UNIVERSITA' DEGLI STUDI DI NAPOLI PARTHENOPE

DIPARTIMENTO DI SCIENZE E TECNOLOGIE

CORSO DI BASI DI DATI

AZIENDA DI AUTO-TRASPORTI

PROPONENTI:
Vetrano Alessio 0124/002326
D'anna Luca 0124/002324
Galluccio Luigi 0124/002293

DATA DI CONSEGNA:
25/06/2022

ANNO ACCADEMICO:
2021-2022

CATEGORIA:
Gestione Aziendale

Indice

1 PROGETTAZIONE

1.1 La base di dati in sintesi

Il nostro database nasce con lo scopo di gestire i dati di un'azienda di auto-trasporti. Quest'ultimo è popolato da un numero finito di **dipendenti**, di cui registriamo i fondamentali dati anagrafici, il loro contratto, lo stipendio, e le ferie che essi hanno. Il singolo dipendente può far parte solamente di una delle due specializzazioni lavorative previste dalla nostra azienda, che influenzeranno interamente la nostra base di dati.
La prima specializzazione è quella degli **impiegati**: essi lavorano in uffici numerati, e la nostra base di dati prevede che di essi vengano registrate le presenze, con rispettiva data, ora di entrata e ora di uscita, ed inoltre venga registrato eventualmente l'ufficio di appartenenza.
Maggiormente complessa è la seconda specializzazione: **gli autisti.** Questi si occupano dei trasporti e delle spedizioni in giro per la nazione. Ogni autista effettua una serie di viaggi dalla durata di uno o più giorni, ed ha il compito di caricare i prodotti presso un fornitore esterno, e portarli alle aziende che richiedono la spedizione di tale prodotto. Ogni viaggio con la rispettiva data, chilometri percorsi, soste effettuate, deve essere salvato nel database.
L'autista, con un singolo viaggio, ha la possibilità di smistare diversi lotti presso le aziende esterne interessate. Il viaggio può essere composto da più spedizioni, ed a ogni spedizione corrisponde una singola azienda esterna. Da precisare che ovviamente, alla singola azienda esterna è possibile associare più spedizioni in date diverse.
L'azienda prevede che ad ogni autista sia associato un veicolo personale, che può essere di 4 tipi diversi: furgone, motrice, bilico, autotreno. Ogni veicolo deve essere sottoposto a manutenzione registrata quando ne ha bisogno, e la base di dati si occupa anche di gestire tutte le officine in cui i veicoli vengono riparati o controllati.

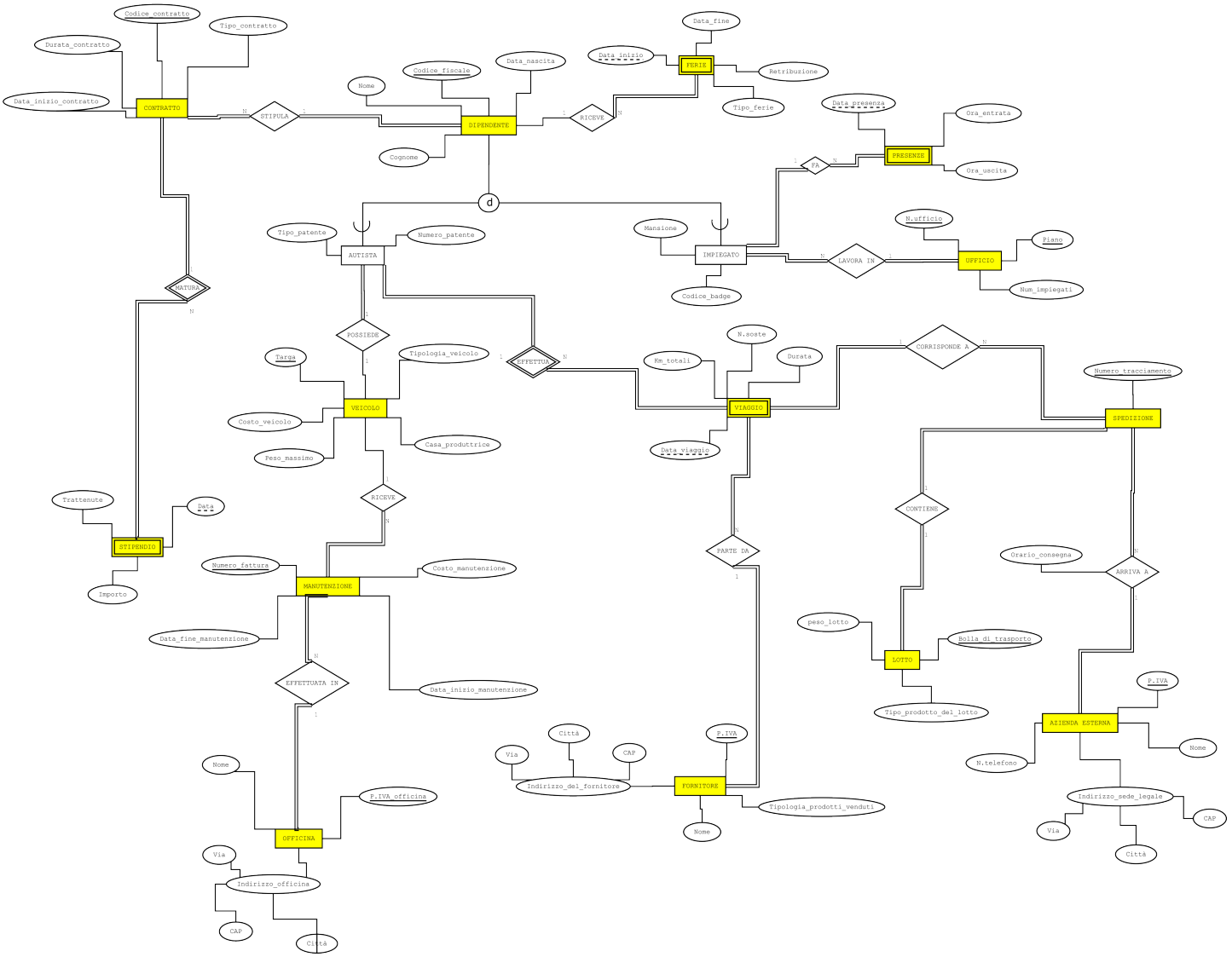
1.2 Glossario

Il glossario ha lo scopo di definire con la massima trasparenza, il gergo tecnico usato nella descrizione dei nostri requisiti. Anche se nel nostro caso la terminologia è pressochè semplice ed immediata.

TERMINE	DEFINIZIONE
DIPENDENTE	Singolo lavoratore aziendale divisibile in due specializzazioni
CONTRATTO	Definisce il tipo di contratto stipulato tra il singolo dipendente e l'azienda
STIPENDIO	Definisce lo stipendio che percepisce ogni dipendente
FERIE	Ferie richieste dal dipendente o che gli spettano
IMPIEGATO	Lavora solo in ufficio, oppure si occupa delle pulizie
PRESENZE	Presenze giornaliere di ogni impiegato in azienda
UFFICIO	Luogo di lavoro di un impiegato
AUTISTA	Si occupa delle spedizioni alle aziende, effettuando viaggi
VEICOLO	Singolo veicolo posseduto dall'autista per viaggiare
MANUTEZIONE	Riparazione a cui viene sottoposto il veicolo quando necessario
OFFICINA	Luogo dove i veicoli vengono sottoposti a manutenzione
VIAGGIO	Effettuato da ogni autista per spedire prodotti
FORNITORE	Luogo da cui parte l'autista dopo aver caricato i prodotti
AZIENDA EST	Luogo di arrivo dei prodotti
SPEDIZIONE	Atto del trasportare i prodotti da un luogo all'altro
LOTTO	Singolo insieme di prodotti da scaricare ad una singola azienda esterna

