



"McDonald's"



Università degli Studi di Napoli Parthenope, CdS in Informatica

Partecipanti al progetto:
Vincenzo Bucciero 0124002256
Camilla De Martino 0124002258
Noemi Ruocco 0124002445

A.A. 2021/2022
Consegna:

Indice

Elenco delle figure	4
Elenco delle tabelle	5
1 Progettazione	6
1.1 Sintesi dei requisiti	6
1.2 Glossario	7
1.3 Diagramma EE/R	8
1.3.1 <i>Modello concettuale completo</i>	8
1.3.2 <i>Gestione del personale</i>	9
1.3.3 <i>Gestione degli ordini</i>	10
1.3.4 <i>Gestione del magazzino</i>	12
1.4 Diagramma relazionale	13
1.4.1 <i>Gestione del personale</i>	14
1.4.2 <i>Gestione degli ordini</i>	15
1.4.3 <i>Gestione del magazzino</i>	15
1.5 Utenti	16
1.6 Operazioni degli utenti	17
1.7 Volumi	22
1.8 Vincoli di Integrità	23
1.8.1 Vincoli di integrità statici	23
1.8.2 Vincoli di integrità dinamici	24
1.9 Verifica di normalità	25
1.9.1 Prima forma normale	25
1.9.2 Seconda forma normale	25
1.9.3 Terza forma normale	26
1.10 Possibili estensioni	26

2	Implementazioni	27
2.1	Creazione degli utenti	27
2.2	Data Control Language - DCL	27
2.3	Data Definition Language - DDL	28
2.3.1	DROP per evitare conflitti	28
2.3.2	PERSONALE	30
2.3.3	TURNI	30
2.3.4	STIPENDIO	31
2.3.5	ORDINE	32
2.3.6	CLIENTE_ISCRITTO	33
2.3.7	PRODOTTO	33
2.3.8	MENU_STANDARD	34
2.3.9	OFFERTE	34
2.3.10	ordine_contiene_prod	35
2.3.11	offerta_contiene_prod	35
2.3.12	MAGAZZINO	36
2.3.13	ORDINE_RIFORNIMENTO	36
2.3.14	FATTURA	37
2.3.15	FORNITORE	38
2.3.16	MERCE	38
2.3.17	LOTTO	39
2.4	Data Manipulation Language - DML	39
2.5	Trigger	41
2.5.1	check_turno	41
2.5.2	check_stipendio	42
2.5.3	check_menu	42
2.5.4	Accesso_offerta	43
2.5.5	Check_Magazzino	44
2.5.6	Check_offerta_valida	44
2.5.7	Check_doppio_stipendio	45
2.5.8	Check_eta	46
2.5.9	Check_Max	46
2.5.10	Check_Ultimo_Turno	47
2.5.11	Check_cassa	48
2.6	Procedure	49
2.6.1	BonusCassiere	49
2.6.2	OffertaProdotto	51
2.6.3	Promozione	52
2.6.4	totale_ordine	55
2.6.5	trova_merce	56

2.6.6	crea_ordine	57
2.6.7	calcolo_giacenza	60
2.7	Viste	64
2.7.1	Componenti dei prodotti	64
2.7.2	Informazioni nutrizionali dei prodotti	65
2.7.3	Composizione offerte	65
2.7.4	Magazzino	66
2.7.5	Turni di lavoro	66

Elenco delle figure

1.1	Diagramma EE/R	8
1.2	Diagramma EE/R - Gestione personale	9
1.3	Diagramma EE/R - Gestione ordini	10
1.4	Diagramma EE/R - Gestione magazzino	12
1.5	Diagramma relazionale	13
1.6	Diagramma relazionale - Gestione personale	14
1.7	Diagramma relazionale - Gestione ordini	15
1.8	Diagramma relazionale - Gestione magazzino	15
1.9	Operazioni possibili dai vari utenti	17

Elenco delle tabelle

1.1	Glossario	7
1.2	Tavola dei volumi	22

Capitolo 1

Progettazione

Nelle seguenti pagine si riporta la documentazione relativa alla progettazione di un sistema di basi di dati riguardante la gestione di un fast food.

I requisiti presenti sono frutto di un attenta analisi sulla gestione del fast food e delle regole aziendali da implementare.

1.1 Sintesi dei requisiti

La situazione che si vuole rappresentare è quella di una gestione di un fast food, nello specifico si vogliono rappresentare le diverse parti che coinvolgono la sua gestione, ovvero:

- la gestione del personale
- la gestione degli ordini
- la gestione del magazzino

Per *gestione del personale* si intende la possibilità di avere informazioni relative ai dipendenti del fast food, di cui si registrano i dati anagrafici principali, i turni di lavoro effettuati e lo stipendio. Il nostro database stabilisce un numero finito di dipendenti, i quali possono assumere due ruoli diversi.

Per *gestione degli ordini* si intende tutto ciò che è strettamente legato all'ordine, da ciò che contiene all'addetto che se ne occupa. Questa rappresenta di fatto la parte principale della base di dati. Un aspetto importante di questa parte di database è la gestione delle offerte contenute all'interno degli ordini, esse infatti sono riservate unicamente ai clienti iscritti al fast food.

Infine, per *gestione del magazzino*, si intende tutto ciò che riguarda la gestione delle singole merci di cui ha bisogno il fast food per poter operare. Riportiamo infatti, oltre i dati del fornitore, le informazioni necessarie a descrivere tutto ciò che è presente nel magazzino e quello che è stato venduto.

1.2 Glossario

All'interno del database non sono stati utilizzati termini appartenenti ad un particolare gergo tecnico ma, per evitare fraintendimenti, di seguito è inserito un glossario 1.1 in cui sono riportati attributi con nomi diversi a seconda della tabella in cui sono stati inseriti.

Termine	Sinonimi	Descrizione
tessera	cod_personale, codice_personale, addetto	Numero di tessera che identifica il dipendente
e_mail	cliente	Identificativo dei clienti iscritti
num_ordine	ordine, cod_ordine	Numero che identifica l'ordine
codice_offerta	offerta, cod_offerta	Codice identificativo per l'offerta
codice_menu	cod_menu_std, cod_menu	Codice identificativo dei menù standard
cod_barre_prodotto	prodotto, prod_riferito	Codice identificativo del prodotto
cod_area	codice_area_mag	Codice che identifica quella specifica area all'interno del magazzino
nome_merce	componente, merce, merce_contenuta	Nome che identifica in modo univoco la specifica merce, intesa come l'ingrediente specifico del prodotto
partita_iva	fornitore, rif_fornitore	Codice che identifica il fornitore univocamente
cod_ordine_rif	ordine_rif	Codice che identifica l'ordine rifornimento

Tabella 1.1: Glossario

1.3 Diagramma EE/R

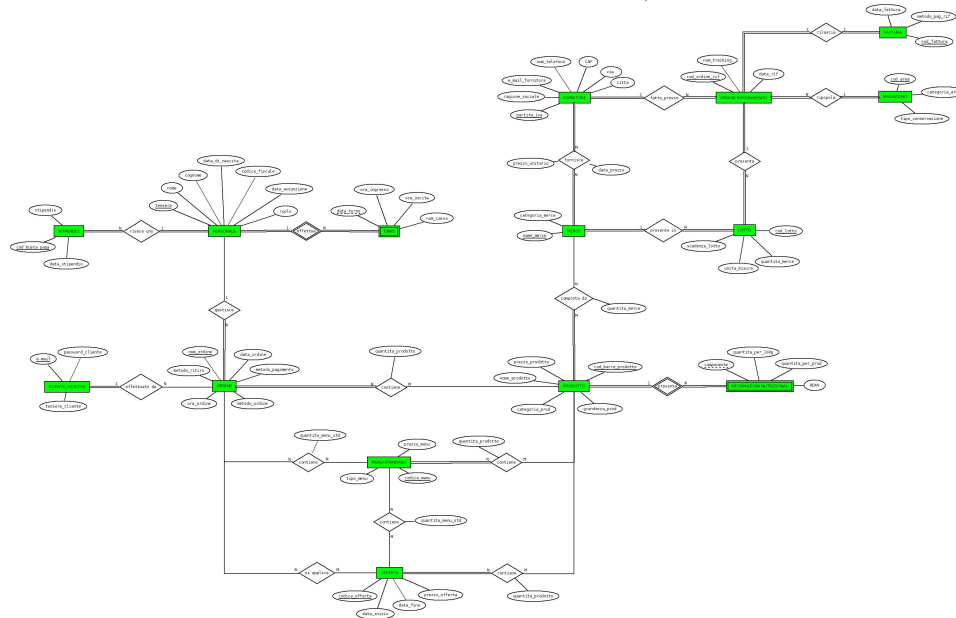
Di seguito viene riportato il diagramma EE/R, estensione del modello E/R (*entity-relationship*), che rappresenta lo schema concettuale del database.

Per rendere più semplice la comprensione del diagramma, oltre alla figura del modello completo, si approfondiranno anche le tre macro aree in maniera separata.

Ogni entità rappresentata nel diagramma possiede attributi che descrivono le loro proprietà. Inoltre vengono espresse anche le totalità delle relazioni tra entità, che possono godere di diversi tipi di molteplicità.

1.3.1 Modello concettuale completo

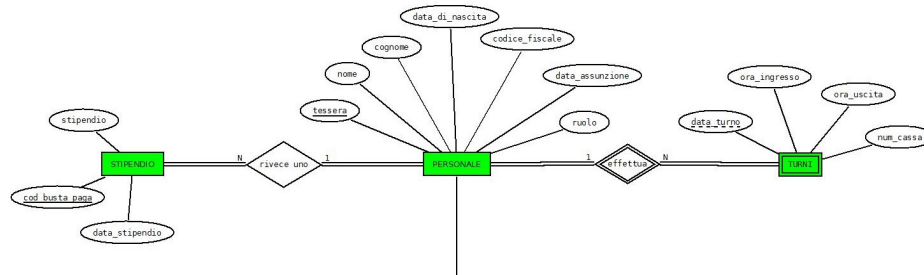
Figura 1.1: Diagramma EE/R



Dal modello concettuale completo è possibile avere un'idea chiara sul funzionamento del database.

1.3.2 Gestione del personale

Figura 1.2: Diagramma EE/R - Gestione personale



PERSONALE

Legate all'entità personale vi sono tutte le informazioni necessarie per l'identificazione del dipendente e le informazioni legate strettamente al suo ruolo, quali il suo numero tessera, il suo ruolo (si considerano solo due ruoli effettivi, ossia *Dipendente* e *Manager*). Inoltre viene registrata anche la data della sua assunzione.

TURNI

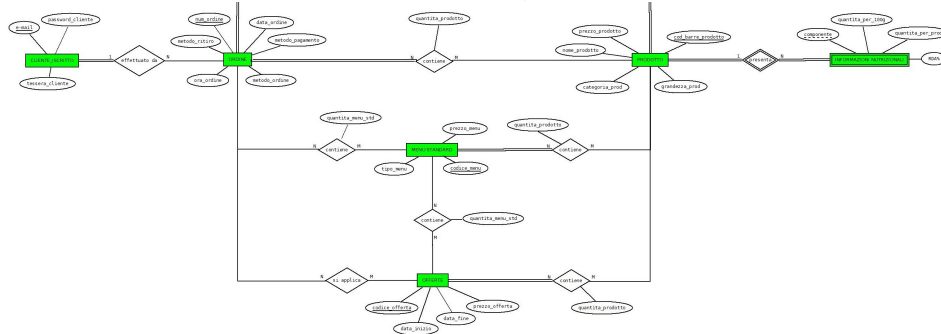
L'entità che rappresenta i turni è un'entità debole. Le informazioni che vengono registrate al suo interno sono quindi il numero di tessera del personale e la data in cui quest'ultimo ha effettuato il turno, oltre all'orario di ingresso e di uscita, per verificare le effettive ore di lavoro effettuate.

STIPENDIO

L'entità stipendio comprende tutte le informazioni necessarie al fine di identificare univocamente a chi appartiene quella busta paga, il suo valore (che dipende dal ruolo che si svolge all'interno dell'azienda) e quando è stata emessa.

1.3.3 Gestione degli ordini

Figura 1.3: Diagramma EE/R - Gestione ordini



ORDINE

Ordine contiene le informazioni necessarie ad identificare univocamente ogni sua istanza, oltre ai riferimenti al personale che lo ha preso in carico e al cliente che lo ha richiesto, se quest'ultimo è presente nell'entità di *CLIENTE_ISCRITTO*.

Un ordine può contenere singolarmente prodotti, menù standard o offerte, oppure combinazioni di essi. Per tale motivo ci sono tre diverse associazioni legate ad esso: *ordine_contiene_prod* la quale indica se sono presenti prodotti singoli all'interno dell'ordine, *ordine_contiene_menu* la quale informa della presenza di menù standard nell'ordine e *ordine_contiene_offerte* che indica se sono state utilizzate offerte all'interno dell'ordine effettuato.

Dunque all'interno dello stesso ordine è possibile trovare i singoli prodotti, i menù standard, combinazioni o ripetizioni di questi. Inoltre è anche possibile che ci siano offerte riguardanti un prodotto o menù standard e avere gli stessi prodotti o menù standard, a cui facevano riferimento le offerte, presi singolarmente. Questo perché per regole aziendali si è deciso che in uno stesso ordine ci possono essere diverse offerte, ma non può comparire due volte la stessa.

