6 camere al giorno.';
		end if;
	return new;
	end;
$max_turni_pulizie$ language plpgsql;

create trigger max_turni_pulizie
after insert or update on TurnoPulizie
for each row
execute procedure max_turni_pulizie();
```

Query di Inserimento

Dipendente

```
insert into Dipendente values('TGRFTG76T56H225Q','Matteo','Pietri');
insert into Dipendente values('PNTRCA39J77B291C','Giovanni','Bottazzi');
insert into Dipendente values('PLNRDC88G53M883T','Marta','Galati');
insert into Dipendente values('TGBOKN65R59G261V','Riccardo','Marata');
insert into Dipendente values('OMRVET74B29V345T','Greta','Ascari');
insert into Dipendente values('ABCRTG25R88G441W','Mattia','Alberti');
insert into Dipendente values('ZZZERF45V52G998K','Ambra','Maccari');
insert into Dipendente values('QWEVGT28P71J665F','Sofia','Montanari');
insert into Dipendente values('RFVTGB85O19A257O','Alessandro','Teggi');
```

Sconto

```
insert into Sconto values('ABC23','Sconto fedeltà +5 prenotazioni',5);
insert into Sconto values('NNN77','Sconto fedeltà +3 visite in un anno',10);
```

Cliente

```
insert into Cliente values('MRCSMZ66Y67H223W','Marco','Simonazzi');
insert into Cliente values('POIYTR87G54N546F','Silvia','Lo Verde','ABC23');
insert into Cliente values('PLMFCR64T37H337Q','Filippo','Onesti','NNN77');
insert into Cliente values('BHURDC64T38B503V','Silvia','Drago');
insert into Cliente values('BUYCSW74M99B375L','Giada','Calzolari');
```

Dirigente

```
insert into Dirigente values('PNTRCA39J77B291C','Giovanni','Bottazzi',5500.00);
```

PersonalePulizie

```
insert into PersonalePulizie
values('PLNRDC88G53M883T','Marta','Galati',0,'PNTRCA39J77B291C');
insert into PersonalePulizie
values('OMRVET74B29V345T','Greta','Ascari',0,'PNTRCA39J77B291C');
insert into PersonalePulizie
values('ABCRTG25R88G441W','Mattia','Alberti',0,'PNTRCA39J77B291C');
```

Receptionist

```
insert into Receptionist
values('TGRFTG76T56H225Q','Matteo','Pietri',0,'PNTRCA39J77B291C');
insert into Receptionist
values('TGBOKN65R59G261V','Riccardo','Marata',0,'PNTRCA39J77B291C');
insert into Receptionist
values('ZZZERF45V52G998K','Ambra','Maccari',0,'PNTRCA39J77B291C');
insert into Receptionist
values('QWEVGT28P71J665F','Sofia','Montanari',0,'PNTRCA39J77B291C');
insert into Receptionist
values('RFVTGB85O19A257O','Alessandro','Teggi',0,'PNTRCA39J77B291C');
```

TurnoReception

```
insert into TurnoReception values('7-Sep-2018','Notte',10,'TGRFTG76T56H225Q');
insert into TurnoReception values('8-Sep-
2018','Mattino',9,'TGBOKN65R59G261V','RFVTGB85O19A257O');
insert into TurnoReception values('7-Sep-2018','Mattino',9,'ZZZERF45V52G998K');
insert into TurnoReception values('7-Sep-2018','Sera',9,'QWEVGT28P71J665F');
insert into TurnoReception values('8-Sep-2018','Notte',10,'TGRFTG76T56H225Q');
insert into TurnoReception values('7-Sep-2018','Pomeriggio',8,'RFVTGB85O19A257O');
```

```
/*
```

```
Query errata per testare il Trigger numero 4
        insert into TurnoReception values('07-Sep-
        2018','Pomeriggio',8,'TGRFTG76T56H225Q');
*/
```

Camera

```
insert into Camera values(101,3,37.5,0);
insert into Camera values(102,4,55,0);
insert into Camera values(103,2,30,0);
insert into Camera values(201,4,63.5,50);
insert into Camera values(202,2,45,50);
insert into Camera values(203,4,70,50);
insert into Camera values(301,5,120,100);
insert into Camera values(302,3,95.5,100);
insert into Camera values(303,4,250,150);
```

TurnoPulizie