1.3 Diagramma EE/R

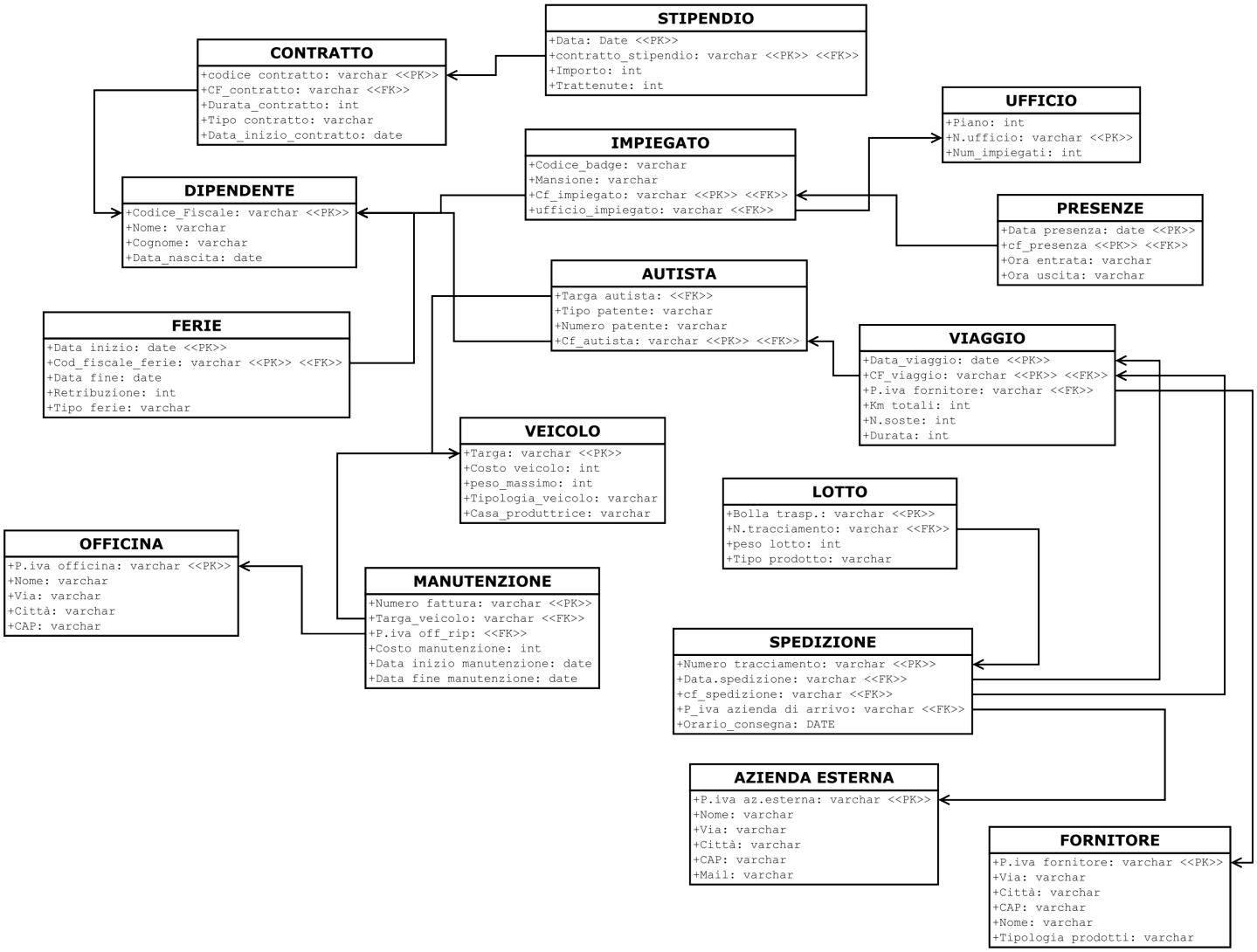
Il diagramma EE/R detto anche **modello concettuale**, costituisce una prima bozza del nostro data-base.
Nel diagramma ogni entità è collegata ad un' altra tramite una relazione che può godere di diversi tipi di molteplicità. Ogni entità contiene delle proprietà che la descrivono, dette **attributi**. Il diagramma contiene le eventuali specializzazioni delle entità, le totalità delle relazioni, le molteplicità tra le entità.



1.4 Diagramma Relazionale

Nel modello relazionale traduciamo le entità viste nel modello concettuale in tabelle, definendo i tipi degli attributi che abbiamo precedentemente dichiarato, e collegando le entità attraverso opportune chiavi esterne e rispettando, ovviamente, le regole delle molteplicità che ne seguono:

- Nella specializzazione (l'unica usata nel nostro data base),abbiamo un caso particolare in cui la chiave primaria delle entità figlie, coincide con la loro chiave esterna, che ovviamente deriva dalla chiave primaria dell'entità madre.
- Per le molteplicità 1-N, inseriamo la chiave primaria dell'entità (1) come chiave esterna sull' entità (N)
- Per le molteplicità 1-1, se abbiamo le totalità ad entrambi, l'inserimento della chiave esterna risulta indifferente, ma nel caso la totalità non sia di entrambi è consigliato inserire la chiave esterna dove c'è la totalita.

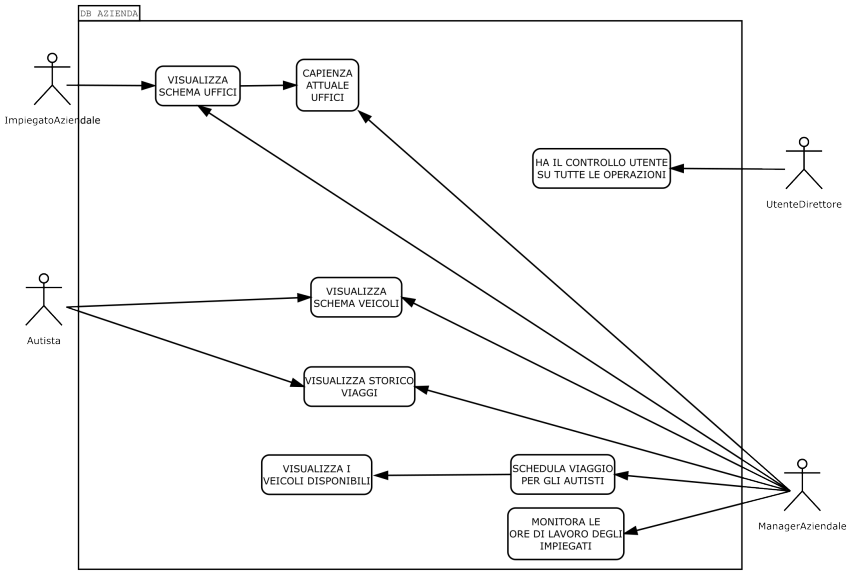


1.5 Utenti e loro categorie

Il controllo e la gestione del nostro data-base avvengono attraverso le operazioni di 4 utenti diversi.
Le operazioni di base previste dall' utente amministratore sono le classiche operazioni DML (Data Manipulation Language), ossia l'inserimento, modifica, o eliminazione di tuple, e le operazioni di interrogazione per la visualizzazione dei dati che ci servono in base alla richiesta.

UTENTE	TIPO	VOLUME	PERMESSI
utenteDirettore	Amministratore	1	ALL
ManagerAziendale	Comune	1	SELECT ON SchemaUffici SELECT ON SchemaVeicoliAssociati SELECT ON VeicoliDisponibili SELECT ON ImpiegatiInUfficioAttuale EXECUTE ON nuovoDirettore EXECUTE ON ScheduleViaggio EXECUTE ON contaOre TO ManagerAziendale
AutistaAziendale	Comune	1	SELECT ON SchemaVeicoliAssociati SELECT ON StoricoViaggi
ImpiegatoAziendale	Comune	1	SELECT ON SchemaUffici SELECT ON ImpiegatiInUfficioAttuale

L'utente **utenteDirettore** gode di tutti i privilegi aziendali, quindi la visualizzazione dei dati,e la modifica di essi.
ManagerAziendale ha i permessi per visualizzare lo schema uffici-impiegati, lo schema veicoli-autisti, lo storico dei viaggi operati dall'azienda e i veicoli da assegnare ad eventuali nuovi autisti. Ha il permesso di schedulare nuovi viaggi, trovando le giuste date libere ed autista disponibili.
AutistaAziendale può visualizzare lo storico viaggi e lo schema veicoli-autisti, mentre **ImpiegatoAziendale** ha i soli permessi di visualizzare lo schema uffici-impiegati e la capienza attuale.



Di seguito, è presentata la descrizione delle procedure su cui possono operare i nostri utenti

OPERAZIONE	**SchedulaViaggi**
SCOPO:	Trovare autista libero
RISULTATO:	Assegnare viaggio ad un autista
ERRORI:	Giorno non consentito
PRIMA:	Non ci sono viaggi per l'autista in quel giorno
DOPO:	L'autista ha un viaggio schedulato per quel giorno

OPERAZIONE	**ContaOre**
SCOPO:	Monitorare le ore di un impiegato
RISULTATO:	Avere il numero preciso di ore
ERRORI:	Impiegato non presente in azienda
PRIMA:	Nessun informazione sulle ore di presenza
DOPO:	Ore esatte di presenza

1.6 Volumi

Di seguito è rappresentata la tavola dei volumi, che riporta il numero di tuple presenti in ciascuna tabella una volta che il DB è stato completato.