Quindi nel caso in cui si vogliano acquistare due prodotti presenti in un offerta o si effettuano due ordini per utilizzare due volte quell'offerta, oppure si acquisterà una volta il prodotto tramite offerta e una volta prendendo il prodotto al suo prezzo standard.

OFFERTE

Quest'entità rappresenta l'insieme di offerte che accessibili ai clienti iscritti, in quanto i semplici clienti non hanno la possibilità di utilizzarle.

Questa è indipendente dall'ordine in quanto rappresenta tutte le offerte ideate dal fast food. Possono esserci offerte che contengono singolarmente prodotti e menù standard, oppure offerte che comprendono entrambi, per sapere a cosa si lega un'offerta esistono le associazioni *offerta_contiene_prod* e *offerta_contiene_menu*.

Bisogna tenere bene a mente che, come visto dal diagramma E/R, le offerte hanno un proprio prezzo fisso, ciò sta ad indicare che il loro prezzo viene deciso indipendentemente dai singoli prezzi dei prodotti o dei menù standard che ne fanno parte.

PRODOTTO

Quest'entità rappresenta i diversi prodotti che è possibile acquistare. Possono rientrare nelle categorie panini, bibite, extra (come ad esempio patatine fritte, chicken nuggets, etc.) o dolci.

Oltre al nome del prodotto e alla sua categoria, sono riportati informazioni riguardo la sua grandezza e al suo prezzo. Inoltre ad ogni prodotto sono associate le proprie informazioni nutrizionali.

MENU_STANDARD

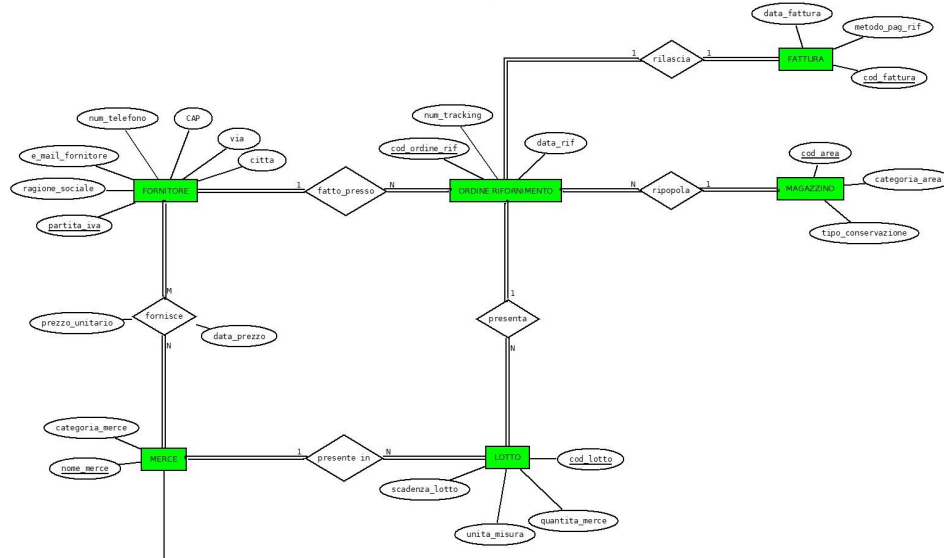
Quest'entità rappresenta tutti i pacchetti menù ideati dal fast food.

Attraverso l'associazione con l'entità PRODOTTI, chiamata *menu_contiene_prod*, è possibile visualizzare quali sono i prodotti che vengono inseriti nel pacchetto standard ideato.

Il suo prezzo, trattandosi di un pacchetto menù, è indipendente dai singoli prezzi legati ai prodotti che contiene al suo interno.

1.3.4 Gestione del magazzino

Figura 1.4: Diagramma EE/R - Gestione magazzino



MAGAZZINO

Con magazzino si vuole indicare il luogo fisico in cui sono presenti le scorte delle merce necessarie per il funzionamento del fast food.

Questo è diviso in aree, per cui il codice area identificherà in modo univoco la sezione di magazzino interessata. Inoltre per ogni area è segnalato il tipo di conservazione per la merce che vi fa parte.

ORDINE_RIFORNIMENTO

In quest'entità rientrano le informazioni necessarie all'identificazione dell'ordine rifornimento, quindi il suo codice, il suo numero di tracking e la data in cui è stato effettuato.

FATTURA

Strettamente legata all'ordine rifornimento c'è la fattura, che contiene informazioni su quando è stata rilasciata, sul metodo pagamento utilizzato e sul codice per identificarla.

In questa entità vi sono le informazioni utili per identificare il fornitore. Poiché si trattano di informazioni legate univocamente al fast food McDonald's, i fornitori riportati in questa entità non sono tutti quelli esistenti per quella specifica merce, ma solo quelli con cui l'azienda ha a che fare.

Un lotto contiene le informazioni sulla merce contenuta, sulla sua scadenza, sul quantitativo di merce presente nel singolo lotto e ovviamente il codice che lo identifica.

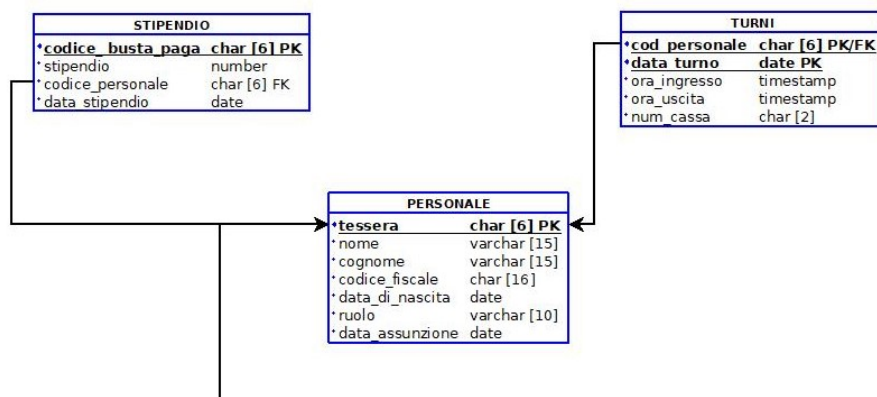
Per merce si intende la specifica merce necessaria a comporre un prodotto che può essere comprato all'interno del fast food. Oltre al suo nome, che lo identifica, viene specificata anche la categoria della merce.

Analogamente al diagramma concettuale, anche per lo schema relazionale, dopo aver inserito una figura che renda possibile osservarlo nel suo complesso, si inseriranno altre immagini che concernono singolarmente i tre tipi di gestione da analizzare.

1.4.1 *Gestione del personale*

Di seguito è riportata nello specifico la traduzione della parte relativa alla gestione del personale.

Figura 1.6: Diagramma relazionale - Gestione personale



1.5 Utenti

Al fine di rendere il nostro database una base dati chiara ed efficiente, si è reso necessario suddividere in quattro categorie gli utenti che possono accedere.

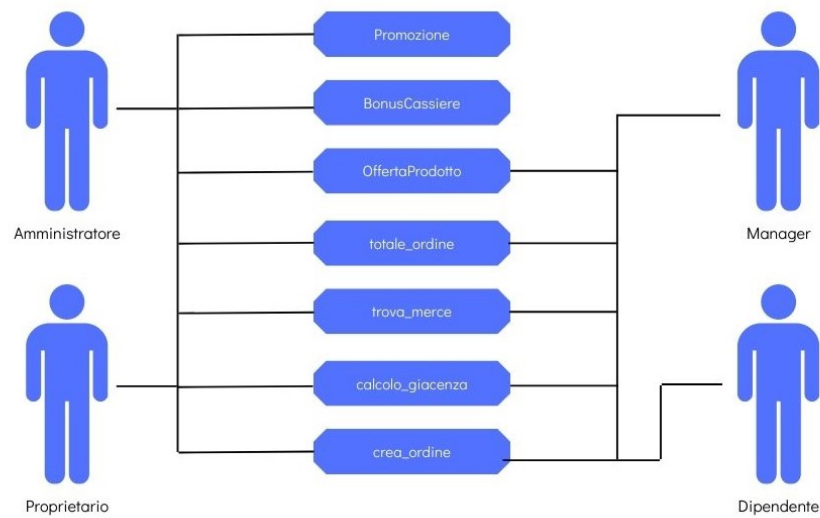
Il primo utente è l'**amministratore** del database, una persona esterna all'azienda che si occupa dell'implementazione della base di dati, e che di conseguenza ha accesso a tutti i privilegi possibili, quali la visualizzazione dei dati e la modifica di essi.

Il secondo utente è il **proprietario** del fast food, che ha la possibilità di eseguire ogni operazione prevista, di accedere a tutte le tabelle coinvolte nelle operazioni, ma può apportare modifiche solo alla tabella STIPENDIO, in quanto uno dei suoi attributi può essere modificato o meno in una delle procedure implementate.

Discorso analogo per gli altri due utenti, che sono **manager** e **dipendente**. Anche queste non possono in alcun modo modificare il database, in quanto hanno solo i permessi di eseguire determinate operazioni e di accedere alle tabelle coinvolte nelle operazioni a loro disponibili.

1.6 Operazioni degli utenti

Figura 1.9: Operazioni possibili dai vari utenti



Escludendo le operazioni base, quali l’inserimento, l’aggiornamento o la modifica di tuple e la visualizzazione di informazioni tramite viste, le operazioni più complesse che possono essere effettuate dagli utenti vengono implementate tramite procedure.

Nel diagramma seguente sono rappresentati i casi d’uso, in cui sono visibili le operazioni implementate e gli utenti coinvolti per ogni operazione.

Più nello specifico seguono le schede delle diverse operazioni:

Operazione	BonusCassiere
Scopo:	Attribuisce un BONUS del 20% allo stipendio attuale del dipendente (esclusi i manager) che nel mese precedente ha fatto più ordini
Argomenti:	Non prende argomenti in input.
Risultato:	Si stampa il corretto inserimento del bonus allo stipendio
Usa:	PERSONALE, ORDINE, STIPENDIO, TURNI
Modifica:	Si modifica il valore di Stipendio , al quale viene aggiunto il valore calcolato del bonus del 20%.
Prima:	Lo stipendio dato ad un dipendente è sempre quello base.
Poi:	Al dipendente che nel mese precedente già ha assunto uno stipendio e che ha effettuato il maggior numero di ordini, viene aggiunto il bonus e mostrato a video.

Operazione	OffertaProdotto
Scopo:	Trova il menù meno venduto ed applica al menù che possiede l'offerta più costosa, un'ulteriore sconto del 50%.
Argomenti:	Non prende nessun argomento in input.
Risultato:	Mostra a video a quale menù è stata applicato uno sconto del 50% sull'offerta più costosa.
Errori:	Viene mostrato a video un'errore (Error -20001) nel momento in cui non viene venduto nessun menù.
Usa:	OFFERTE, MENU_STANDARD e le tabelle di transizione <code>ordine_contiene_menu</code> ed <code>offerta_contiene_menu</code> che mostrano i legami tra i menù e le entità di Ordine ed Offerte.
Modifica:	Il valore dell'offerta alla quale viene applicato un'ulteriore sconto.
Prima:	Offerta non scontata
Poi:	Offerta scontata per un mese

Operazione	Promozione
Scopo:	Promuove a MANAGER il dipendente che ha venduto più menù con il prezzo più alto (in caso di più dipendenti che rispettano questo requisito, si promuove il dipendente con data di assunzione più vecchia.
Argomenti:	Non prende argomenti in input.
Risultato:	Viene mostrato a video la corretta esecuzione della promozione a manager se il dipendente rispecchia i requisiti prestabiliti.
Usa:	PERSONALE, TURNI, ORDINE, MENU_STANDARD, e la conseguente tabella di transizione <code>ordine_contiene_menu</code> .
Modifica:	Si cambia il ruolo del dipendente (Dipendente -> Manager).
Prima:	Il dipendente era un cassiere
Poi:	Il dipendente viene promosso a manager

Operazione	totale_ordine
Scopo:	Calcolare il prezzo totale dell'ordine
Argomenti:	Viene preso in input il numero dell'ordine
Risultato:	Stamperà il prezzo totale
Usa:	ORDINE, PRODOTTO, MENU_STANDARD, OFFERTE e le tabelle di transizione tra ordine e le altre tre entità
Modifica:	Non si modificano attributi in quanto il totale viene stampato a video
Prima:	L'ordine effettuato non aveva un prezzo totale ma tanti prezzi quanti prodotti conteneva
Poi:	Si conosce perfettamente il costo totale dell'ordine

Operazione	trova_merce
Scopo:	Crea un cursore che scorre tutti i valori di nome_merce che trova e per ognuno di loro richiama calcolo_giacenza
Argomenti:	Non prende argomenti in input
Risultato:	Il valore della giacenza sarà calcolato per ogni valore di nome_merce, e se necessario verrà creato un ordine di rifornimento legato ad essa
Usa:	MERCE, la procedura calcolo_giacenza
Modifica:	Non modifica valori, ma nel caso dovesse richiamarsi la crea_ordine allora si genera un nuovo ordine rifornimento
Prima:	Giacenza della merce sconosciuta
Poi:	Si conosce la giacenza di ogni merce e si sono generati eventuali ordini per il rifornimento per la merce che ne ha necessità