```
insert into TurnoPulizie values('17-Sep-
2018',101,6.5,'PLNRDC88G53M883T','OMRVET74B29V345T');
insert into TurnoPulizie values('18-Sep-
2018',202,8,'OMRVET74B29V345T','ABCRTG25R88G441W');
insert into TurnoPulizie values('19-Sep-2018',303,9.5,'ABCRTG25R88G441W');
insert into TurnoPulizie values('17-Sep-2018',303,9.5,'PLNRDC88G53M883T');
insert into TurnoPulizie values('18-Sep-2018',303,9.5,'ABCRTG25R88G441W');
insert into TurnoPulizie values('19-Sep-2018',101,6.5,'PLNRDC88G53M883T');
insert into TurnoPulizie values('19-Sep-2018',102,6.5,'PLNRDC88G53M883T');
insert into TurnoPulizie values('19-Sep-2018',103,6.5,'PLNRDC88G53M883T');
insert into TurnoPulizie values('19-Sep-2018',201,8,'PLNRDC88G53M883T');
insert into TurnoPulizie values('19-Sep-2018',202,8,'PLNRDC88G53M883T');
insert into TurnoPulizie values('19-Sep-2018',203,8,'PLNRDC88G53M883T');
insert into TurnoPulizie values('19-Sep-2018',301,9.5,'ABCRTG25R88G441W');
insert into TurnoPulizie values('19-Sep-2018',302,9.5,'OMRVET74B29V345T');
```

PromozioneGiornaliera

```
insert into PromozioneGiornaliera values('24-Dec-2018',15,'Regalo di Natale');
insert into PromozioneGiornaliera values('8-Aug-2018',5,'Operazione Ferragosto');
```

Prenotazione

```
insert into Prenotazione values('AAAA0001','17-Sep-2018','18-Sep-2018',101,3,0,'8-Aug-
2018','MRCSMZ66Y67H223W');
insert into Prenotazione values('AAAA0002','20-Sep-2018','25-Sep-2018',303,3,0,'24-Dec-
2018','POIYTR87G54N546F');
insert into Prenotazione values('AAAA0003','01-Sep-2018','07-Sep-
2018',202,2,0,null,'PLMFCR64T37H337Q');
```

```
insert into Prenotazione values('AAAA0004','16-Sep-2018','21-Sep-
2018',201,3,0,null,'BHURDC64T38B503V');
insert into Prenotazione values('AAAA0005','4-Sep-2018','6-Sep-
2018',203,4,0,null,'BUYCSW74M99B375L');
insert into Prenotazione values('AAAA0006','7-Sep-2018','13-Sep-
2018',303,3,0,null,'MRCSMZ66Y67H223W');
insert into Prenotazione values('AAAA0007','27-Sep-2018','30-Sep-2018',102,4,0,'24-Dec-
2018','BHURDC64T38B503V');
insert into Prenotazione values('AAAA0008','14-Sep-2018','19-Sep-
2018',301,5,0,null,'MRCSMZ66Y67H223W');

/*
Query errata per testare il Trigger numero 1
        insert into Prenotazione values ('AAAA1111','19-Sep-2018','20-Sep-
        2018',201,3,0,null,'MRCSMZ66Y67H223W');

Query errata per testare il Trigger numero 2
        insert into Prenotazione values('AAAA0005','23-Nov-2018','28-Nov-
        2018',103,4,0,null,'BUYCSW74M99B375L');
*/
```

PostoAuto

```
insert into PostoAuto values(7,'AAAA0001',30);
insert into PostoAuto values(3,'AAAA0002',20);
insert into PostoAuto values(4,'AAAA0004',27);
insert into PostoAuto values(1,'AAAA0006',15);

/*
Query errata per testare il Trigger numero 10
        insert into PostoAuto values(7,'AAAA0004',20);
*/
```

Riservare