TABELLA	TIPO	VOLUME	INCREMENTO	PERIODO
DIPENDENTE	E	30	10	anno
CONTRATTO	E	30	10	anno
STIPENDIO	E	120	20	bimestre
FERIE	ED	36	15	semestre
IMPIEGATO	E	15	5	anno
PRESENZE	ED	75	50	settimana
UFFICIO	E	15	0	anno
AUTISTA	E	15	10	anno
VEICOLO	E	20	5	anno
MANUTEZIONE	E	15	10	mese
OFFICINA	E	15	0	anno
VIAGGIO	ED	30	45	settimana
FORNITORE	E	15	5	mese
AZIENDA EST	E	15	20	mese
SPEDIZIONE	E	45	135	mese
LOTTO	E	45	135	mese

- **TIPO**: E-ED corrispondono rispettivamente ad ENTITA' ed ENTITA' DEBOLE.
- **INCREMENTO** rappresenta l'incremento atteso del numero di tuple in un periodo di tempo prefissato.
- **PERIODO**: rappresenta il periodo entro quale si può registrare l'incremento del numero delle tuple.

1.7 Vincoli di Integrità

Sono detti **statici**, i vincoli che limitano i valori assumibili dagli attributi delle nostre entità indipendentemente dal tempo che passa. I vincoli **dinamici**, invece, sono vincoli che possono variare nel tempo, oltre che presentarsi, a volte, anche come regole di business. Di seguito si riportano alcuni vincoli del nostro Data-base:

1.7.1 Vincoli statici

- Sono ammessi 4 tipi di veicoli: furgone, motrice, bilico, autotreno.
- Sono ammesse 5 mansioni aziendali: Segretario, Analista, Legale, Manager, Direttore.
- Un veicolo aziendale ha una capacità massima di trasporto,che non può essere violata.
- Se l'autista non è provvisto di apposita patente, non può guidare un determinato tipo di veicolo.
- Il numero di dipendenti associati ad un ufficio, non può essere superiore alla capacità massima dell'ufficio stesso.

- All'assunzione, un dipendente non può aver superato i 60 anni, ed ovviamente deve essere maggiorenne.
- In azienda non può esserci più di un direttore.
- Se un'officina ha già due veicoli in riparazione, non può essere ammesso il terzo.
- Se un viaggio, ha raggiunto un carico approssimativamente massimo, non si potrà aggiungere un'altro lotto nel veicolo.
- Per ogni mansione, esiste un valore minimo di stipendio.
- Non posso mandare in ferie un dipendente se ci sono già due persone in ferie, nello stesso ufficio.
- E' impossibile aggiungere una presenza a quell'impiegato, se esso è in ferie.
- Non posso eliminare un ufficio se ci sono impiegati associati ad esso.
- Il numero massimo di spedizioni per ogni viaggio è fissato a 3.

1.7.2 Vincoli dinamici

- E' possibile schedulare un viaggio appena un autista è disponibile per quella data, e se quella data non è disponibile, viene cercata automaticamente una data libera.
- Un segretario può essere promosso a manager solo se ha un contratto di tipo determinato o indeterminato.
- Non è possibile licenziare un direttore senza averne eletto un nuovo.
- Il dipendente non può ricevere l'aumento se non ha ancora percepito lo stipendio.
  item Il rinnovo del contratto avviene se quest'ultimo non è a tempo indeterminato.

1.8 Verifica di Normalità

Con la verifica di normalità, vogliamo combattere la ridondanza limitando al minimo le dipendenze funzionali anomale. Perciò sottoponiamo il nostro schema ad una serie di test per "certificare" che soddisfi una data forma normale.

1.8.1 Prima forma normale

Il DB rispetta la prima forma normale, data la presenza di tutti campi unici. La targa del veicolo, che può non sembrare atomica, poichè divisibile in vari campi, nel nostro caso assume valore atomico poichè ha il solo scopo di identificare il singolo veicolo a essa associato, appartenente alla nostra azienda. Il campo data, cosi come nella maggior parte dei linguaggi di programmazione, è convenzionalmente considerato atomico anche in ORACLE DBMS.

1.8.2 Seconda forma normale

Una buona base di dati è in 2NF quando tutti gli attributi non primi dipendono funzionalmente dall' intera chiave composta, quindi non dipendono funzionalmente solo da una parte della chiave. Nel nostro caso, abbiamo 5 entità con chiave composta: ferie,stipendio,presenza,viaggio e spedizione. In ognuna di esse, è rispettata la 2NF poichè ogni attributo non primo di esse, è dipendente funzionalmente solo se consideriamo l'intera chiave composta. Possiamo dire che il DB si trova in seconda forma normale.

1.8.3 Terza forma normale

La terza forma normale gestisce le dipendenze funzionali anomale, ossia le dipendenze funzionali da attributi non primi (che non siano chiavi). In parole semplici, tale forma è atta ad evitare che un generico attributo non-chiave dipenda da altri attributi non-chiave generici. Nel nostro DB, abbiamo controllato che non ci fossero dipendenze funzionali anomale, tranne un caso in cui nella tabella veicolo, `peso massimo` dipende dalla tipologia del veicolo, ma in realtà questo è solo un semplice constraint check inserito da noi per poi effettuare tutti i possibili controlli al fine di evitare veicoli che trasportino peso maggiore di quello consentito. Rispettando tale forma, possiamo anche dire che lo schema è in Boys-Codd Normal Form (BCNF). Ricordiamo che tale forma impone che ogni qual volta che sussiste in R una dipendenza funzionale non banale X→A, X è una superchiave di R.

2 IMPLEMENTAZIONE

Una volta terminata tutta la fase di progettazione, con tutte le idee unite in un singolo puzzle, il tutto deve essere convertito in codice eseguibile scritto correttamente. Utilizziamo il linguaggio PL/SQL.

2.1 Creazione Utenti

Il primo passo che effettuiamo è creare gli utenti che potranno accedere alla base di dati, in questo caso utente direttore,impiegato,manager,autista. Dopo aver creato gli utenti, diamo loro tutti i privilegi usando gli opportuni comandi che fornisce il linguaggio.

```
CREATE USER utenteDirettore       IDENTIFIED BY adminDirector1234;
CREATE USER ManagerAziendale      IDENTIFIED BY AziendaManagmentDB;
CREATE USER AutistaAziendale      IDENTIFIED BY AziendaAutista345;
CREATE USER ImpiegatoAziendale    IDENTIFIED BY AziendaImpiegato895;
```

2.2 Data Definition Language

Il DDL riflette il modello relazionale: creiamo le tabelle dello schema usando il comando `CREATE TABLE`.
Nelle tabelle vanno inclusi la maggior parte dei vincoli statici della base di dati, attraverso il comando `CONSTRAINT nomevincolo CHECK()`, e vanno specificati tutti i tipi dei dati che andiamo ad utilizzare per il nostro data-base.

Prima di iniziare a fare alcuni esempi di creazione tabelle, precisiamo che quando usiamo il tipo DATE stiamo includendo anche l'ora, quindi nel caso volessimo dati del tipo (ora,minuti,secondi) questi campi sono inclusi in tale tipo di dato.

2.2.1 Alcuni esempi di tabelle

Il dipendente rappresenta il perno principale di tutta la nostra azienda, ha una chiave primaria rappresentata dal codice fiscale, che è un varchar di 16 caratteri. Abbiamo ritenuto coerente oltre che obbligatorio ai fini della correttezza del nostro data-base, aggiungere un constraint per il formato del codice fiscale in modo che, durante l'aggiunta di quest'ultimo, non venga inserito un carattere fuori posto che poi possa condizionare le diverse interrogazioni sulla nostra base di dati. Inoltre abbiamo inserito un vincolo statico per cui i dipendenti non possano avere una data di assunzione che vada prima del 1990 (anno di fondazione della nostra azienda)

```
CREATE TABLE dipendente (
cf VARCHAR2(16) NOT NULL PRIMARY KEY,
nome VARCHAR2(20) NOT NULL,
cognome VARCHAR2(20) NOT NULL,
data_nascita DATE NOT NULL,
data_assunzione DATE NOT NULL,

CONSTRAINT check_cf CHECK(
REGEXP_LIKE(cf,'[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]')),

CONSTRAINT data_assunzione_check CHECK(

TO_CHAR(data_assunzione,'YYYY-MM-DD') between '1990-01-01' AND '2022-01-01')
);
```