Operazione	crea_ordine
Scopo:	Genera un nuovo ordine rifornimento nel momento in cui i pezzi rimanenti di una merce sono inferiori a 10, trovando però il fornitore che nella data più recente aveva il prezzo minore per la vendita di quella specifica merce
Argomenti:	Prende come input il nome della merce che risulta avere meno di 10 pezzi rimanenti dalla sua giacenza
Risultato:	Viene inserito un nuovo ordine per quella merce nella tabella di ORDINE_RIFORNIMENTO
Usa:	FORNITORE, MERCE, la tabella di trasizione fornitore_fornisce_merce presente tra loro, e ORDINE_RIFORNIMENTO
Prima:	I pezzi di quella specifica merce stavano per finire
Poi:	Si ha un nuovo ordine rifornimento per quella merce

Operazione	calcolo_giacenza
Scopo:	Calcola la giacenza di una specifica merce
Argomenti:	Prende il nome della merce di cui calcolare la giacenza in input
Risultato:	Stampa a video la giacenza di quella merce
Usa:	Utilizza tutte le tabelle tra merce e ordine per calcolare il totale dei pezzi venduti di quella specifica merce, mentre tramite le tabelle tra merce e magazzino calcola il totale dei pezzi acquistati sommati a quelli sempre presenti in magazzino come scorta. Il risultato è la differenza tra i pezzi acquistati e quelli venduti. Nel caso in cui la giacenza dovrebbe essere inferiore a 10 pezzi richiama la procedura crea_ordine
Prima:	Non si conosce la giacenza di quel prodotto
Poi:	Si conosce la giacenza di quel prodotto, e nel caso risulti minore di 10 pezzi si richiama la procedura per generare un ordine rifornimento di quella merce

1.7 Volumi

Nella tabella che segue sono indicate le tabelle presenti all'interno del database, il numero verosimile di tuple che conterranno una volta che il database sarà funzionante e l'incremento atteso per ognuna di loro in un tempo prefissato.

Poiché come regola aziendale non ci possono essere più di 30 dipendenti, si stima di assumere altri 15 dipendenti nell'anno e che poi l'incremento sia nullo.

Tabella	Tipo	Volume	Incremento	Periodo
PERSONALE	E	15	15	anno
TURNI	ED	15	60	mese
STIPENDIO	E	15	15	mese
CLIENTE_ISCRITTO	E	15	15	mese
ORDINE	E	24	30	giorno
PRODOTTO	E	15	6	anno
INFORMAZIONI_NUTRIZIONALI	ED	15	42	anno
MENU_STANDARD	E	15	3	anno
OFFERTE	E	24	12	anno
MERCE	E	15	2	anno
FORNITORE	E	17	3	anno
ORDINE_RIFORNIMENTO	E	15	30	bimestre
FATTURA	E	15	30	bimestre
LOTTO	E	18	50	bimestre
MAGAZZINO	A	17	2	anno
ordine_contiene_prod	A	15	20	giorno
ordine_contiene_menu	A	15	15	giorno
ordine_contiene_offerte	A	20	15	giorno
menu_contiene_prod	A	15	3	anno
offerta_contiene_prod	A	15	6	anno
offerta_contiene_menu	A	15	6	anno
prodotto_composto_da	A	18	20	anno
fornitore_fornisce_merce	A	18	3	anno

Tabella 1.2: Tavola dei volumi

- **Tipo:** **E-ED-A** rappresentano rispettivamente le entità, le entità deboli e le tabelle di transizione.

- **Incremento:** è l'incremento atteso del numero di tuple.
- **Periodo:** è il periodo entro il quale l'incremento è atteso.

1.8 Vincoli di Integrità

Vengono qui riportati tutti i vincoli, statici e dinamici, del database. Da questi sono esclusi i vincoli di chiave primaria e di chiave esterna in quanto banali.

1.8.1 Vincoli di integrità statici

Sono **statici**, i vincoli di integrità che pongono dei limiti ai valori assumibili dagli attributi delle entità.

Gestione del personale

- L'attributo ruolo presente per il personale può avere come valori soltanto 'Manager' o 'Dipendente'.
- Non possono esserci più di trenta dipendenti all'interno del fast food, manager compresi.
- Per quanto riguarda lo stipendio, non è possibile inserire uno stipendio inferiore alla soglia minima di 1200.00euro.
- Non è possibile assumere un dipendente minorenni o che abbia superato i 50 anni di età.

Gestione degli ordini

- I metodi di pagamento relativi all'ordine sono solo "Carta di Credito" o "Contanti".
- I metodi di ordinazione possibile per effettuare l'ordine sono tramite "cassa", "totem" o "drive".
- I metodi di ritiro dell'ordine sono "da portare" o "da mangiare".
- Non è possibile avere lo stesso codice offerta ripetuto due volte all'interno dello stesso ordine.

- I diversi tipi di menù standard che possono essere inseriti nell'ordine, che sono essenzialmente dei menù già composti, rientrano solo nelle categorie "famiglia", "doppio" e "singolo".
- Il prodotto può rientrare nelle categorie "panini", "bibite", "extra" e "dolci", mentre la sua grandezza può essere "piccolo", "medio", "grande" o "standard", nel caso in cui non abbia diverse grandezze tra cui scegliere.
- Le componenti presenti in informazioni nutrizionali sono "calorie(kcal)", "carboidrati", "fibre", "grassi", "proteine", "sale" e "zuccheri".

Gestione del magazzino

- Non è possibile avere un ordine rifornimento in cui compaiono fornitori diversi. Analogamente non ci sono ordini in cui compaiono lotti di categorie diverse, tuttavia, se lo stesso fornitore possiede merci diverse ma della stessa categoria allora possono comparire nello stesso ordine.
- Le categorie presenti all'interno del magazzino rientrano in "bibite", "carni", "dolci", "formaggi", "frutta", "ortaggi", "pane", "uova" e "verdure".
- Le categorie possibili di merce sono "acqua", "bibite", "carne", "cereali", "dolce", "frutta", "latticini", "ortaggi", "patate", "pollo", "uova", e "verdura".
- Per quanto riguarda il metodo pagamento relativo all'ordine rifornimento, questo può essere soltanto "carta" o "contanti".

1.8.2 Vincoli di integrità dinamici

I vincoli **dinamici**, sono vincoli di integrità che possono variare nel tempo o che riguardano alcune *regole di business*.

Gestione del personale

- Non è possibile effettuare un turno da più di otto ore.
- Il dipendente non può iniziare un nuovo turno o presenza se quello precedente non è ancora terminato.
- Lo stipendio di un dipendente deve essere adeguato al rispettivo ruolo.
- Il numero di cassa 0 è riservato ai Manager, per una *politica aziendale*.

Gestione degli ordini

- Per una *politica aziendale* non è possibile avere un ordine in cui non è presente un cliente iscritto ma che è comunque legato ad un codice offerta.
- Non è possibile inserire in un ordine offerte scadute.

Gestione del magazzino

- Non possono essere inseriti lotti scaduti in magazzino.
- È regola aziendale che per ogni merce non si abbia una giacenza inferiore a 10, in tal caso verrà avviato un ordine per tale prodotto.

1.9 Verifica di normalità

Una delle cose più importanti da dover analizzare nel momento della normalizzazione di un database è che non ci siano dipendenze funzionali anomale, limitando la ridondanza. Il processo di normalizzazione sottopone uno schema di relazione a una serie di test per “certificare” se soddisfa una data forma normale

1.9.1 Prima forma normale

La **prima forma normale** (o **1NF**) è stata definita per non permettere l’uso di attributi multivalore, di attributi composti e delle loro combinazioni. Il nostro database rispetta la prima forma normale in quanto sono presenti tutti campi unici e atomici. Tutti i campi di tipo **DATE**, sono considerati atomici convenzionalmente in ORACLE DBMS.

1.9.2 Seconda forma normale

Questa forma normale (detta anche **2NF**) si applica per le relazioni in cui la chiave primaria è costituita da più attributi, gli attributi non-chiave non devono dipendere funzionalmente solo da una parte della chiave primaria. Nel nostro database abbiamo 9 entità, 7 di queste di transizione, con chiave composta ad esempio: **INFORMAZIONI_NUTRIZIONALI** e **TURNI**. In ognuna di esse è rispettata la 2NF poiché ogni attributo non primo di esse, è dipendente funzionalmente dall’intera chiave composta.

1.9.3 Terza forma normale

La **terza forma normale** (o **3NF**) si basa sul concetto di dipendenza transitiva. Ovvero uno schema è in 3NF quando tutti gli attributi di una relazione dipendono funzionalmente solo dalla chiave primaria della relazione e non da attributi non-chiave. All'interno del nostro database è stato controllato che non ci fossero dipendenze funzionali anomale, e ciò ci permette di dire che la terza forma normale è rispettata. Rispettando tale forma, possiamo dire che lo schema è in **Boys-Codd Normal Form**.

1.10 Possibili estensioni

Pensando ad una possibile estensione si può immaginare il più realistico caso in cui il proprietario del fast food decide di aprire una seconda sede. In tal caso, il criterio per cui non possono esserci oltre trenta dipendenti dovrà essere eliminato o revisionato, in quanto si potrebbe decidere semplicemente di raddoppiare il numero massimo di dipendenti consentito.

Un'altra aggiunta potrebbe essere quella di creare una nuova sede anche per il magazzino fisico, o di allargare quello già esistente, portando quindi ad una modifica della gestione del magazzino attuale, in cui si dovrà poi specificare non solo l'area magazzino, ma anche in quale sede viene portata la merce presente negli ordini di rifornimento.

Capitolo 2

Implementazioni

Di seguito si riportano tutte le implementazioni necessarie a garantire il corretto funzionamento del database, sulla base dei requisiti ottenuti in fase di progettazione.

L'ordine in cui compariranno le varie entità, associazioni, e altre componenti del database non rappresenta l'effettivo ordine in cui vanno eseguite, bensì sono posizionate in questo ordine per poter seguire in maniera più lineare il funzionamento e il legame tra le diverse entità.

2.1 Creazione degli utenti

Come spiegato precedentemente, gli utenti del database sono quattro, e sono stati creati come segue.

```
create user amministratore identified by admin;  
create user proprietario identified by proprietario;  
create user manager identified by manager;  
create user dipendente identified by dipendente;
```

2.2 Data Control Language - DCL

Strettamente legati alla creazione degli utenti vi sono i privilegi a loro associati.

Come discusso nella parte progettuale, gli utenti non hanno tutti gli stessi permessi.

L'unico utente ad avere tutti i permessi possibili è **l'amministratore/admin**, in quanto è l'effettivo creatore della base di dati ed è una persona esterna all'azienda che avrà quindi solo il ruolo di creare la base di dati a partire dalle esigenze del proprietario del fast food.

Immediatamente dopo si trova il **proprietario** del fast food, che ha invece la possibilità di mandare in esecuzione tutte le procedure presenti, quindi può compiere ogni procedura presente nel database. Per assicurare ciò, ovviamente gli viene concesso l'accesso ad ogni tabella coinvolta.

In particolare, poiché la procedura che assegna un bonus sullo stipendio modifica il valore dello stesso gli è concessa anche la possibilità di modificare la tabella STIPENDIO. Analogamente, la procedura che promuove un dipendente a manager modifica l'attributo "ruolo", dunque ha il diritto di modificare anche la tabella PERSONALE.

Un gradino più in basso ci sono i **manager**, che hanno gli stessi permessi del proprietario, ma non possono eseguire le procedure per il calcolo del bonus e per la promozione a manager di un dipendente, per tale motivo non hanno i permessi di modifica sulle tabelle STIPENDIO e PERSONALE.

Infine ci sono i **dipendenti**, i quali possono solo mandare in esecuzione la procedura che serve a calcolare il costo totale di un ordine, di conseguenza hanno i permessi di accedere solo alle tabelle interessate in quella procedura.

2.3 Data Definition Language - DDL

Il DDL rappresenta uno specchio dello schema relazionale.

Tutte le entità e le relazioni tra loro sono state create mediante tante istruzioni **CREATE TABLE**. Ogni tabella contiene tutti i vincoli di integrità necessari a garantirne il corretto funzionamento.

2.3.1 DROP per evitare conflitti

Per evitare possibili conflitti tra tabelle al momento della compilazione, o errori derivati da essi, sono stati inseriti tutti i drop necessari prima di iniziare con la creazione vera e propria delle tabelle, in modo tale da poter essere certi che la creazione delle tabelle vada a buon fine.

```
drop table PERSONALE cascade constraint;

drop table STIPENDIO;

drop table TURNI;

drop table CLIENTE_ISCRITTO cascade constraint;

drop table OFFERTE cascade constraint;

drop table MENU_STANDARD cascade constraint;

drop table MAGAZZINO cascade constraint;

drop table PRODOTTO cascade constraint;

drop table INFORMAZIONI_NUTRIZIONALI;

drop table ORDINE cascade constraint;

drop table FORNITORE cascade constraint;

drop table MERCE cascade constraint;

drop table ORDINE_RIFORNIMENTO cascade constraint;

drop table LOTTO cascade constraint;

drop table FATTURA;

    --tabelle di transizione--

drop table menu_contiene_prod;

drop table ordine_contiene_menu;

drop table ordine_contiene_prod;

drop table offerta_contiene_menu;

drop table offerta_contiene_prod;

drop table ordine_contiene_offerte;

drop table prodotto_composto_da;
```

```
drop table fornitore_fornisce_merce;
```

La prima macro sezione è quella che rappresenta la *gestione del personale*, in cui rientrano le seguenti entità.