```
insert into Riservare values('18-Sep-2018','20:30',1,3,35,'AAAA0004');
insert into Riservare values('02-Sep-2018','13:00',4,2,25,'AAAA0003');
insert into Riservare values('20-Sep-2018','21:30',3,3,40,'AAAA0004');
insert into Riservare values('17-Sep-2018','13:00',6,3,25,'AAAA0001');
insert into Riservare values('22-Sep-2018','20:30',7,3,30,'AAAA0002');
insert into Riservare values('5-Sep-2018','13:30',2,4,20,'AAAA0005');
insert into Riservare values('28-Sep-2018','20:30',4,4,25,'AAAA0007');
insert into Riservare values('18-Sep-2018','21:00',5,5,40,'AAAA0008');

/*
Query errata per testare il Trigger numero 9
        insert into Riservare values('18-Sep-2018','19:00',1,5,20,'AAAA0001');
*/
```

Query di Interrogazione e Modifica

Le query sono state divise in tre gruppi di utilità, ovvero quelle utili per il cliente, quelle utili per il proprietario dell'hotel e infine quelle utili per i dipendenti.

- Query per il Cliente:

- Il cliente arriva alla cassa e vuole pagare, cerco la sua prenotazione e restituisco il costo totale

```
select CostoTot
from Prenotazione
where DataInizio = '20-Sep-2018'
and DataFine = '25-Sep-2018'
and NumeroCamera = 303
```

- Un cliente vuole sapere tutte le prenotazioni che ha effettuato nel nostro Hotel

```
select DataInizio,DataFine,CostoTot,NumeroCamera
from Prenotazione
where CF = 'MRCSMZ66Y67H223W'
order by DataInizio
```

- Query per il Proprietario:

- Il proprietario desidera visionare tutti gli stipendi dei receptionist

(Per questo si faccia riferimento alla vista "Stipendi" creata nel file hotel.sql)

- Il proprietario desidera sapere l'ammontare delle entrate di tutto il 2018

```
select sum(CostoTot)
from Prenotazione
where DataInizio between '01-Jan-2018' and '31-Dec-2018'
and DataFine between '01-Jan-2018' and '31-Dec-2018'
```

- Il proprietario vorrebbe sapere quali sono i clienti che spendono di più, in particolare quelli che hanno superato i 1000€ in almeno una prenotazione

```
select distinct Nome,Cognome
from Cliente c
where exists
(
	select *
	from Prenotazione p
	where c.CF = p.CF
	and CostoTot > 1000
)
```

- Il proprietario è curioso di sapere quante persone sono state in hotel durante il mese di settembre

```
select sum(NumPersone) Persone, count(*) Prenotazioni
from Prenotazione
where DataInizio between '01-Sep-2018' and '30-Sep-2018'
and DataFine between '01-Sep-2018' and '30-Sep-2018'
```

- Ogni mese, dopo aver pagato tutti gli stipendi, essi vengono azzerati per poter "ripartire" da zero

```
update Receptionist
set Stipendio = 0;
update PersonalePulizie
set Stipendio = 0
```

- Vengono mostrati tutti quei clienti (e la relativa prenotazione) i quali hanno usufruito sia del posto auto sia del ristorante.

```
select Nome,Cognome,CodPrenotazione
from Cliente, Prenotazione
where Cliente.CF = Prenotazione.CF
and CodPrenotazione in
(
	select CodPrenotazione
	from PostoAuto
)
and CodPrenotazione in
(
	select CodPrenotazione
	from Riservare
)
order by Cognome
```

- Query per i Dipendenti:

- Un receptionist vuole visualizzare tutti i turni che ha effettuato nel mese di settembre

```
select Giorno, FasciaGiornaliera
from TurnoReception
where CF = 'TGRFTG76T56H225Q'
and Giorno between '01-Sep-2018' and '30-Sep-2018'
```

- Un dipendente vuole sapere, per il mese di Settembre, quante camere ha pulito e in quali giorni

```
select Giorno, count(*) Camere_Pulite
from TurnoPulizie
where CF = 'PLNRDC88G53M883T'
and Giorno between '01-Sep-2018' and '30-Sep-2018'
group by Giorno
```