L'impiegato è una delle due specializzazioni del dipendente nella nostra azienda. Dal codice, qui in basso, si notano alcuni vincoli statici che abbiamo deciso di fissare, ossia la possibilità di avere soltanto 5 tipi di mansioni. Come possiamo notare, uno dei dati in questa tabella, ossia `ufficio impiegato`, rappresenta la chiave esterna per l'ufficio, Lo facciamo grazie all' uso dei comandi `FOREIGN KEY` e `REFERENCES` che collegano la chiave esterna di questa tabella con la chiave primaria della tabella UFFICIO.

```
CREATE TABLE impiegato (
cf_impiegato VARCHAR2(16) PRIMARY KEY,
ufficio_impiegato VARCHAR2(2) NOT NULL,
codice_badge VARCHAR2(5) NOT NULL UNIQUE,
mansione VARCHAR2(30) NOT NULL,
FOREIGN KEY(ufficio_impiegato) REFERENCES ufficio(num_ufficio)
ON DELETE CASCADE,
FOREIGN KEY(cf_impiegato) REFERENCES dipendente(cf)
ON DELETE CASCADE,

CONSTRAINT cf_ammesso CHECK(
REGEXP_LIKE(cf_impiegato,'[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]')),

CONSTRAINT mansione_ammessa CHECK(
LOWER(mansione) IN
('segretario','analista','legale','manager','direttore'))
);
```

Lo stipendio viene percepito una volta al mese da ogni dipendente. In questa tabella abbiamo deciso di utilizzare doppia chiave composta da `data stipendio` e `contratto stipendio`,dato che nella progettazione della base di dati, stipendio risulta essere un'entità debole.
Abbiamo ritenuto coerente inserire un vincolo per il formato del codice del contratto, in modo che l'inserimento risulti sempre corretto.

```
    CREATE TABLE stipendio (
data_stipendio DATE,
contratto_stipendio VARCHAR2(10) NOT NULL,
importo NUMBER(4) NOT NULL,
trattenute NUMBER(3) NOT NULL,
FOREIGN KEY(contratto_stipendio) REFERENCES contratto(codice_contratto)
ON DELETE CASCADE,
CONSTRAINT pk_stipendio PRIMARY KEY(data_stipendio,contratto_stipendio),

CONSTRAINT cod_contratto_stipendio CHECK(
REGEXP_LIKE(contratto_stipendio,'[A-Z]{3}[0-9]{7}'))
);
```

L'autista è la seconda specializzazione di dipendente. Abbiamo adottato il vincolo che i nostri autisti debbano avere almeno uno dei quattro tipi di patente come requisito minimo per lavorare in azienda. Anche qui inseriamo vincoli di formato sul codice fiscale e targa, oltre che al numero di patente.

```
CREATE TABLE autista (
cf_autista VARCHAR2(16) PRIMARY KEY,
tipo_patente VARCHAR2(3) NOT NULL,
num_patente VARCHAR2(10) NOT NULL UNIQUE,
targa_autista VARCHAR2(7) NOT NULL UNIQUE,
FOREIGN KEY (targa_autista) REFERENCES veicolo(targa)
ON DELETE CASCADE,
FOREIGN KEY(cf_autista) REFERENCES dipendente(cf)
ON DELETE CASCADE,

CONSTRAINT cf_autista_mask CHECK(
REGEXP_LIKE(cf_autista,'[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]')),

CONSTRAINT targa_autista_mask CHECK(
REGEXP_LIKE(targa_autista,'[A-Z]{2}[0-9]{3}[A-Z]{2}')),

CONSTRAINT tipo_patente_mask CHECK (
UPPER(tipo_patente) IN
('B1','C1','C','CE')),

CONSTRAINT num_patente_mask CHECK (
REGEXP_LIKE(num_patente,'[A-Z]{2}[0-9]{7}[A-Z]'))
);
```

Il veicolo effettua i viaggi che permettono di spedire materiale e prodotti alle aziende esterne, con cui la nostra collabora. La chiave primaria è costituita dalla targa. In questa tabella abbiamo inserito diversi vincoli: un'opportuno inserimento della targa del veicolo fa si che non ci siano errori nel nostro data-base. Il secondo vincolo consiste nel poter inserire solo quattro tipi di veicoli per effettuare tratte più o meno lunghe. Il terzo vincolo consiste nel poter inserire solo quattro case produttrici di veicolo nella nostra base, e l'ultimo ma più importante vincolo dà coerenza alla relazione tra capienza del veicolo e peso della spedizione che esso trasporta. In questo modo non potremo mai avere registrato un veicolo che trasporta prodotti per un peso complessivo maggiore alla sua capienza.

```
CREATE TABLE veicolo (
targa VARCHAR2(7) NOT NULL PRIMARY KEY,
costo_veicolo INT NOT NULL,
peso_massimo NUMBER(3) NOT NULL,
tipo_veicolo VARCHAR2(20) NOT NULL,
casa_produttrice VARCHAR2(10) NOT NULL,
CONSTRAINT targa_mask CHECK(
REGEXP_LIKE(targa,'[A-Z]{2}[0-9]{3}[A-Z]{2}')),

CONSTRAINT veicolo_ammesso CHECK(
LOWER(tipo_veicolo) IN
('furgone','motrice','bilico','autotreno')),

CONSTRAINT casa_prod CHECK(
INITCAP(casa_produttrice) IN
('Mercedes','Iveco','Scania','Daf')),

CONSTRAINT peso_veicolo CHECK(
(LOWER(tipo_veicolo) = 'furgone' AND peso_massimo = 10)
OR
(LOWER(tipo_veicolo) = 'motrice' AND peso_massimo = 25)
OR
(LOWER(tipo_veicolo) = 'bilico' AND peso_massimo = 240)
OR
(LOWER(tipo_veicolo) = 'autotreno' AND peso_massimo = 260))
);
```

Il contratto definisce le formalità tra dipendente ed azienda,definendo ovviamente il tipo di contratto e il periodo di collaborazione tra dipendente ed azienda. Con `CONSTRAINT durataContratto CHECK` ci assicuriamo che ogni contratto abbia la durata assicurata, tranne nel caso in cui sia a tempo indeterminato, dove la durata può assumere valore nullo.

```
CREATE TABLE contratto (
codice_contratto VARCHAR2(10) NOT NULL PRIMARY KEY,
cf_contratto VARCHAR2(16) NOT NULL,
durata_contratto NUMBER,
tipo_contratto VARCHAR2(25) NOT NULL,
data_inizio_contratto DATE NOT NULL,
FOREIGN KEY(cf_contratto) REFERENCES dipendente(cf)
ON DELETE CASCADE,

CONSTRAINT data_inizio_contratto_check CHECK(

TO_CHAR(data_inizio_contratto,'YYYY-MM-DD') > '1990-01-01' ),

CONSTRAINT durata_contatto CHECK(
(LOWER(tipo_contratto) = 'indeterminato' and durata_contratto is null)
OR
(LOWER(tipo_contratto) = 'determinato' and durata_contratto is not null)
OR
(LOWER(tipo_contratto) = 'part-time' and durata_contratto is not null)
OR
(LOWER(tipo_contratto) = 'apprendistato' and durata_contratto is not null)),

CONSTRAINT cod_contratto CHECK(
REGEXP_LIKE(codice_contratto,'[A-Z]{3}[0-9]{7}')),

CONSTRAINT cf_contratto_mask CHECK(
REGEXP_LIKE(cf_contratto,'[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]'))

);
```

La presenza è fondamentale per memorizzare le ore di entrata ed uscita di ogni impiegato dall'azienda, ovviamente verifichiamo tramite dei check, che gli orari vengano correttamente memorizzati nel DB.

```
CREATE TABLE presenza (
Data_presenza DATE,
cf_presenza VARCHAR2(16),
ora_entrata DATE,
ora_uscita DATE,
FOREIGN KEY(cf_presenza) REFERENCES impiegato(cf_impiegato)
ON DELETE CASCADE,
CONSTRAINT pk_presenza PRIMARY KEY(data_presenza,cf_presenza),

CONSTRAINT cf_presenza_mask CHECK(
REGEXP_LIKE(cf_presenza,'[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]')),

CONSTRAINT data_presenza_check CHECK(


TO_CHAR(data_presenza,'YYYY-MM-DD') > '1990-01-01') --CONTROLLA SE FUNZIONA

);
```

2.3 Data Manipulation Language

Il popolamento delle nostre tabelle avviene attraverso operazioni dirette messe a disposizione dal linguaggio SQL,ossia i comandi `INSERT` o `UPDATE`, l'abilitazione all'inserire dati all'interno delle tabelle è esclusivamente garantita al direttore d'azienda.