2.3.2 PERSONALE

Questa tabella contiene tutte le informazioni relative al personale: i loro dati anagrafici, il ruolo che svolgono all'interno dell'azienda e soprattutto la loro tessera, che ha un valore univoco e ne permette l'identificazione.

Non sono presenti chiavi esterne, ma solo due check che vanno a controllare rispettivamente il ruolo che il personale può assumere, e il corretto formato del codice fiscale.

```
create table PERSONALE (  
    tessera          char(6) primary key,  
    nome             varchar(15) not null,  
    cognome          varchar(15) not null,  
    codice_fiscale   char(16) unique not null,  
    data_di_nascita  date not null,  
    ruolo            varchar(10) not null,  
    data_assunzione date not null,  
    --vincoli  
    constraint CHK_personale1 check (initcap(ruolo) in ←  
        ('Dipendente', 'Manager')),  
    constraint CHK_personale2 check ←  
        (regexp_like(codice_fiscale, ←  
            '[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]'))  
);
```

2.3.3 TURNI

Questa tabella contiene le informazioni relative ai turni di lavoro dell'intero personale. Grazie alle informazioni riguardanti l'ora di ingresso e uscita del personale, siamo in grado di risalire alla durata del turno di ogni dipendente. Sono presenti due attributi, cod_personale e data_turno, che insieme permettono di identificare un'istanza di questa tabella. Poiché TURNI è un'entità debole, uno dei due attributi, cod_personale, fa riferimento alla tessera del personale che ha effettuato quello specifico turno.

Questi due attributi insieme fanno sì che ogni membro del personale possa effettuare un solo turno in quel giorno.

```
create table TURNI (  
    cod_personale char(6) not null,  
    data_turno date not null,  
    ora_ingresso timestamp not null,  
    ora_uscita timestamp not null,  
    num_cassa char(2) not null,  
    --chiavi  
    constraint PK_turni primary key(cod_personale, data_turno),  
    constraint FK_turni foreign key (cod_personale) references ↵  
        PERSONALE(tessera) on delete cascade  
);
```

2.3.4 STIPENDIO

In questa tabella troviamo le informazioni riguardanti le buste paga dei dipendenti, identificate proprio tramite il codice della busta paga.

Lo stipendio viene percepito una volta al mese tranne che nel mese di dicembre, in cui è possibile ottenere la tredicesima.

```
create table STIPENDIO (  
    codice_busta_paga char(6) primary key,  
    stipendio number(6,2) not null,  
    codice_personale char(6) not null,  
    data_stipendio date not null,  
    --chiavi  
    constraint FK_stipendio foreign key (codice_personale) ↵  
        references PERSONALE(tessera) on delete cascade,  
    --vincoli  
    constraint CHK_stipendio check (stipendio >= 1200.00)  
);
```

Non sono presenti tabelle di transizione per rappresentare le relazioni tra le entità di questa sezione.

La seconda macro sezione è quella in cui si trovano tutte le entità e le relazioni tra esse che riguardano strettamente la *gestione degli ordini*.

2.3.5 ORDINE

L'entità principale, a cui sono legate direttamente o indirettamente le altre, è rappresentata da ORDINE.

Quest'entità, le cui istanze sono identificate tramite il numero dell'ordine, comprende anche un riferimento al dipendente che ha preso in carico quell'ordine, esclusi i manager in quanto non rientra nelle loro mansioni, e sull'eventuale cliente iscritto a cui è associato.

Contiene inoltre informazioni riguardanti il metodo di pagamento, riguardo al metodo con cui è stato effettuato l'ordine e riguardo al metodo per il ritiro, ognuno per i quali è stato definito un apposito controllo.

Possiamo notare che l'attributo cliente può assumere valore NULL, questo perché si vuole tenere traccia di tutti gli ordini effettuati, anche se non sono legati ad un cliente iscritto.

Tale attributo lega l'ordine al cliente iscritto, proprio perché rappresenta la chiave esterna di questa tabella, e fa riferimento all'email del cliente iscritto che lo identifica in maniera univoca all'interno della tabella CLIENTE_ISCRITTO. Dunque se l'ordine è effettuato da un cliente che non ha effettuato un'iscrizione al fast food, tale attributo assumerà valore nullo.

```
create table ORDINE (
    num_ordine      char(6) primary key,
    addetto         char(6) not null,
    cliente         varchar(25),
    metodo_pagamento varchar(16) not null,
    metodo_ordine   varchar(6) not null,
    metodo_ritiro   varchar(12) not null,
    data_ordine     date not null,
    ora_ordine      timestamp not null,
    --chiavi
    constraint FK_ordine1 foreign key (addetto) references ←
        PERSONALE(tessera) on delete cascade,
    constraint FK_ordine2 foreign key (cliente) references ←
        CLIENTE_ISCRITTO(e_mail) on delete cascade,
    --vincoli
    constraint CHK_ordine1 check (metodo_pagamento in ('Carta ←
        di Credito', 'Contanti')),
    constraint CHK_ordine2 check (metodo_ordine in ('drive', ←
        'cassa', 'totem'))),
```

```
constraint CHK_ordine3 check (metodo_ritiro in ('da ←  
    portare', 'da mangiare'))  
);
```

2.3.6 CLIENTE_ISCRITTO

Un entità allo stesso tempo importante, ma che, come detto, può non sempre essere presente in un ordine, è CLIENTE_ISCRITTO.

Questi vengono identificati tramite la loro mail e, per politiche aziendali, bisogna specificare che sono effettivamente gli unici clienti che hanno accesso alle offerte, proprio perché clienti iscritti, mentre dei semplici clienti non si tiene traccia di nessuna informazione se non dell'ordine effettuato.

```
create table CLIENTE_ISCRITTO (  
    e_mail          varchar(25) primary key,  
    password_cliente char(8) not null,  
    tessera_cliente char(6) unique not null  
);
```

2.3.7 PRODOTTO

Questa tabella si occupa di memorizzare tutte le informazioni necessarie per l'identificazione dei prodotti.

Ogni prodotto viene identificato tramite il suo codice a barre, e un check si occupa di controllare che la categoria del prodotto e la rispettiva grandezza rientrino tra i valori permessi.

```
create table PRODOTTO (  
    cod_barre_prodotto char(13) primary key,  
    nome_prodotto      varchar(25) not null,  
    prezzo_prodotto    number(4,2) not null,  
    grandezza_prod    varchar(10) not null,  
    categoria_prod     varchar(7) not null,  
    --vincoli  
    constraint CHK_prodotto1 check (categoria_prod in ←  
        ('panini', 'bibite', 'extra', 'dolci')),  
    constraint CHK_prodotto2 check (grandezza_prod in ←  
        ('piccolo', 'medio', 'grande', 'standard'))  
);
```

2.3.8 MENU_STANDARD

In questa tabella sono contenute le informazioni riguardanti i menù standard ideati dal fast food.

Ogni menù standard viene identificato tramite il suo codice, si specifica inoltre il tipo di menù, controllato tramite un check, e il suo prezzo, che essendo un prezzo fisso è indipendente dai singoli prezzi dei prodotti che sono contenuti all'interno del menù.

```
create table MENU_STANDARD (  
    codice_menu      char(4) primary key,  
    tipo_menu        varchar(15) not null,  
    prezzo_menu      number(4,2) not null,  
    --vincoli  
    constraint CHK_menustd1 check (tipo_menu in ('menu ←  
        famiglia', 'menu doppio', 'menu singolo'))  
);
```

2.3.9 OFFERTE

In questa tabella sono riportate le informazioni principali di un offerta, quindi il suo codice, da quando a quando ha validità e soprattutto il suo prezzo. Il prezzo di un offerta è un prezzo statico, completamente indipendente dal prezzo dei singoli prodotti o dei menù standard che contiene al suo interno, in quanto non è derivati da essi.

I prodotti e i menù standard quindi compongono un offerta ma non ne determinano direttamente il prezzo in alcun modo.

```
create table OFFERTE (  
    codice_offerta   char(5) primary key,  
    data_inizio      date not null,  
    data_fine        date not null,  
    prezzo_offerta   number(4,2) not null  
);
```

Vi sono poi diverse tabelle di transizione volte a rappresentare alcune associazioni tra le entità sopraelencate.

Relative all'ordine troviamo, ad esempio:

2.3.10 ordine_contiene_prod

Questa tabella indica quali prodotti e in quale quantità sono stati inseriti all'interno dell'ordine presi singolarmente.

Non ci sono restrizioni su quanti prodotti si possono ordinare all'interno dello stesso ordine.

Ci sono due attributi, `cod_ordine` e `prodotto`, che permettono di identificare le istanze in maniera univoca, entrambe sono anche dei riferimenti ad attributi esterni alla tabella in quanto una fa riferimento al numero dell'ordine e l'altro fa riferimento al prodotto ordinato.

```
create table ordine_contiene_prod (
  cod_ordine      char(6) not null,
  prodotto        char(13) not null,
  quantita_prodotto number(2) not null,
  --chiavi
  constraint PK_ord_con_prod primary key (cod_ordine, prodotto),
  constraint FK_ord_con_prod1 foreign key (cod_ordine) ↔
    references ORDINE(num_ordine) on delete cascade,
  constraint FK_ord_con_prod2 foreign key (prodotto) ↔
    references PRODOTTO(cod_barre_prodotto) on delete cascade
);
```

.

Relative alle offerte troviamo, invece, ad esempio:

2.3.11 offerta_contiene_prod

Questa tabella di transizione permette di sapere quali prodotti sono presenti in un'offerta.

Ci sono due attributi, `cod_offerta` e `prodotto`, che fanno da chiave primaria, ma entrambi sono anche chiavi esterne, che fanno riferimento rispettivamente al codice dell'offerta e al codice a barre del singolo prodotto.

Non ci sono restrizioni su quanti prodotti possano essere presenti all'interno dell'offerta.

```
create table offerta_contiene_prod (
```

```

cod_offerta      char(5) not null,
prodotto         char(13) not null,
quantita_prodotto number(1) not null,
--chiavi
constraint PK_offerta_cont_prod primary key (prodotto, ↵
cod_offerta),
constraint FK_offerta_cont_prod1 foreign key (prodotto) ↵
references PRODOTTO(cod_barre_prodotto) on delete cascade,
constraint FK_offerta_cont_prod2 foreign key (cod_offerta) ↵
references OFFERTE(codice_offerta) on delete cascade
);

```

Infine, l'ultima macro sezione del database è rappresentata dalla *gestione del magazzino*, di cui fanno parte le seguenti entità.

2.3.12 MAGAZZINO

Questa tabella fornisce le informazioni riguardanti la divisione in aree del magazzino, infatti le sue istanze vengono identificate proprio tramite il codice dell'area, e possiede altre informazioni sulla categoria di merce in quell'area e sul tipo di conservazione, che può essere frigorifero o scaffale, identificati anch'essi da due codici, rispettivamente 00 e 01.

I controlli relativi proprio alla categoria dell'area e al tipo di conservazione sono stati implementati tramite due check due check.

```

create table MAGAZZINO (
cod_area         char(3) primary key,
categoria_area   varchar(20) not null,
tipo_conservazione char(2) not null,
--vincoli
constraint CHK_magazzino1 check (categoria_area in ↵
('uova', 'frutta', 'bibite', 'carni', 'pane', 'ortaggi', ↵
'formaggi', 'verdure', 'dolci', 'latticini', 'prodotti ↵
da forno')),
constraint CHK_magazzino2 check (tipo_conservazione in ↵
('00', '01'))
);

```

2.3.13 ORDINE_RIFORNIMENTO

Questa tabella è di fondamentale importanza perché contiene informazioni riguardante l'ordine che rifornisce il magazzino quando la merce contenuta sta per finire.

Il codice dell'ordine rappresenta proprio la chiave con cui sono identificate le sue istanze.

Le chiavi esterne indirizzano rispettivamente all'area del magazzino all'interno della quale verranno poi posizionati i prodotti riforniti e alla partita iva del fornitore di quella merce.

È da tener conto che all'interno di un'ordine rifornimento possono essere presenti non uno solo ma anche più lotti, a patto che siano della categoria di merce e ovviamente che l'ordine sia fatto verso un singolo fornitore.

```
create table ORDINE_RIFORNIMENTO (  
    cod_ordine_rif      number(7) primary key,  
    codice_area_mag     char(3) not null,  
    num_tracking        number(12) unique not null,  
    data_rif            date not null,  
    rif_fornitore       char(11) not null,  
  
    --chiavi  
    constraint FK_ord_rif1 foreign key (codice_area_mag) ↔  
        references MAGAZZINO(cod_area) on delete cascade,  
    constraint FK_ord_rif2 foreign key (rif_fornitore) ↔  
        references FORNITORE(partita_iva) on delete cascade  
);
```

2.3.14 FATTURA

Tabella che tiene conto delle informazioni riguardanti le fatture rilasciate per ogni ordine rifornimento.