Di seguito,un esempio di inserimento per ogni tabella creata:

```
INSERT INTO dipendente (cf,nome,cognome,data_nascita)
VALUES ('CLMGLC85L02I926V','Gianluca','Colombarini',DATE '1985-07-02');

INSERT INTO contratto
VALUES ('QRT1859037','MRTPLN65L69G577R',0.6,'Part-time',DATE'2004-09-01');

INSERT INTO ufficio(num_ufficio, piano_ufficio, num_impiegati) values ('01',1,2);

INSERT INTO impiegato(cf_impiegato,ufficio_impiegato,codice_badge,mansione)
VALUES ('MRTPLN65L69G577R','10','01003','Analista');

INSERT INTO stipendio (data_stipendio,contratto_stipendio,importo,trattenute)
VALUES (date'2022-01-27','QRT1859038',2057,143);

INSERT INTO veicolo (targa, costo_veicolo, peso_massimo, tipo_veicolo, casa_produttrice)
VALUES ('BA767WF','32000',100,'furgone','Mercedes');

INSERT INTO officina (p_iva_officina, nome, via, citta, cap)
VALUES ('46083770191', 'SUDueRuote', 'Via Gian Battista Vico','Napoli','80137');

INSERT INTO fornitore VALUES
('74015300846','Biscottificio Fazzi', 'Via Della Meccanica, 10','Verona','37139','biscotti');

INSERT INTO ferie(data_inizio_ferie,Cod_fiscale_ferie,data_fine,Retribuzione,tipo_ferie)
VALUES (date'2022-04-08','MRTPLN65L69G577R',date'2022-04-08',NULL,'Impegni Familiari');

INSERT INTO autista (cf_autista,tipo_patente,num_patente,targa_autista)
VALUES ('CLMGLC85L02I926V','C1','NA2113212M','FS717BG');

INSERT INTO presenza
VALUES (to_date('2022-04-04','yyyy-mm-dd'),'MRTPLN65L69G577R',to_date('2022-04-04
08:00','yyyy-mm-dd hh24:mi'),to_date('2022-04-04 17:00','yyyy-mm-dd hh24:mi'));

INSERT INTO manutenzione
VALUES  ('122030','CZ371YF','46083770191',1600,DATE '2022-01-10',DATE'2022-01-05');

INSERT INTO viaggio VALUES(to_date('2022-04-04 8:00','yyyy-mm-dd
hh24:mi'),'CLMGLC85L02I926V','74015300846','1000','6','41');

INSERT INTO azienda_esterna VALUES ('28392680808','Euronics', 'Via San Secondo,
98Bis','Torino','10128','euro@euronicsitalia.it');

INSERT INTO spedizione VALUES
('1945690239',to_date('2022-04-04 08:00','yyyy-mm-dd hh24:mi'),'CLMGLC85L02I926V',to_date
('2022-04-06 09:00','yyyy-mm-dd hh24:mi'),'56363534242');

INSERT INTO lotto(bolla_trasporto,tracciamento_lotto,peso_lotto,tipo_prodotto)
VALUES ('ALBN837589','1945690239','20','mobili');
```

2.4 Trigger

I trigger DML sono utili per il controllo dei vincoli aziendali in fase di immissione o aggiornamento dei dati, oltre che usati anche per implementare le regole di business. Nel nostro caso i trigger sono usati per popolare tutte le tabelle coinvolte nella base di dati,assicurando la non violazione dei vincoli.

2.4.1 checkEta

Questo trigger controlla semplicemente che all'inserimento di un dipendente, quest'ultimo risulti avere un'età compresa tra i 18 e 60 anni: requisito fondamentale della nostra azienda. Inseriamo l'eta del dipendente che si vuole inserire, in una variabile ed effettuiamo un controllo su di essa. Nel caso l'età non rispettasse i nostri requisiti, il trigger va in eccezione.

```
CREATE OR REPLACE TRIGGER check_eta
before insert on dipendente
for each row
DECLARE
check_eta EXCEPTION;
BEGIN
if (floor((sysdate-:new.data_nascita)/365) < 18 OR (floor((sysdate-:new.data_nascita)/365)) > 60 ) then
raise check_eta;
end if;
EXCEPTION
when check_eta then
raise_application_error(-20001,'Il candidato dipendente non rientra nei paramentri di assunzione');
end;
```

2.4.2 checkMansione

Il trigger `checkMansione` effettua due controlli in uno, all'inserimento sulla stessa tabella impiegato. Verifica non solo che l'impiegato non sia direttore, nel caso già ce ne fosse uno, ma verifica anche che l'ufficio che gli stiamo assegnando non abbia raggiunto la sua massima capacità. Controlliamo che nell'azienda ci sia già un direttore con una query che ci restituisce il numero di impiegati con mansione di direttore: se ci troviamo in una situazione regolare, allora la query ci restituirà contatore = 1, e il trigger andrà in errore. Stessa cosa per il controllo sull'ufficio. Abbiamo ragionato sempre usando un contatore che attraverso una query, conta il numero di impiegati per quell'ufficio, e va in errore nel caso `((contatoreUff + 1) > numero ufficio)`, dove `numero ufficio`→numero di impiegati nell'ufficio, al momento dell'inserimento.

```
CREATE OR REPLACE TRIGGER checkMansione
before insert on impiegato
for each row
DECLARE
overNum EXCEPTION;
overNumUfficio EXCEPTION;
contatore int;
contatoreUff NUMBER;
num_max INT;
numero_ufficio varchar(30);
BEGIN

select count(*) into contatore from impiegato
where mansione = 'Direttore';

if :new.mansione = 'Direttore' and contatore = 1 then
raise overNum;
end if;

select count(*) into contatoreUff
from impiegato im join ufficio uff on im.ufficio_impiegato = uff.num_ufficio
where uff.num_ufficio = :new.ufficio_impiegato;


select num_impiegati into numero_ufficio
from ufficio
where :new.ufficio_impiegato = num_ufficio;
```

```
if ((contatoreUff + 1) > numero_ufficio) then
raise overNumUfficio;
end if;

EXCEPTION
when overNum then
raise_application_error(-20001,'ESISTE GIA UN DIRETTORE IN AZIENDA');
when overNumUfficio then
raise_application_error(-20001,'Errore l''ufficio pieno. L''impiegato non può essere assegnato.');
end;
```

2.4.3 checknumVeicoli

Il trigger controlla se nello stesso momento, un' officina ha in riparazione dei veicoli. In tal caso, quando proveremo ad aggiungere il terzo, il trigger ci darà errore poichè abbiamo stabilito che un officina non può avere più di due veicoli in riparazione. Anche in questo caso usiamo un semplice contatore, che attraverso una query controlla il numero di veicoli in manutenzione.

```
CREATE OR REPLACE TRIGGER checknumVeicoli
before insert on manutenzione
for each row
DECLARE
overNum EXCEPTION;
contatore NUMBER;
BEGIN
select count(*) into contatore from manutenzione where p_iva_meccanico = :new.p_iva_meccanico AND
(:new.data_inizio_man between data_inizio_man and data_fine_man);
if (contatore > 2) then
  raise overNum;
end if;
EXCEPTION
when overNum then
raise_application_error(-20001,'OFFICINA TROPPO PIENA');
END;
```

2.4.4 checkPeso

Questo trigger è fondamentale ed articolato, al fine di evitare che venga aggiunto ad un viaggio, un lotto che poi faccia sforare il tetto massimo di peso, imposto ai nostri veicoli. Con la prima query, calcoliamo il peso massimo del veicolo con cui effettuiamo il viaggio, e con la seconda query calcoliamo il peso totale dei lotti già associati al viaggio. Quando tentiamo di inserire un lotto che fa sforare il peso massimo, il trigger ci impedirà di farlo segnalandoci l'errore.

```
CREATE OR REPLACE TRIGGER checkPeso
before insert or update on lotto
for each row
DECLARE
overPeso EXCEPTION;
peso INT;
somma_lotti INT;
BEGIN
select peso_massimo into peso from viaggio v
join fornitore forn on v.p_iva_forn = forn.p_iva_fornitore
join spedizione s on v.cf_viaggio  = s.cf_spedizione and v.data_viaggio = s.data_spedizione
join azienda_esterna az on s.p_iva_aziendaArrivo = az.p_iva_azienda_esterna
join autista aux on v.cf_viaggio = aux.cf_autista
join veicolo v on v.targa = aux.targa_autista
where s.num_tracciamento = :new.tracciamento_lotto;

select sum(peso_lotto) into somma_lotti from lotto lt join spedizione sp
on lt.tracciamento_lotto = sp.num_tracciamento
where cf_spedizione = (select cf_spedizione from spedizione where num_tracciamento = :new.tracciamento_lotto)
and data_spedizione = (select data_spedizione from spedizione where num_tracciamento = :new.tracciamento_lotto);

if (peso <= :new.peso_lotto + somma_lotti) then
```

```
raise overPeso;
end if;
EXCEPTION
when overPeso then
raise_application_error(-20001,'IL VEICOLO NON PUO CONTENERE QUESTO LOTTO');
END;
```

2.4.5 checkStipendio

Controlla che ogni stipendio inserito per una determinata mansione, non vada al di sotto di un certo valore imposto da standard aziendali.

```
CREATE OR REPLACE TRIGGER checkStipendio
before insert on stipendio
for each row
DECLARE
mansioneCheck VARCHAR2(30);
stipendio_basso EXCEPTION;
contatore NUMBER;
contatoreIm NUMBER;
BEGIN
select count(*),mansione into contatore, mansioneCheck from impiegato im join dipendente dip
on im.cf_impiegato = dip.cf
    join contratto contr on dip.cf = contr.cf_contratto
    where contr.codice_contratto = :new.contratto_stipendio
    group by cf_impiegato, mansione;

if (contatore > 0) then
     if (:new.importo < 3000 and mansioneCheck = 'Direttore') then
        raise stipendio_basso;
    elsif (:new.importo < 1200 and mansioneCheck = 'Segretario') then
        raise stipendio_basso;
    elsif (:new.importo < 1300 and mansioneCheck = 'Manager') then
        raise stipendio_basso;
    elsif (:new.importo < 2300 and mansioneCheck = 'Legale' ) then
        raise stipendio_basso;
    elsif (:new.importo < 1950 and mansioneCheck = 'Analista') then
        raise stipendio_basso;
    end if;
end if;

EXCEPTION
when NO_DATA_FOUND then
    if :new.importo < 1200 then
    raise stipendio_basso;
    end if;
when stipendio_basso then
raise_application_error(-20001,'STIPENDIO TROPPO BASSO PER QUESTA DETERMINATA MANSIONE');
end;
```

2.4.6 checkNumDip

Il trigger effettua un controllo in fase di inserimento di un dipendente, verificando attraverso un contatore che con la presente assunzione, non si sfori il tetto massimo di dipendenti all'interno dell'azienda.

```
CREATE OR REPLACE TRIGGER checkNumDip
before insert or update on dipendente
for each row
DECLARE
MaxNumDip EXCEPTION;
cont NUMBER;
BEGIN
select count(*) into cont from dipendente;
```

```
if (cont=40) then
raise MaxNumDip;
end if;
EXCEPTION
when MaxNumDip then
raise_application_error(-20001,'RAGGIUNTO MASSIMO NUMERO DI DIPENDENTI');
END;
```

2.4.7 checkPatente

Verifica, con una serie di controlli, che quando vogliamo associare un nuovo autista ad uno dei veicoli ovviamente disponibili in azienda, l'autista stesso possa guidare il veicolo e quindi sia dotato di opportuna patente per farlo regolarmente.

```
CREATE OR REPLACE TRIGGER checkPatente
before insert on autista
FOR EACH ROW
DECLARE
checkVeicolo veicolo.tipo_veicolo%type;
Error1 EXCEPTION;

BEGIN
select tipo_veicolo into checkVeicolo from veicolo where targa = :new.targa_autista;

if checkVeicolo = 'motrice' AND :new.tipo_patente = 'B1' then
raise Error1;
elsif checkVeicolo = 'bilico' AND :new.tipo_patente = 'B1' then
raise Error1;
elsif checkVeicolo = 'bilico' AND :new.tipo_patente = 'C1' then
raise Error1;
elsif checkVeicolo = 'autotreno' AND :new.tipo_patente != 'CE' then
raise Error1;
end if;

EXCEPTION
when Error1 then
raise_application_error(-20001,'QUESTO VEICOLO NON PUO ESSERE GUIDATO CON TALE PATENTE');
END;
```

2.4.8 checkFerie

Con questo trigger, diventa sicura la generazione di un errore nel momento in cui vogliamo mandare in ferie un impiegato, quando nell'ufficio di appartenenza già due impiegati sono in ferie.

```
CREATE OR REPLACE TRIGGER check_ferie
BEFORE INSERT ON FERIE
FOR EACH ROW
DECLARE
num_ufficio VARCHAR2(2);
contatore_impiegati NUMBER;
underImpiegati EXCEPTION;
BEGIN

select ufficio_impiegato into num_ufficio from impiegato im
join dipendente dip on dip.cf = im.cf_impiegato
where cf_impiegato = :new.cod_fiscale_ferie;

select count(*) into contatore_impiegati from impiegato
join dipendente dip on cf_impiegato = dip.cf
where UFFICIO_IMPIEGATO = num_ufficio
and cf_impiegato != :new.cod_fiscale_ferie;


if (contatore_impiegati > 2) then
```

```
  raise underImpiegati;
end if;


EXCEPTION

when underImpiegati then
raise_application_error(-20001,'Non è possibile assegnare le ferie');
END;
```

2.4.9 checkPres

Il trigger impossibilita l'inserimento di una presenza nel caso ci fosse una ferie associata a quel dipendente, in quell'esatto giorno.

```
CREATE OR REPLACE TRIGGER checkPres
before insert on presenza
for each row
DECLARE
check_ferie EXCEPTION;
contatore NUMBER;
BEGIN
select count(*) into contatore from ferie
where cod_fiscale_ferie = (select cf_presenza from impiegato im
join presenza pr on im.cf_impiegato = pr.cf_presenza
where cf_presenza = :new.cf_presenza
and :new.data_presenza between DATA_INIZIO_FERIE and DATA_FINE
group by cf_presenza);

if contatore > 0 then
    raise check_ferie;
end if;

EXCEPTION
when check_ferie then
raise_application_error(-20001,'Non è possibile inserire una presenza poichè il dipednente è in ferie');
end;
```

2.4.10 checkElimUff

Rende impossibile l'eliminazione di un ufficio, nel caso in cui fossero dipendenti che lavorano in esso. Rispetto a quelli usati in precedenza, questo trigger non opera sull'inserimento ma sull'eliminazione dalla tabella, attraverso il comando `BEFORE DELETE ON ufficio`.

```
CREATE OR REPLACE TRIGGER check_elim_uff
before delete on ufficio
for each row
DECLARE
num_impiegati_uff ufficio.num_impiegati%type;
ufficio_imp impiegato.ufficio_impiegato%type;
overNumImp EXCEPTION;
BEGIN

select count(*) into num_impiegati_uff
from impiegato
where ufficio_impiegato = :old.num_ufficio
group by ufficio_impiegato;

if (num_impiegati_uff != 0) then
raise overNumImp;
end if;

EXCEPTION
when overNumImp then
```

```
raise_application_error(-20001,'Impossibile eliminare l''ufficio
poichè esistono degli impiegati assegnati ad esso');
end;
```

2.4.11 checkNumSped

Questo trigger impone una regola aziendale per cui ogni viaggio non può avere più di 3 spedizioni destinate ad ogni azienda esterna.

```
CREATE OR REPLACE TRIGGER checkNumSped
Before insert on spedizione
FOR EACH ROW
DECLARE
NumTotSpedizione int;
err1 exception;

BEGIN
select count(*) into NumTotSpedizione from viaggio join spedizione on cf_viaggio = cf_spedizione and data_spedizione
where data_viaggio = :new.data_spedizione
and cf_viaggio = :new.cf_spedizione;

if (NumTotSpedizione + 1 > 3) then
raise err1;
end if;

EXCEPTION
when err1 then
DBMS_OUTPUT.PUT_LINE('Hai raggiunto il numero massimo di spedizioni');
END;
```

2.5 Procedure

Le procedure sono atte ad automatizzare la nostra base di dati, al fine di gesirla al meglio traendone il massimo profitto.

2.5.1 nuovoDirettore

La procedura `nuovoDirettore` automatizza l'elezione di un nuovo direttore aziendale. L'input consiste in un codice fiscale da inserire, ossia quello del dipendente che si vuole rendere direttore. Ovviamente ci sono alcune regole da rispettare: il nuovo direttore non può essere un segretario e dovrà necessariamente avere un contratto a tempo indeterminato. Se tutto andrà per il giusto verso e i requisiti verranno rispettati, ci sarà il corretto aggiornamento del direttore attraverso il comando `UPDATE impiegato SET mansione`,e la conferma di questa operazione con la transazione `COMMIT`. Se cercheremo di rendere direttore il direttore stesso, la procedura ci avviserà del fatto che tale persona ha già il ruolo di direttore.

```
CREATE OR REPLACE PROCEDURE nuovoDirettore (CodFiscale varchar)
IS
nomeDir varchar2(30);
cognomeDir varchar2(30);
cod_fiscale VARCHAR(16);
giaDirettore EXCEPTION;
CodDirettore varchar2(30);

BEGIN
select cf_impiegato into CodDirettore from impiegato where mansione = 'Direttore';

if CodDirettore = CodFiscale then
raise giaDirettore;
end if;

select nome, cognome, cf  into nomeDir,cognomeDir,cod_fiscale from impiegato im
join dipendente dip on im.cf_impiegato = dip.cf
join contratto on cf_impiegato = cf_contratto
WHERE cf = codFiscale
AND MANSIONE != 'Segretario'
and tipo_contratto = 'Indeterminato';


update impiegato set mansione = 'Manager' where mansione = 'Direttore';
update impiegato set mansione = 'Direttore' where cf_impiegato = CodFiscale;
COMMIT;
DBMS_OUTPUT.PUT_LINE('L''impiegato ' || (nomeDir) ||' ' || (cognomeDir) ||
'è ufficialmente il nuovo direttore');

EXCEPTION
when NO_DATA_FOUND then
DBMS_OUTPUT.PUT_LINE('L''impiegato ' || (nomeDir) ||' ' || (cognomeDir) ||
'non può diventare direttore per le regole aziendali');

when giaDirettore then
DBMS_OUTPUT.PUT_LINE('L''impiegato ' || (nomeDir) ||' ' || (cognomeDir) ||' è gia Direttore');
END;
```