L'identificativo per le sue istanze è proprio il codice della fattura. Esiste una chiave esterna che fa riferimento ovviamente al codice dell'ordine rifornimento a lui deve essere legata la fattura, mentre un check si occupa di controllare la corretta immissione del metodo con cui viene pagato l'ordine di rifornimento.

```
create table FATTURA (  
    cod_fattura         char(14) primary key,  
    data_fattura        date not null,  
    ordine_rif          number(7) not null,  
    metodo_pag_rif      varchar(8) not null,  
  
    --chiavi  
    constraint FK_fattura foreign key (ordine_rif) references ↔  
        ORDINE_RIFORNIMENTO(cod_ordine_rif) on delete cascade,  
    --vincoli
```

```
constraint CHK_fattura check (metodo_pag_rif in ('carta', ←  
        'contanti'))  
);
```

2.3.15 FORNITORE

Questa tabella tiene traccia dei fornitori (si sono presi in considerazione solo fornitori italiani) che riforniscono il magazzino del fast food.

Ciò che identifica in maniera univoca i fornitori è la loro partita iva.

```
create table FORNITORE (  
    partita_iva          char(11) primary key,  
    e_mail_fornitore     varchar(36) not null,  
    ragione_sociale      varchar(30) not null,  
    num_telefono         char(13) not null,  
    via                  varchar(30) not null,  
    citta                varchar(25) not null,  
    CAP                  char(5) not null  
);
```

2.3.16 MERCE

Questa è un'altra tabella di particolare rilevanza all'interno del database, perché al suo interno vengono memorizzate informazioni riguardanti la merce che viene fornita dai fornitori.

Ricordiamo che per merce si vogliono intendere gli ingredienti presenti in ogni prodotto, quindi le singole componenti che andranno poi a formare un prodotto.

Si utilizza poi un check per far sì che la categoria della merce abbia effettivamente gli stessi valori che assume la merce presente in magazzino.

```
create table MERCE (  
    nome_merce           varchar(15) primary key,  
    categoria_merce      varchar(20) not null,  
  
    --vincoli  
    constraint CHK_merce1 check (categoria_merce in ('frutta', ←  
        'formaggi', 'bibite', 'pane', 'carne', 'pollo', ←  
        'verdure', 'ortaggi', 'dolce', 'latticini', 'uova', ←  
        'prodotti da forno'))  
);
```

2.3.17 LOTTO

All'interno di questa tabella vengono memorizzate le informazioni riguardanti i lotti di merce che vengono ordinati negli ordini rifornimento.

Ogni lotto ha una propria scadenza, di conseguenza anche nel caso che un'ordine rifornimento dovesse essere composto da più lotti della stessa merce questi risulteranno avere scadenze diverse.

Esistono due chiavi esterne che fanno riferimento rispettivamente al nome della merce contenuta nel lotto e al codice dell'ordine rifornimento, in modo tale da poter vedere quale merce è contenuta all'interno del lotto e in quale ordine è stato acquistato.

```
create table LOTTO (
    cod_lotto      char(5) primary key,
    merce_contenuta varchar(15) not null,
    quantita_merce number(3) not null,
    scadenza_lotto date not null,
    unita_di_misura varchar(10) not null,
    ordine_rif     number(7) not null,

    --chiavi
    constraint FK_lotto1 foreign key (merce_contenuta) ↔
        references MERCE(nome_merce) on delete cascade,
    constraint FK_lotto2 foreign key (ordine_rif) references ↔
        ORDINE_RIFORNIMENTO(cod_ordine_rif) on delete cascade
);
```

2.4 Data Manipulation Language - DML

Il popolamento delle tabelle sopra create avviene attraverso operazioni messe a disposizione dal linguaggio SQL, cioè i comandi INSERT e UPDATE, ma in questo caso sono utilizzati solo INSERT in quanto si fa riferimento al primo inserimento nelle tabelle sopra create.

Verrà mostrato solo un'esempio di inserimento per ogni tipo di tabella creata:

```
insert into PERSONALE values ('000001', 'Mario', 'Rossi', ↔
    'RSSMRA84P17F839U', TO_DATE('17-09-1984', 'DD-MM-YYYY'), ↔
    'Manager', TO_DATE('15-05-2015', 'DD-MM-YYYY'));

insert into STIPENDIO values ('0A24Z4', 1600.00, '000001', ↔
    TO_DATE('02-08-2022', 'DD-MM-YYYY'));
```



```

insert into TURNI values ('000001', TO_DATE('02-08-2022', ←
'DD-MM-YYYY'), TO_TIMESTAMP('02-08-2022 09:00:00', ←
'DD-MM-YYYY HH24:MI:SS'), TO_TIMESTAMP('02-08-2022 ←
17:00:00', 'DD-MM-YYYY HH24:MI:SS'), 00);

insert into CLIENTE_ISCRITTO values('mariabianchi@gmail.com', ←
'bianchi0', '255144');

insert into OFFERTE values('G2501', TO_DATE('01-08-2022', ←
'DD-MM-YYYY'), TO_DATE('01-09-2022', 'DD-MM-YYYY'), 7.99 );

insert into ORDINE values ('111111', '000002', ←
'mariabianchi@gmail.com', 'Carta di Credito', 'cassa', 'da ←
mangiare', TO_DATE('10-08-2022', 'DD-MM-YYYY'), ←
TO_TIMESTAMP('10-08-2022 15:34', 'DD-MM-YYYY HH24:MI'));

insert into MENU_STANDARD values('7654', 'menu famiglia', 14.99);

insert into MAGAZZINO values('001', 'uova', '00');

insert into PRODOTTO values('8032089000017', 'bigmac', 6.50, ←
'standard', 'panini' );

insert into INFORMAZIONI_NUTRIZIONALI values('8032089000017', ←
'carboidrati', 19.00, 42.00, '16%');

insert into FORNITORE values('00709150155', ←
'ottolinaregione@gmail.com', 'Ottolina', '+393885647275', ←
'Via Decemviri', 'Milano', '20137');

insert into MERCE values('Mele', 'frutta');

insert into ORDINE_RIFORMIMENTO values(0000001, '011', ←
'000000000001', TO_DATE('02-09-2022', 'DD-MM-YYYY'), ←
'00124290214');

insert into LOTTO values('L0109', 'Mele', 100, ←
TO_DATE('15-10-2022', 'DD-MM-YYYY'), '25kg', 0000001);

insert into FATTURA values('Mc-Dns-2022-01', ←
TO_DATE('04-09-2022', 'DD-MM-YYYY'), 0000001, 'carta');

insert into menu_contiene_prod values('7654', '8032089000028', 1);

insert into ordine_contiene_menu values('111111', '7654', 1);

```

```

insert into ordine_contiene_prod values('111126', ←
    '8032089000017', 4);

insert into offerta_contiene_menu values('G2501', '7654', 1);

insert into offerta_contiene_prod values('G2502', ←
    '8032089000017', 2);

insert into ordine_contiene_offerte values('111111', 'G2501');

insert into prodotto_composto_da values('Bacon', ←
    '8032089000017', 3);

insert into fornitore_fornisce_merce values ('Mele', 1.70, ←
    '00124290214', TO_DATE('07-09-2022', 'DD-MM-YYYY'));

```

2.5 Trigger

I diversi vincoli aziendali e i controlli in fase di inserimento, aggiornamento, eliminazione dei dati sono stati implementati tramite trigger DML.

Nel dettaglio il database presenta i seguenti trigger:

2.5.1 check_turno

Questo trigger controlla che un dipendente non vada ad effettuare un turno di lavoro che vada oltre le 8 ore.

```

CREATE OR REPLACE TRIGGER Check_turno
BEFORE INSERT ON TURNI
FOR EACH ROW
DECLARE
    durata_turno NUMBER := (cast(:new.ora_uscita as date)- ←
        cast(:new.ora_ingresso as date)) *24 *60;
    check_turno EXCEPTION;
BEGIN
    IF durata_turno > 480
        THEN RAISE check_turno;
    END IF;
EXCEPTION
    WHEN check_turno

```

```

    THEN RAISE_APPLICATION_ERROR(-20002, 'Massimo ore di lavoro superate');
END;

```

2.5.2 check_stipendio

Il trigger `check_stipendio` definisce per ciascuna mansione del personale il suo salario minimo, per politiche aziendali si è deciso che lo stipendio base di un dipendente è di 1200 euro, mentre per il manager è di 1600 euro. Se viene inserito uno stipendio errato verrà visualizzato un messaggio di errore.

```

CREATE OR REPLACE TRIGGER check_stipendio
BEFORE INSERT ON STIPENDIO
FOR EACH ROW
DECLARE
    ruoloim          VARCHAR(10);
    stip_base        NUMBER(6,2) := (:new.stipendio);
    check_stipendio  EXCEPTION;
BEGIN
    SELECT ruolo INTO ruoloim
    FROM personale PR
    WHERE PR.tessera = :new.codice_personale;
    IF stip_base < 1200.00 AND ruoloim = 'Dipendente'
        THEN RAISE check_stipendio;
    ELSIF stip_base < 1600.00 AND ruoloim = 'Manager'
        THEN RAISE check_stipendio;
    END IF;
EXCEPTION
    WHEN check_stipendio
    THEN RAISE_APPLICATION_ERROR(-20003, 'Stipendio troppo basso per questo ruolo');
END;

```

2.5.3 check_menu

Questo trigger controlla che per ogni istanza di `ordine_contiene_menu`, cioè la tabella di transizione che permette di visualizzare quali e quanti menu ci sono all'interno di un singolo ordine, il numero dei menù standard da poter ordinare non superi un massimo di 10.

```

CREATE OR REPLACE TRIGGER Check_menu
BEFORE INSERT ON Ordine_contiene_menu

```

```

FOR EACH ROW
DECLARE
    check_ordine    EXCEPTION;
BEGIN
    IF :new.quantita_menu_std > 10
        THEN RAISE check_ordine;
    END IF;
EXCEPTION
    WHEN check_ordine
        THEN RAISE_APPLICATION_ERROR(-20001, 'Numero massimo di ↵
            menu' raggiunto');
END;

```

2.5.4 Accesso_offerta

Questo trigger vieta ad un cliente non iscritto al fast food di effettuare un ordine contenente un' offerta, in quanto solo i clienti iscritti possono accedere alle diverse offerte. In caso contrario, viene visualizzato un messaggio di errore.

Il controllo viene effettuato prima di ogni inserimento di un'offerta nell'ordine, e per fare ciò, il trigger opera sulla tabella di transizione tra le due entità.

```

CREATE OR REPLACE TRIGGER Accesso_offerta
BEFORE INSERT ON ordine_contiene_offerte
FOR EACH ROW
DECLARE
    email            VARCHAR(25);
    offerte          CHAR(5);
    check_accesso    EXCEPTION;
BEGIN
    SELECT cliente, codice_offerta INTO email, offerte
    FROM ordine NATURAL JOIN offerte
    WHERE num_ordine = :new.ordine AND codice_offerta = ↵
        :new.offerta;

    IF email IS NULL AND offerte IS NOT NULL
        THEN RAISE check_accesso;
    END IF;
EXCEPTION
    WHEN check_accesso
        THEN RAISE_APPLICATION_ERROR(-20004, 'Il cliente non e' ↵
            iscritto');
END;

```

2.5.5 Check_Magazzino

Questo è l'unico trigger riguardante la gestione del magazzino.

Si previene l'inserito in magazzino di un lotto di merci scaduto, in caso affermativo viene visualizzato messaggio di errore.

Il controllo viene effettuato prima dell'inserimento di una tupla in LOTTO.

```
CREATE OR REPLACE TRIGGER Check_Magazzino
BEFORE INSERT ON Lotto
FOR EACH ROW
DECLARE
    check_scadenza EXCEPTION;
BEGIN
    IF :new.scadenza_lotto < SYSDATE
        THEN RAISE check_scadenza;
    END IF;
EXCEPTION
    WHEN check_scadenza
        THEN RAISE_APPLICATION_ERROR(-20005, 'Prodotto inserito ←
        scaduto');
END;
```

2.5.6 Check_offerta_valida

Questo trigger controlla, prima dell'inserimento di un'offerta sulla tabella di transizione ordine_contiene_offerte, se questa è ancora valida.

Si controlla quindi se la data al momento dell'ordine è compresa tra la data di inizio offerta e la data di fine offerta.

```
CREATE OR REPLACE TRIGGER Check_offerta_valida
BEFORE INSERT ON ordine_contiene_offerte
FOR EACH ROW
DECLARE
    dataord      DATE;
    inizio       DATE;
    fine         DATE;
    check_offerta EXCEPTION;
    contatore    NUMBER := 0;

BEGIN

    SELECT count(*) INTO contatore
    FROM ordine
    WHERE num_ordine = :new.ordine;
```

```

IF (contatore > 0) THEN
    SELECT data_ordine, data_inizio, data_fine INTO dataord, ↵
        inizio, fine
    FROM ordine NATURAL JOIN offerte
    WHERE num_ordine = :new.ordine AND codice_offerta = ↵
        :new.offerta;

    IF dataord NOT BETWEEN inizio AND fine
    THEN RAISE check_offerta;
END IF;

EXCEPTION
    WHEN check_offerta
    THEN RAISE_APPLICATION_ERROR(-20006, 'Offerta ↵
        scaduta o non disponibile');
END;

```

2.5.7 Check_doppio_stipendio

Questo è un trigger riguardante il controllo sull'inserimento degli stipendi del personale.