2.5.2 contaOre

Questa procedura non effettua nessuna modifica, ma è molto comoda al fine di visualizzare e monitorare le ore di presenza e di lavoro di ogni singolo impiegato. Basterà inserire in input il codice fiscale della persona interessata e la procedura ci dirà in risposta, quante ore di lavoro ha effettuato l'impiegato.

```
CREATE OR REPLACE PROCEDURE contaOre (nomeP varchar, cognomeP varchar)
IS
numOre NUMBER;
nomeDir VARCHAR2(30);
cognomeDir VARCHAR2(30);

BEGIN
select nome, cognome, sum(floor(((ora_uscita - ora_entrata)*24 - 1))) into nomeDir,cognomeDir,numOre
    from dipendente dip
```

```
    join impiegato im on dip.cf = im.cf_impiegato
    join presenza pr on im.cf_impiegato = pr.cf_presenza
    where nomeP = dip.nome and cognomeP = dip.cognome
group by nome, cognome;
DBMS_OUTPUT.PUT_LINE('L''impiegato ' || (nomeDir) ||' ' || (cognomeDir) ||
' ha effettuato '||(numOre) || ' ore di presenze');

exception

when no_data_found then
DBMS_OUTPUT.PUT_LINE('Non esiste il dipendente nel database');
END;
```

2.5.3 ScheduleViaggio

Questa è la procedura più articolata e utile dell'intera base di dati. Serve a schedulare un viaggio,inserendone anche la spedizione, ed il lotto destinati ad esso. La procedura prende in input diversi dati come la data del viaggio che vogliamo schedulare, i kilometri da fare,il fornitore da cui prendere i prodotti, l'azienda esterna a cui consegnarli, il tipo del prodotto,l'orario di consegna e il peso del prodotto: una serie di dati finalizzati all'esatta e completa schedulazione del viaggio. Quando inseriremo la data in cui vogliamo organizzare il nostro viaggio, sarà la procedura a verificare le disponibilità di tutti gli autisti per sceglierne uno che può fare il viaggio. Nel caso non fosse possibile schedulare il viaggio in quel giorno, la procedura ci suggerirà anche un giorno vicino in cui sarà possibile, per un autista, effettuare la spedizione. A fine procedura, dopo aver cercato tutti i dati interessati, la procedura inserirà nella base di dati il viaggio schedulato, con la propria spedizione, il lotto, ed ovviamente il codice fiscale identificativo dell'autista che lo effettua, committando il tutto.

```
CREATE OR REPLACE PROCEDURE ScheduleViaggio(dataViaggio date, kilometri number,pivaforn varchar,
pivaesterna varchar, orario_cons date, peso_in int, tipo_In varchar2)
IS
AutistaCandidato autista.cf_autista%type;
error1 exception;
error2 exception;
numSoste viaggio.num_soste%type := floor(kilometri/300);
DurataViaggio int := floor(Kilometri/90);
DurataIn viaggio.durata%type;
viaggioIn viaggio.data_viaggio%type;
numTracc integer := dbms_random.value(1000000000,9999999999);
contatore number := 0;
dataAssegnata viaggio.data_viaggio%type;
dataAssegnataFinale viaggio.data_viaggio%type;
--NUMERO RANDOM BOLLA CON 4 LETTERE E 6 NUMERI

random_value integer := dbms_random.value(100000,999999);
random_value_string varchar2(4) := dbms_random.string('U',4);
num_bolla_nuovo varchar2(10) := random_value_string || (random_value);

BEGIN
    while(AutistaCandidato IS NULL)
    loop
    BEGIN

    select cf_autista, (data_viaggio) into AutistaCandidato, dataAssegnata
    from viaggio join autista on cf_viaggio = cf_autista
    where data_viaggio <= (dataViaggio+contatore)
    and not
    (dataViaggio+contatore) between
    data_viaggio and (data_viaggio+(durata/24)+8/24) and
    not ((dataViaggio+contatore)+(durataViaggio/24)+8/24) between data_viaggio and
    (data_viaggio+(durata/24)+8/24)
    group by cf_autista,data_viaggio
    order by data_viaggio
    fetch first 1 row only;


    if (dataViaggio != dataAssegnata) then
        raise error1;
    elsif (dataViaggio = dataAssegnata) then
```

```
        raise error2;
    end if;

EXCEPTION
    when error1 then
    select durata, data_viaggio into durataIN, viaggioIn from viaggio where cf_viaggio = AutistaCandidato
    and data_viaggio <= dataViaggio order by data_viaggio
    fetch first 1 row only;

    dataAssegnataFinale := viaggioIn + durataIn/24 + 8/24;

    if (dataAssegnataFinale < dataViaggio) then
        select durata, data_viaggio into durataIN, viaggioIn from viaggio where cf_viaggio = AutistaCandidato
        and data_viaggio > dataViaggio order by data_viaggio
        fetch first 1 row only;
        dataAssegnataFinale := dataViaggio;
    end if;

    when error2 then
    DBMS_OUTPUT.PUT_LINE('Error 2: dataAssegnata ' || (dataAssegnata));
    select durata, data_viaggio into durataIN, viaggioIn from viaggio where cf_viaggio = AutistaCandidato
    and data_viaggio = dataViaggio;
    dataAssegnataFinale := dataAssegnata + durataIn/24 + 8/24;

    when NO_DATA_FOUND then
    contatore := +1;
    end;
    end loop;
    DBMS_OUTPUT.PUT_LINE('End : Il viaggio puo essere schedulato nella seguente data: '
    || to_char(dataAssegnataFinale,'yyyy-mm-dd hh24:mi') || ' Autista: '|| (AutistaCandidato)) ;

    insert into viaggio values (dataAssegnataFinale,AutistaCandidato,pivaforn,kilometri,numSoste,DurataViaggio);
    insert into spedizione values (numTracc,dataAssegnataFinale,AutistaCandidato,orario_cons,pivaesterna);
    insert into lotto values (num_bolla_nuovo,numTracc,peso_in,tipo_in);
    COMMIT;

END;
```

2.5.4 Promozione

La procedura si occupa di automatizzare la promozione di un segretario, rendendolo manager. Questo avverrà solo nel caso in cui il segretario godrà di un contratto che non sia part-time o apprendistato. Dopo aver testato che il codice fiscale dato in input, sia di un segretario che rispetti i giusti requisiti, la procedura committerà le operazioni di update sul campo mansione. Nel caso il segretario non rispetti i requisiti per essere promosso a manager, la procedurà effettuerà un `ROLLBACK` annullando tutte le operazioni fatte in precedenza.

```
CREATE OR REPLACE PROCEDURE promozione(nomep varchar2, cognomep varchar2)
IS
mansione_im impiegato.mansione%type;
nomeProm dipendente.nome%type;
cognomeProm dipendente.cognome%type;
cfprom dipendente.cf%type;
tp_contratto contratto.tipo_contratto%type;
BEGIN
select nome, cognome, cf_impiegato, mansione, tipo_contratto into nomeProm, cognomeProm, cfprom,
mansione_im, tp_contratto from impiegato im
join dipendente dip on dip.cf = im.cf_impiegato
join contratto contr on dip.cf = contr.cf_contratto
where nomep = nome and cognomep = cognome
order by data_inizio_contratto desc
fetch first 1 row only;

if lower(mansione_im) = 'segretario' and (tp_contratto = 'Indeterminato' or tp_contratto = 'Determinato') then
    UPDATE IMPIEGATO set mansione = 'manager' where cf_impiegato = cfprom;
    COMMIT;
```

```
    DBMS_OUTPUT.PUT_LINE('Il dipendente '|| (nomep) ||' ' ||(cognomep)||' è stato promosso correttamente');
else
    DBMS_OUTPUT.PUT_LINE('Il dipendente '|| (nomep) ||' ' ||(cognomep)||
    ' non è un segretario a tempo determinato o indeterminato');
    ROLLBACK;
    end if;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Il dipendente non esiste nel data_base');
end;
--LICENZIAMENTO DIPENDENTE (IMPIEGATO/AUTISTA)
CREATE OR REPLACE PROCEDURE licenziamento(codFiscale varchar)
IS
error1 EXCEPTION;
cfDir VARCHAR2(16);
BEGIN
select cf_impiegato into cfDir from impiegato where mansione = 'Direttore';

if cfDir = codFiscale then
    raise error1;
else
    delete from dipendente where cf = codFiscale;
    COMMIT;
end if;
DBMS_OUTPUT.PUT_LINE('ELIMINAZIONE DAL DATA-BASE ANDATA A BUON FINE');

EXCEPTION
when error1 then
raise_application_error(-20001,'NON PUOI LICENZIARE UN DIRETTORE SENZA AVERNE ELETTO UN ALTRO');
when no_data_found then
DBMS_OUTPUT.PUT_LINE('Non esiste il dipendente nel database');

END;
```