Si controlla che nello stesso mese dello stesso anno, non possano essere inseriti per lo stesso dipendente o manager 2 stipendi fatta eccezione per il mese di Dicembre. Questo è stato fatto al fine di poter assegnare la tredicesima mensilità, la quale viene considerata come doppio stipendio.

```

CREATE OR REPLACE TRIGGER Check_doppio_stipendio
BEFORE INSERT ON Stipendio
FOR EACH ROW
DECLARE
    ggstipendio    DATE := (:new.data_stipendio);
    contatore      NUMBER;
    check_doppio   EXCEPTION;
    tredicesima_ricevuta EXCEPTION;
BEGIN
    SELECT count(*) INTO contatore
    FROM stipendio
    WHERE codice_personale = :new.codice_personale AND ↵
        to_char(ggstipendio, 'MM-YYYY') = ↵
        to_char(data_stipendio, 'MM-YYYY');

    IF contatore >= 2 and to_char(ggstipendio, 'MM') = '12'
    THEN RAISE tredicesima_ricevuta;

```

```

        ELSIF contatore = 1 and to_char(ggstipendio, 'MM') <> '12'
            THEN RAISE check_doppio;
        END IF;

    EXCEPTION
        WHEN check_doppio
            THEN RAISE_APPLICATION_ERROR(-20007, 'Stipendio gia' ←
                inserito per questo mese');
        WHEN tredicesima_ricevuta
            THEN RAISE_APPLICATION_ERROR(-20008, '----');
    END;

```

2.5.8 Check_eta

Semplice trigger che controlla se l'età del dipendente che si sta assumendo è appunto idonea per l'assunzione, che per regole aziendali è compresa tra i 18 anni e i 50 anni.

In caso contrario viene stampato messaggio di errore.

Il controllo viene effettuato prima dell'inserimento di una tupla in PERSONALE.

```

CREATE OR REPLACE TRIGGER Check_eta
BEFORE INSERT ON Personale
FOR EACH ROW
DECLARE
    eta NUMBER := FLOOR((SYSDATE - :new.data_di_nascita)/365);
    check_eta EXCEPTION;
BEGIN
    IF eta NOT BETWEEN 18 AND 50
        THEN RAISE check_eta;
    END IF;

    EXCEPTION
        WHEN check_eta
            THEN RAISE_APPLICATION_ERROR(-20008, 'Eta' del dipendete ←
                non rientra nei criteri di assunzione');
END;

```

2.5.9 Check_Max

Trigger utilizzato affinché non possano essere assunti più di un massimo numero di dipendenti (≥ 30). Il controllo viene effettuato prima dell'inse-

rimento di una tupla in Personale.

```
CREATE OR REPLACE TRIGGER Check_Max
BEFORE INSERT ON Personale
FOR EACH ROW
DECLARE
    num_pers NUMBER;
    check_max EXCEPTION;
BEGIN
    SELECT count(*) INTO num_pers
    FROM Personale;
    IF num_pers >= 30
        THEN RAISE check_max;
    END IF;
EXCEPTION
    WHEN check_max
        THEN RAISE_APPLICATION_ERROR(-20009, 'Numero massimo di ↵
        Personale raggiunto');
END;
```

2.5.10 Check_Ultimo_Turno

Ulteriore trigger riguardante il controllo dei turni del personale.

Viene visualizzato un messaggio di errore nel momento in cui si prova ad inserire per un dipendente un nuovo turno se il suo precedente non è ancora terminato.

C'è da chiarire che questo trigger vale solo per i turni che iniziano e terminano dopo la mezzanotte. Se infatti assumiamo che un dipendente abbia un turno che inizia alle 8pm del giorno 14/09 e termini alle ore 4am del giorno 15/09, non potrà essere inserito per il dipendente un turno che inizi il 15/09 alle ore 2am poiché non si è ancora concluso. Tale eccezione nasce dal fatto che la chiave dell'entità TURNI è data dalla coppia (cod_personale, data_turno), per cui è un controllo effettuato implicitamente.

```
CREATE OR REPLACE TRIGGER Check_Ultimo_Turno
BEFORE INSERT ON Turni
FOR EACH ROW
DECLARE
    fine_turno      turni.ora_uscita%Type;
    check_ultimo    EXCEPTION;
BEGIN
    SELECT max(ora_uscita) INTO fine_turno
    FROM turni
    WHERE cod_personale = :new.cod_personale;
```



```

        IF fine_turno > :new.ora_ingresso
            THEN RAISE check_ultimo;
        END IF;
    EXCEPTION
        WHEN check_ultimo
            THEN RAISE_APPLICATION_ERROR(-20010, 'Turno precedente ←
                non ancora concluso');
    END;

```

2.5.11 Check_cassa

Questo trigger controlla che la cassa inserita inerente al dipendente sia corretta, poiché per politiche aziendali si è deciso che la cassa '00' è esclusivamente riservata al manager proprio perché è una cassa fittizia, utilizzata per evitare di avere valori NULL all'interno della tabella.

Se accade il contrario viene mostrato un messaggio di errore.

Il controllo viene effettuato prima dell'inserimento di una tupla in TURNI.

```

CREATE OR REPLACE TRIGGER Check_cassa
BEFORE INSERT ON Turni
FOR EACH ROW
DECLARE
    ruoloim          varchar(10);
    cassa            char(2) := (:new.num_cassa);
    check_cassa      EXCEPTION;
BEGIN
    SELECT ruolo INTO ruoloim
    FROM personale PR
    where PR.tessera = :new.cod_personale;

    IF cassa <> '0' and ruoloim = 'Manager'
        THEN RAISE check_cassa;
    ELSIF cassa = '0' and ruoloim <> 'Manager'
        THEN RAISE check_cassa;
    END IF;
EXCEPTION
    WHEN check_cassa
        THEN RAISE_APPLICATION_ERROR(-20011, 'ERRORE, NUMERO ←
            CASSA SBAGLIATO');
END;

```

2.6 Procedure

Le operazioni che possono essere effettuate dagli utenti sono state implementate tramite procedure per cercare di gestire la base dati nella maniera più efficiente possibile.

2.6.1 BonusCassiere

Questa procedura serve ad automatizzare il processo di attribuzione di un bonus del 20% allo stipendio attuale del dipendente (manager esclusi) che nel mese precedente ha effettuato più ordini.

Per scelte aziendali, nel momento in cui si dovesse verificare una parità di ordini serviti, si premia il dipendente che ha effettuato più ore lavorative. Ricordiamo che si seleziona il dipendente che è stato già stipendiato nel mese precedente (poiché non può essere inserito uno stipendio nel mese attuale se in precedenza non è stato stipendiato). Dunque i dipendenti che sono stati assunti nel mese corrente non possono avere bonus (solo dal mese successivo). Al fine di ottenere il mese precedente/successivo, utilizziamo la funzione `ADD_MONTHS`, che ha come parametri una data e il numero di mesi che vogliamo aggiungere ad essa. Nello specifico, per ottenere il mese precedente passiamo come secondo parametro il valore `-1`, mentre per ottenere quello successivo utilizziamo come secondo parametro `1`. Il formato della data utilizzato in questa procedura durante la conversione a `CHAR` è `'MM-YYYY'`, questo perché non è importante sapere il giorno e il minutaggio.

N.B. non ho bisogno di selezionare esplicitamente i dipendenti poiché gli ordini sono effettuati esclusivamente da essi

```
CREATE OR REPLACE PROCEDURE BonusCassiere
IS
    Cassiere CHAR(6);

BEGIN
    FOR i IN (
        SELECT sum(((cast(ora_uscita as date) - cast(ora_ingresso as date)) * 24)), cod_personale
        FROM turni
        WHERE cod_personale IN (SELECT addetto
                                FROM ordine ord JOIN personale pr ON ord.addetto = pr.tessera JOIN stipendio ST ON st.codice_personale = pr.tessera
                                WHERE to_char(ADD_MONTHS(SYSDATE, -1), 'MM-YYYY') = to_char(data_ordine, 'MM-YYYY'))
    )
```

```

        'MM-YYYY') AND <↵
        to_char(ADD_MONTHS(SYSDATE, -1), <↵
        'MM-YYYY') = <↵
        to_char(data_stipendio, 'MM-YYYY')
    GROUP BY addetto
    HAVING count(*) = (SELECT max(count(*))
                        FROM ordine ord JOIN <↵
                        personale pr ON <↵
                        ord.addetto = <↵
                        pr.tessera JOIN <↵
                        stipendio ST ON <↵
                        st.codice_personale <↵
                        = pr.tessera
                        WHERE <↵
                        to_char(ADD_MONTHS(SYSDATE, <↵
                        -1), 'MM-YYYY') = <↵
                        to_char(data_ordine, <↵
                        'MM-YYYY') AND <↵
                        to_char(ADD_MONTHS(SYSDATE, <↵
                        -1), 'MM-YYYY') = <↵
                        to_char(data_stipendio, <↵
                        'MM-YYYY')
                        GROUP BY addetto
                    ))
    GROUP BY cod_personale
)
LOOP

    SELECT cod_personale INTO Cassiere
    FROM turni
    WHERE cod_personale = i.cod_personale
    GROUP BY cod_personale
    HAVING sum((((cast(ora_uscita as date) - cast(ora_ingresso <↵
        as date)) * 24)) IN ( SELECT max(sum((((cast(ora_uscita <↵
        as date) - cast(ora_ingresso as date)) * 24)))
                                FROM turni
                                WHERE cod_personale <↵
                                    = i.cod_personale
                                GROUP BY cod_personale
                            );

END LOOP;

UPDATE Stipendio SET Stipendio = ( Stipendio + (Stipendio * <↵
    0.2)) Where codice_personale = cassiere;

```

```

UPDATE Stipendio SET data_stipendio = SYSDATE Where ↵
    codice_personale = cassiere;

COMMIT;
DBMS_OUTPUT.PUT_LINE('Bonus del cassiere ' ||cassiere|| ' ↵
    inserito con successo allo stipendio');

END;

```

2.6.2 OffertaProdotto

La procedura OffertaProdotto si occupa di trovare il menù meno venduto ed applicare a quello con l'offerta più costosa, un'ulteriore sconto del 50%. La nuova offerta che verrà a crearsi sarà valida a partire dalla data attuale fino allo stesso giorno del mese successivo (esattamente un mese di validità). Verrà visualizzato un messaggio di errore nel momento in cui non viene venduto nessun menù.

```

CREATE OR REPLACE PROCEDURE OffertaProdotto
IS
    menu    CHAR(4);
    offerta CHAR(5);
BEGIN
    SELECT cod_menu INTO menu
    FROM ordine_contiene_menu
    GROUP BY cod_menu
    HAVING count(*) = (SELECT min(count(*))
                       FROM ordine_contiene_menu
                       GROUP BY cod_menu
                      )
    FETCH FIRST 1 ROW ONLY;

    SELECT codice_offerta INTO offerta
    FROM Offerte offr JOIN offerta_contiene_menu trans ON ↵
        trans.cod_offerta = offr.codice_offerta JOIN ↵
        menu_standard menu ON trans.cod_menu_std = menu.codice_menu
    WHERE codice_menu = menu AND
        prezzo_offerta = ( SELECT max(prezzo_offerta)
                           FROM Offerte offr JOIN ↵
                               offerta_contiene_menu trans ON ↵
                               trans.cod_offerta = ↵
                               offr.codice_offerta JOIN ↵

```

```

                                menu_standard menu ON ↔
                                trans.cod_menu_std = menu.codice_menu
                                WHERE codice_menu = menu
                                );

UPDATE offerte SET prezzo_offerta = prezzo_offerta * 0.5 ↔
      where codice_offerta = offerta;
COMMIT;

UPDATE offerte SET data_inizio = SYSDATE where ↔
      codice_offerta = offerta;
COMMIT;

UPDATE offerte SET data_fine = ADD_MONTHS(SYSDATE,1) where ↔
      codice_offerta = offerta;
COMMIT;

DBMS_OUTPUT.PUT_LINE('e' stata applicata al menu ' ||menu|| ↔
      ' uno sconto del 50% sull offerta ' ||offerta);

EXCEPTION
  WHEN NO_DATA_FOUND
  THEN RAISE_APPLICATION_ERROR(-20001, 'Non e' stato ↔
      venduto nessun menu');

END;
```

2.6.3 Promozione

La procedura Promozione automatizza il processo di promozione a Manager del dipendente che ha venduto un numero maggiore di menu con il prezzo più alto e che è stato assunto prima.

Poiché il numero massimo di manager per la nostra azienda è 3, se questo massimo è stato raggiunto, si può effettuare un'eccezione. Si verifica che, tra i manager già presenti, vi sia uno che abbia effettuato un numero di ore minore rispetto al prossimo candidato a manager, questa condizione è verificata solo se il candidato è stato assunto prima di tale manager.

```

CREATE OR REPLACE PROCEDURE Promozione
IS
  cassiere CHAR(6);
  dataM    DATE;
  dataC    DATE;
  oreM     NUMBER;
```

```

oreC      NUMBER;
contatore NUMBER := 0;

BEGIN

    --cerco il cassiere che ha venduto un numero maggiore del ←
    --prodotto piu' caro e selezionare in caso di un risultato ←
    --multiplo quello assunto piu' lontano nel tempo--
    SELECT addetto INTO cassiere
    FROM ordine ord JOIN ordine_contiene_menu or_con_mstd ON ←
        ord.num_ordine = or_con_mstd.cod_ordine
        JOIN menu_standard mstd ON ←
            or_con_mstd.cod_menu = mstd.codice_menu
        JOIN personale pr ON pr.tessera = ord.addetto
    GROUP BY addetto, data_assunzione
    HAVING sum(quantita_menu_std) = ( SELECT ←
        max(sum(quantita_menu_std))
        FROM ordine ord JOIN ←
            ordine_contiene_menu ←
            or_con_mstd ON ←
            ord.num_ordine = ←
            or_con_mstd.cod_ordine
            JOIN ←
                menu_standard ←
                mstd ON ←
                or_con_mstd.cod_menu ←
                = ←
                mstd.codice_menu
        WHERE prezzo_menu = (SELECT ←
            max(prezzo_menu)
            FROM ←
                menu_standard
            )
        GROUP BY addetto
    )

    ORDER BY data_assunzione
    FETCH FIRST 1 ROW ONLY;

    SELECT count(*) INTO contatore
    FROM personale
    WHERE ruolo = 'Manager';

    IF contatore >= 3 THEN

```

```

--seleziono la data di assunzione e le ore effettuate ←
  del manager che presenza un minore ore di lavoro ←
  rispetto agli altri gia' presenti--
SELECT data_assunzione, sum(((cast(ora_uscita as date) - ←
  cast(ora_ingresso as date)) * 24)) INTO dataM, oreM
FROM personale pr join turni tu on pr.tessera = ←
  tu.cod_personale
WHERE ruolo = 'Manager'
GROUP BY data_assunzione
HAVING sum(((cast(ora_uscita as date) - ←
  cast(ora_ingresso as date)) * 24)) =
  ( SELECT min(sum(((cast(ora_uscita as date) - ←
    cast(ora_ingresso as date)) * 24)))
    FROM personale pr join turni tu on pr.tessera = ←
      tu.cod_personale
    WHERE ruolo = 'Manager'
    GROUP BY cod_personale
  );

--seleziono la data di assunzione e le ore di lavoro del ←
  nostro candidato a manager--
SELECT sum(((cast(ora_uscita as date) - ←
  cast(ora_ingresso as date)) * 24)), data_assunzione ←
  INTO oreC, dataC
FROM personale pr JOIN turni tu ON pr.tessera = ←
  tu.cod_personale
WHERE tessera = cassiere
GROUP BY tessera, data_assunzione;

IF oreC > oreM AND dataC < dataM THEN
  UPDATE personale SET ruolo = 'Manager' WHERE tessera ←
    = cassiere;
  DBMS_OUTPUT.PUT_LINE('Il dipendete: ' ||cassiere|| ' ←
    e' stato promosso a MANAGER');
ELSE
  DBMS_OUTPUT.PUT_LINE('Il dipendete non rispecchia i ←
    criteri');
END IF;
ELSE
  UPDATE personale SET ruolo = 'Manager' WHERE tessera = ←
    cassiere;
  DBMS_OUTPUT.PUT_LINE('Il dipendete: ' ||cassiere|| ' e' ←
    stato promosso a MANAGER');

```

```

    END IF;

END;

```

2.6.4 totale_ordine

Questa procedura si occupa di calcolare il prezzo totale di un ordine. Per fare ciò calcola separatamente i prezzi totali dei prodotti singoli, dei menù standard e delle offerte che un ordine contiene, ponendo come zero i valori nel caso dovessero essere NULL in quanto in un ordine è possibile avere prodotti, menù standard e offerte in maniera separata o avere combinazioni di essi.

Successivamente i valori trovati vengono sommati tra loro in modo tale da avere il prezzo effettivo totale dell'ordine.

```

CREATE OR REPLACE PROCEDURE totale_ordine (num_ordine varchar)
IS
    ordine_preso VARCHAR(10) := num_ordine;
    tot_prodotti NUMBER;
    tot_mstd     NUMBER;
    tot_offerte  NUMBER;
    tot_ordine   NUMBER;

BEGIN
    --trovare il prezzo totale dei prodotti in ordine
    SELECT sum(pr.prezzo_prodotto * ←
              or_con_pr.quantita_prodotto) INTO tot_prodotti
    FROM ordine ord JOIN ordine_contiene_prod or_con_pr ON ←
          ord.num_ordine = or_con_pr.cod_ordine
              JOIN prodotto pr ON or_con_pr.prodotto = ←
              pr.cod_barre_prodotto
    WHERE ord.num_ordine = ordine_preso;

    --trovare il prezzo totale dei menu standard in ordine
    SELECT sum(mstd.prezzo_menu * ←
              or_con_mstd.quantita_menu_std) INTO tot_mstd
    FROM ordine ord JOIN ordine_contiene_menu or_con_mstd ON ←
          ord.num_ordine = or_con_mstd.cod_ordine
              JOIN menu_standard mstd ON ←
              or_con_mstd.cod_menu = mstd.codice_menu
    WHERE ord.num_ordine = ordine_preso;

    --trovare il prezzo totale delle offerte in ordine
    SELECT sum(oft.prezzo_offerta) INTO tot_offerte

```



```

FROM ordine ord JOIN ordine_contiene_offerte or_con_ofi ON ↔
ord.num_ordine = or_con_ofi.ordine
JOIN offerte ofi ON or_con_ofi.offerta = ↔
ofi.codice_offerta
WHERE ord.num_ordine = ordine_preso;

--trovare il prezzo totale dell'ordine
IF (tot_prodotti IS NULL)
    THEN tot_prodotti := 0;
END IF;

IF (tot_mstd IS NULL)
    THEN tot_prodotti := 0;
END IF;

IF (tot_offerte IS NULL)
    THEN tot_prodotti := 0;
END IF;

tot_ordine := tot_prodotti + tot_mstd + tot_offerte;

DBMS_OUTPUT.PUT_LINE ('Il prezzo totale dei prodotti in ↔
ordine ammonta a : ' || tot_prodotti);
DBMS_OUTPUT.PUT_LINE('Il prezzo totale dei menu standard in ↔
ordine ammonta a : ' || tot_mstd);
DBMS_OUTPUT.PUT_LINE ('Il prezzo totale delle offerte in ↔
questo ordine ammonta a : ' || tot_offerte);
DBMS_OUTPUT.PUT_LINE('Il prezzo totale di questo ordine ↔
ammonta a : ' || tot_ordine);

END;
```

2.6.5 trova_merce

Questa procedura non è altro che un cursore che scorre l'entità MERCI visualizzando, per ogni valore di *nome_merce*, la sua giacenza tramite la procedura del calcolo della giacenza.

```

CREATE OR REPLACE PROCEDURE trova_merce
IS
    merce_trovata VARCHAR(20);

    CURSOR merce IS
```

```

        SELECT nome_merce
        FROM merce;

BEGIN
    OPEN merce;
    LOOP
        FETCH merce INTO merce_trovata;
        DBMS_OUTPUT.PUT_LINE('Merce trovata : ' || merce_trovata);
        calcolo_giacenza(merce_trovata);
        exit WHEN merce%notfound;
    END LOOP;
    CLOSE merce;

END;
```

2.6.6 crea_ordine

Questa procedura si occupa di creare automaticamente un nuovo ordine nel caso in cui le scorte di quella determinata merce, trovata utilizzando *trova_merce*, siano inferiori a 10.

Prima di effettuare un ordine però si assicura di trovare quale sia il fornitore che vende quella merce al prezzo più basso e alla data più recente possibile. Per poter generare un nuovo ordine si scelgono come codice e un numero tracking identificativo, i successivi valori di essi dell'ultimo ordine rifornimento, al fine di automatizzare la procedura.

```

CREATE OR REPLACE PROCEDURE crea_ordine (merce varchar)
IS
    merce_da_ordinare VARCHAR(20) := merce;
    fornitore_trovato CHAR(11);
    cod_area_mag_prec CHAR(3);
    ord_rif_prec      NUMBER(7);
    nuovo_ordine_rif  NUMBER(7);
    num_track_prec    NUMBER(12);
    nuovo_num_tracking NUMBER(12);

BEGIN
    -- trovare il fornitore che vende quella merce al prezzo piu' ←
    -- basso

    SELECT partita_iva INTO fornitore_trovato
    FROM fornitore fo JOIN fornitore_fornisce_merce f_m ON ←
        fo.partita_iva = f_m.fornitore
        JOIN merce me ON f_m.merce = me.nome_merce
```

```

WHERE me.nome_merce = merce_da_ordinare AND
prezzo_unitario = ( SELECT min(prezzo_unitario)
                    FROM fornitore fo JOIN ↵
                        fornitore_fornisce_merce f_m ON
                        fo.partita_iva = f_m.fornitore ↵
                        JOIN merce me ON f_m.merce = ↵
                        me.nome_merce
                    WHERE me.nome_merce = ↵
                        merce_da_ordinare AND
                    data_prezzo = ( SELECT data_prezzo
                                    FROM fornitore fo ↵
                                    JOIN ↵
                                    fornitore_fornisce_merce ↵
                                    f_m ON ↵
                                    fo.partita_iva = ↵
                                    f_m.fornitore
                                    JOIN merce me ON ↵
                                    f_m.merce = ↵
                                    me.nome_merce
                                    WHERE me.nome_merce ↵
                                        = merce_da_ordinare
                                    ORDER BY ↵
                                        f_m.data_prezzo ↵
                                        DESC
                                    FETCH FIRST 1 ROW only
                                    )
                    GROUP BY nome_merce
                );

DBMS_OUTPUT.PUT_LINE ('Il fornitore che vende questa merce ↵
                        al prezzo piu' basso e' : ' || fornitore_trovato);

--selezionare la chiave dell'ultimo ordine rifornimento, ↵
--servira' perche' la chiave del nuovo ordine rifornimento ↵
--sara' questo trovato + 1
--allo stesso modo facciamo con il numero tracking

SELECT cod_ordine_rif, num_tracking INTO ord_rif_prec, ↵
       num_track_prec
FROM ordine_rifornimento
ORDER BY data_rif, num_tracking DESC
FETCH FIRST 1 ROW only;

```

```

DBMS_OUTPUT.PUT_LINE ('Il precedente ordine rifornimento ←
    aveva codice : ' || ord_rif_prec);
DBMS_OUTPUT.PUT_LINE ('Il precedente num_tracking era : ' ←
    || num_track_prec);

--nuovo codice ordine rifornimento
nuovo_ordine_rif := ord_rif_prec + 1;
DBMS_OUTPUT.PUT_LINE ('Il nuovo cod_ordine_rif sara' : ' || ←
    nuovo_ordine_rif);

--nuovo numero tracking
nuovo_num_tracking := num_track_prec + 1;
DBMS_OUTPUT.PUT_LINE ('Il nuovo num_tracking sara' : ' || ←
    nuovo_num_tracking);

--selezionare il codice_area_mag legato all'ultimo ordine fatto ←
di quella merce
SELECT ord_rif.codice_area_mag INTO cod_area_mag_prec
FROM ordine_rifornimento ord_rif JOIN lotto lo ON ←
    ord_rif.cod_ordine_rif = lo.ordine_rif
    JOIN merce me ON ←
        lo.merce_contenuta = ←
        me.nome_merce
WHERE me.nome_merce = merce_da_ordinare
ORDER BY ord_rif.data_rif DESC
FETCH FIRST 1 ROW only;

DBMS_OUTPUT.PUT_LINE ('Il cod_area_mag di questa merce e' : ←
    ' || cod_area_mag_prec);

--inserire in ordine_rifornimento un nuovo ordine presso quel ←
fornitore trovato
INSERT INTO ordine_rifornimento VALUES (nuovo_ordine_rif, ←
    cod_area_mag_prec, nuovo_num_tracking, SYSDATE, ←
    fornitore_trovato);

DBMS_OUTPUT.PUT_LINE('Un nuovo ordine verso il fornitore ' ←
    || fornitore_trovato || ' e' stato effettuato per la ←
    merce ' || merce_da_ordinare || chr(13));

END;
```

2.6.7 calcolo_giacenza

Questa procedura si occupa di calcolare la giacenza della merce passata in input e trovata tramite *trova_merce*, al momento in cui viene lanciata la procedura.

Si sommano i dieci pezzi minimi della merce, che per regola aziendale sono sempre presenti in magazzino, al totale della merce acquistata tramite gli ordini rifornimento. Successivamente si trova il totale di pezzi di quella merce venduti e infine il calcolo vero e proprio della giacenza sarà dato dalla differenza tra i pezzi in magazzino e quelli venduti.

Vi possono essere casi in cui non siano stati venduti pezzi di quella merce, singolarmente, in menù standard o in offerte, in questo caso relativo totale sarà zero. Nel caso in cui non siamo mai stati venduti pezzi di quella merce allora la giacenza sarà data solo dal totale dei pezzi che si trovano in magazzino di quella merce.