2.5.5 checkBonus

Questa procedura automatizza,dando come input un nome e cognome, l'aggiunta della tredicesima e quattordicesima sullo stipendio dell'interessato. L'aumento sarà del 30 per cento e verrà automaticamente calcolato dalla procedura stessa. L'aumento ovviamente, avverrà solo se è stato già inserito uno stipendio per quel mese.

```
CREATE OR REPLACE PROCEDURE check_bonus(nomeIn varchar2, cognomeIn varchar2)

IS
importo_stip stipendio.importo%type;
nomeSt dipendente.nome%type;
cognomeSt dipendente.cognome%type;
codice_stipendio stipendio.contratto_stipendio%type;
data_s stipendio.data_stipendio%type;
BEGIN

select nome, cognome , importo, contratto_stipendio, data_stipendio into nomeSt, cognomeSt, importo_stip,
codice_stipendio, data_s from dipendente dip join contratto cont
on dip.cf = cont.cf_contratto join stipendio st on cont.codice_contratto  = st.contratto_stipendio
where nome = nomeIn and cognome = CognomeIn
and data_stipendio between date'2022-06-01' and date'2022-06-30'
or data_stipendio between date'2022-12-01' and date'2022-12-31';

UPDATE stipendio set importo = importo_stip*1.3 where contratto_stipendio = codice_stipendio and
data_s = data_stipendio ; --30% in piu
COMMIT;
DBMS_OUTPUT.PUT_LINE('L''impiegato' || (nomeSt) ||' ' || (cognomeSt) ||'ha ricevuto l''aumento');
```

```
EXCEPTION
when NO_DATA_FOUND then
DBMS_OUTPUT.PUT_LINE('L''impiegato' || (nomeSt) ||' ' || (cognomeSt) ||' non ha ancora percepito lo stipendio');
END;
```

2.5.6 RinnovaContratto

Come ultima procedura, abbiamo ovviamente l'automazione del rinnovo contratto di un dipendente. Ci basterà inserire codice fiscale,il tipo e la durata del contratto, e la procedura rinnoverà automaticamente il contratto di quel dipendente con i dati in input, generando un codice contratto che rispetti il giusto formato. E' importante notare che nel caso si scelga di inserire un contratto a tempo indeterminato, la durata di esso sarà settata a `NULL`

```
CREATE OR REPLACE PROCEDURE rinnova_contratto(cf_in varchar, tipo_contratto_in varchar, durata_in number)

IS
data_contratto date;
random_value integer := dbms_random.value(1000000,9999999);
random_value_string varchar(3) := dbms_random.string('U',3);
tp_contratto contratto.tipo_contratto%type;
codice_contratto_nuovo varchar2(10) := random_value_string || (random_value);
err1 EXCEPTION;
mansioneIN impiegato.mansione%type;
BEGIN


select tipo_contratto,sysdate, mansione into tp_contratto, data_contratto, mansioneIN
from dipendente join contratto on cf = cf_contratto
join impiegato on cf_impiegato = cf
where cf = cf_in
order by data_inizio_contratto desc
fetch first 1 row only;

if tp_contratto = 'Indeterminato' and tipo_contratto_in = 'Indeterminato' then
    raise err1;
elsif tp_contratto != 'Indeterminato' then
    insert into contratto  values (codice_contratto_nuovo,cf_in,durata_in,tipo_contratto_in,data_contratto);
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Rinnovo del contratto avvenuto con successo.
    Ora il suo contratto è di tipo: ' || (tp_contratto) ||', con la mansione di: ' || (mansioneIN));

end if;

EXCEPTION
when err1 then
DBMS_OUTPUT.PUT_LINE('Rinnovo del contratto non avvenuto
poichè il dipendente ha gia un contratto a tempo indeterminato');
when no_data_found then
DBMS_OUTPUT.PUT_LINE('Il dipendente non esiste nel database');
END;
```

2.6 Viste

Le viste sono relazioni virtuali utili a regolare l'accesso da parte degli utenti del database. Le viste ci sono utili a mostrare porzioni di dati per cui vogliamo garantire l'accesso. Infatti le usiamo appositamente per la gestione degli utenti del DB.

2.6.1 SchemaUffici

Questa vista è utile al direttore,ai manager ed agli impiegati, poichè permette la visualizzazione dello schema di come sono distribuiti gli impiegati tra i vari uffici.

```
CREATE OR REPLACE VIEW SchemaUffici AS
select cf_impiegato,ufficio_impiegato,num_impiegati
from impiegato join ufficio on ufficio_impiegato = num_ufficio;
```

2.6.2 SchemaVeicoliAssociati

La vista è utile al direttore, manager, e autisti per visualizzare lo schema di come i veicoli sono associati al loro autista. La vista permetterà a loro di visualizzare, quindi, solo i veicoli associati e non quelli disponibili per un'eventuale associazione.

```
CREATE OR REPLACE VIEW SchemaVeicoliAssociati AS
select cf_autista,num_patente,targa as targa_veicolo_associato
from autista join veicolo on targa=targa_autista ;
```

2.6.3 VeicoliDisponibili

Questa vista permette ai manager ed al direttore di visualizzare tutti i veicoli appartenenti all'azienda, ma che non sono associati a nessun autista.

```
CREATE OR REPLACE VIEW Veicoli_disponibili
AS
select targa,tipo_veicolo,casa_produttrice from veicolo
minus
select targa,tipo_veicolo,casa_produttrice from veicolo join autista on targa=targa_autista;
```

2.6.4 ImpiegatiUfficioAttuale

Permette ai manager,agli impiegati ed al direttore di visualizzare,in modo semplice e veloce, gli uffici e quanti impiegati appartengono ad essi.

```
CREATE OR REPLACE VIEW ImpiegatiInUfficio_attuale
AS
select num_ufficio as numero_ufficio,count(*) as numero_attuale_impiegati
from impiegato join ufficio on num_ufficio = ufficio_impiegato
group by num_ufficio;
```

2.6.5 StoricoViaggi

Utile al direttore, ai manager e gli autisti, al fine di visualizzare tutto lo storico dei viaggi che l'azienda effettua, con i rispettivi autisti.

```
CREATE OR REPLACE VIEW StoricoViaggi
AS
select data_viaggio,cf_viaggio
from viaggio join spedizione on data_viaggio = data_spedizione and cf_viaggio = cf_spedizione;
```

2.7 Data Control Language

Il data control language riguarda la gestione degli utenti e quali permessi vogliamo conferire ad essi. Come possiamo notare, il direttore ha tutti i permessi possibili sul DB aziendale.
Il manager ha il permesso di schedulare i viaggi degli autisti e di monitorare uffici e veicoli associati rispettivamente ad impiegati e autisti, oltre che controllare quali veicoli siano disponibili per eventuali nuovi autisti. Autisti e impiegati hanno pochi permessi, che si limitano alla visualizzazione, rispettivamente, di veicoli associati e uffici associati ai dipendenti dell'azienda.

```
GRANT ALL PRIVILEGES TO utenteDirettore;

--PRIVILEGI MANAGER
GRANT SELECT ON SchemaUffici TO ManagerAziendale;
GRANT SELECT ON SchemaVeicoliAssociati TO ManagerAziendale;
GRANT SELECT ON VeicoliDisponibili TO ManagerAziendale;
GRANT SELECT ON ImpiegatiInUfficio_attuale TO ManagerAziendale;
GRANT SELECT  ON StoricoViaggi to ManagerAziendale;
GRANT EXECUTE ON ScheduleViaggio TO ManagerAziendale;
GRANT EXECUTE ON contaOre TO ManagerAziendale;

--PRIVILEGI AUTISTA
GRANT SELECT ON SchemaVeicoliAssociati to AutistaAziendale;
GRANT SELECT ON StoricoViaggi to AutistaAziendale;

--PRIVILEGI IMPIEGATO
GRANT SELECT ON SchemaUffici to ImpiegatoAziendale;
GRANT SELECT ON ImpiegatiInUfficio_attuale TO ImpiegatoAziendale;
```