Inoltre, poiché la procedura relativa al calcolo della giacenza richiama anche la procedura che effettua un ordine in caso di carenza di scorte, si è deciso di non inserirla in uno scheduler, in modo tale da avere pieno controllo su quanti ordini rifornimento si effettuano e quando.

```
CREATE OR REPLACE PROCEDURE calcolo_giacenza (nome_merce in ↵
    varchar)
IS
    merce                VARCHAR(20) := nome_merce;
    tot_merce_acquistata NUMBER;
    tot_merce_prod       NUMBER;
    tot_merce_mstd       NUMBER;
    tot_merce_off_prod   NUMBER;
    tot_merce_off_mstd   NUMBER;
    tot_merce_venduta    NUMBER;
    tot_merce_mag        NUMBER;
    giacenza             NUMBER;

BEGIN
    --controllo che sia effettivamente la merce di cui si vuole ↵
    --calcolare la giacenza
    DBMS_OUTPUT.PUT_LINE ('La merce di cui si sta per calcolare ↵
        la giacenza e' : ' || nome_merce );

    --trovare la quantita' totale di una merce acquistata
```

```

SELECT sum(lo.quantita_merce) INTO tot_merce_acquistata
FROM merce me JOIN lotto lo ON me.nome_merce = ↔
    lo.merce_contenuta
    JOIN ordine_rifornimento ord_rif ON ↔
        lo.ordine_rif = ord_rif.cod_ordine_rif
WHERE me.nome_merce = merce;

--trovare il totale di pezzi venduti di quel prodotto, e quindi ↔
di quella merce specifica presente nel prodotto
SELECT sum(pr_comp.quantita_merce * ↔
    or_con_pr.quantita_prodotto) INTO tot_merce_prod
FROM merce me JOIN prodotto_composto_da pr_comp ON ↔
    me.nome_merce = pr_comp.componente
    JOIN prodotto pr ON pr_comp.prodotto = ↔
        pr.cod_barre_prodotto
    JOIN ordine_contiene_prod or_con_pr ON ↔
        pr.cod_barre_prodotto = or_con_pr.prodotto
    JOIN ordine ord ON or_con_pr.cod_ordine = ↔
        ord.num_ordine
WHERE me.nome_merce = merce;

--trovare il totale di pezzi venduti dei prodotti presenti nei ↔
menu strandard, e quindi di quella specifica merce ↔
contenuta nel prodotto
SELECT sum(or_con_mstd.quantita_menu_std * ↔
    (m_con_pr.quantita_prodotto * pr_comp.quantita_merce)) ↔
    INTO tot_merce_mstd
FROM merce me JOIN prodotto_composto_da pr_comp ON ↔
    me.nome_merce = pr_comp.componente
    JOIN prodotto pr ON pr_comp.prodotto = ↔
        pr.cod_barre_prodotto
    JOIN menu_contiene_prod m_con_pr ON ↔
        pr.cod_barre_prodotto = m_con_pr.prodotto
    JOIN menu_standard mstd ON m_con_pr.cod_menu = ↔
        mstd.codice_menu
    JOIN ordine_contiene_menu or_con_mstd on ↔
        mstd.codice_menu = or_con_mstd.cod_menu
    JOIN ordine ord ON or_con_mstd.cod_ordine = ↔
        ord.num_ordine
WHERE me.nome_merce = merce;

--trovare il totale di pezzi venduti di quella merce presenti ↔
nelle offerte
--a) pezzi legati ai prodotti presenti nei menu standard

```

```

SELECT sum(of_con_pr.quantita_prodotto * ←
pr_comp.quantita_merce) INTO tot_merce_off_prod
FROM merce me JOIN prodotto_composto_da pr_comp ON ←
me.nome_merce = pr_comp.componente
JOIN prodotto pr ON pr_comp.prodotto = ←
pr.cod_barre_prodotto
JOIN offerta_contiene_prod of_con_pr ON ←
pr.cod_barre_prodotto = of_con_pr.prodotto
JOIN offerte oft ON of_con_pr.cod_offerta = ←
oft.codice_offerta
JOIN ordine_contiene_offerte or_con_of ON ←
oft.codice_offerta = or_con_of.offerta
JOIN ordine ord ON or_con_of.ordine = ←
ord.num_ordine
WHERE me.nome_merce = merce;

--b) pezzi legati ai singoli prodotti
SELECT sum(of_con_mstd.quantita_menu_std * ←
(m_con_pr.quantita_prodotto * pr_comp.quantita_merce)) ←
INTO tot_merce_off_mstd
FROM merce me JOIN prodotto_composto_da pr_comp ON ←
me.nome_merce = pr_comp.componente
JOIN prodotto pr ON pr_comp.prodotto = ←
pr.cod_barre_prodotto
JOIN menu_contiene_prod m_con_pr ON ←
pr.cod_barre_prodotto = m_con_pr.prodotto
JOIN menu_standard mstd ON m_con_pr.cod_menu = ←
mstd.codice_menu
JOIN offerta_contiene_menu of_con_mstd ON ←
mstd.codice_menu = of_con_mstd.cod_menu_std
JOIN offerte oft ON of_con_mstd.cod_offerta = ←
oft.codice_offerta
JOIN ordine_contiene_offerte or_con_of on ←
oft.codice_offerta = or_con_of.offerta
JOIN ordine ord ON or_con_of.ordine = ←
ord.num_ordine
WHERE me.nome_merce = merce;

--controlli per garantire di trovare i valori necessari al ←
calcolo della giacenza
IF (tot_merce_prod IS NULL)
THEN tot_merce_prod := 0;
END IF;

IF (tot_merce_mstd IS NULL)

```

```

        THEN tot_merce_mstd := 0;
    END IF;

    IF (tot_merce_off_prod IS NULL)
        THEN tot_merce_off_prod := 0;
    END IF;

    IF (tot_merce_off_mstd IS NULL)
        THEN tot_merce_off_mstd := 0;
    END IF;

    DBMS_OUTPUT.PUT_LINE ('Il totale dei pezzi acquistati di ←
        questa merce e' : ' || tot_merce_acquistata );
    DBMS_OUTPUT.PUT_LINE ('Il totale dei pezzi venduti di ←
        questa merce presente nei prodotti e' : ' || ←
        tot_merce_prod);
    DBMS_OUTPUT.PUT_LINE ('Il totale dei pezzi venduti di ←
        questa merce presente nei menu standard e' : ' || ←
        tot_merce_mstd);
    DBMS_OUTPUT.PUT_LINE ('Il totale dei pezzi venduti di ←
        questa merce presente nelle offerte contenenti un ←
        prodotto e' : ' || tot_merce_off_prod);
    DBMS_OUTPUT.PUT_LINE ('Il totale dei pezzi venduti di ←
        questa merce nelle offerte contenenti menu standard e' : ←
        ' || tot_merce_off_mstd);

--trovare il totale dei pezzi venduti di quella merce
    tot_merce_venduta := tot_merce_prod + tot_merce_mstd + ←
        tot_merce_off_prod + tot_merce_off_mstd;
    DBMS_OUTPUT.PUT_LINE ('Il totale dei pezzi venduti di ←
        questa merce e' : ' || tot_merce_venduta);

--trovare il totale dei prodotti in magazzino
    tot_merce_mag := tot_merce_acquistata + 10;
    DBMS_OUTPUT.PUT_LINE ('Il totale di pezzi presenti in ←
        magazzino dopo il piu' recente ordine rifornimento, ←
        sommati ai 10 pezzi sempre presenti, e' : ' || ←
        tot_merce_mag);

--trovare la giacenza
    giacenza := tot_merce_mag - tot_merce_venduta;
    DBMS_OUTPUT.PUT_LINE ('La giacenza attuale di questa merce ←
        e' : ' || giacenza || chr(13));

```



```

--controllo sulla giacenza per effettuare un ordine di ←
  rifornimento, quando necessario
  IF (giacenza < 10)
    THEN DBMS_OUTPUT.PUT_LINE ('Hai meno pezzi della soglia ←
      minima garantita');

    --richiama la procedura per generare un nuovo ordine di ←
    quella merce al fornitore che la vende al prezzo ←
    piu' basso
    crea_ordine (merce);

  END IF;
END;
```

2.7 Viste

Vista la mole di prodotti e, in generale, di informazioni presenti all'interno di questo database, per rendere più immediata la visualizzazione di alcuni dati sono state pensate alcune viste utilizzabili dai diversi utenti.

2.7.1 Componenti dei prodotti

Questa vista serve a mostrare quali merci sono coinvolte nella composizione di un prodotto.

Può essere molto utile al personale che si occupa della preparazione dei prodotti, oltre che essere vista permanentemente su totem o sullo schermo della cassa del fast food, così che nel caso in cui un cliente dovesse avere preferenze può rimuovere o aggiungere solo una delle componenti del prodotto.

```

create or replace view componenti_prodotto (nome_prodotto, ←
  componente, quantita_merce)
as
  select pr.nome_prodotto, me.nome_merce, ←
    pr_comp.quantita_merce
  from prodotto pr join prodotto_composto_da pr_comp on ←
    pr.cod_barre_prodotto = pr_comp.prodotto
    join merce me on pr_comp.componente = me.nome_merce
  order by pr.nome_prodotto;
```

2.7.2 Informazioni nutrizionali dei prodotti

Questa vista mostra tutte le informazioni nutrizionali legate ai prodotti. Utile per avere un quadro più completo del prodotto che si sta per consumare.

```
create or replace view info_prodotti (nome_prodotto, categoria, ←
    componente, quantita_per_100g, quantita_per_prodotto, RDA)
as
    select max(pr.nome_prodotto), max(pr.categoria_prod), ←
        max(info_nu.componente), ←
        max(info_nu.quantita_per_100g), ←
        max(info_nu.quantita_per_prod), max(info_nu.RDA)
    from prodotto pr join informazioni_nutrizionali info_nu ←
        on pr.cod_barre_prodotto = info_nu.prod_riferito
    order by pr.nome_prodotto;
```

2.7.3 Composizione offerte

La prima vista mostra da quali prodotti è composta un'offerta, la seconda da quali menù standard e composta, e l'ultima SELECT permette invece di vedere da cosa è composta un'offerta che contiene sia prodotti che menù standard.

```
create or replace view componenti_offerta_prodotto (offerta, ←
    inizio, fine, prezzo, prodotto, grandezza)
as
    select oft.codice_offerta, oft.data_inizio, ←
        oft.data_fine, oft.prezzo_offerta, pr.nome_prodotto, ←
        pr.grandezza_prod
    from offerte oft join offerta_contiene_prod oft_con_prod ←
        on oft.codice_offerta = oft_con_prod.cod_offerta
    join prodotto pr on oft_con_prod.prodotto = ←
        pr.cod_barre_prodotto
    order by oft.codice_offerta;
```

```
create or replace view componenti_offerta_menu_std (offerta, ←
    inizio, fine, prezzo, menu, quantita_menu, prodotto, ←
    grandezza)
as
    select oft.codice_offerta, oft.data_inizio, ←
        oft.data_fine, oft.prezzo_offerta, ←
        oft_con_mstd.cod_menu_std, ←
        oft_con_mstd.quantita_menu_std, pr.nome_prodotto, ←
        pr.grandezza_prod
```

```

from offerte oft join offerta_contiene_menu oft_con_mstd ↵
on oft.codice_offerta = oft_con_mstd.cod_offerta
join menu_standard mstd on oft_con_mstd.cod_menu_std ↵
= mstd.codice_menu
join menu_contiene_prod mstd_con_prod on ↵
mstd.codice_menu = mstd_con_prod.cod_menu
join prodotto pr on mstd_con_prod.prodotto = ↵
pr.cod_barre_prodotto
order by oft.codice_offerta;

```

```

select * from componenti_offerta_prodotto oft_prod join ↵
componenti_offerta_menu_std oft_menu on oft_prod.offerta = ↵
oft_menu.offerta;

```

2.7.4 Magazzino

Questa vista permette di visualizzare la merce contenuta in magazzino, in quale area si trovano, la loro conservazione e la loro categoria.

```

create or replace view mostra_merce_magazzino (codice_area, ↵
categoria_area, tipo_conservazione, merce_contenuta, ↵
categoria_merce)
as
select mag.cod_area, mag.categoria_area, ↵
mag.tipo_conservazione, me.nome_merce, ↵
me.categoria_merce
from magazzino mag join ordine_rifornimento ord_rif on ↵
mag.cod_area = ord_rif.codice_area_mag
join lotto lo on ord_rif.cod_ordine_rif = ↵
lo.ordine_rif
join merce me on lo.merce_contenuta = me.nome_merce
order by mag.cod_area;

```

2.7.5 Turni di lavoro

La vista seguente permette di visualizzare i turni di lavoro effettuati dai dipendenti, in quale giorno e gli orari di inizio e fine di ogni turno effettuato.

```

create or replace view mostra_turni (data_turno, ↵
tessera_personale, nome_personale, cognome_personale, ruolo)
as
select tu.data_turno, per.tessera, per.nome, ↵
per.cognome, per.ruolo

```

```
||      from personale per join turni tu on per.tessera = ↵  
||      tu.cod_personale  
||      order by per.tessera, tu.data_turno;
```