



UNIVERSITÀ DI PISA

Ingegneria Informatica

Basi di Dati - Progetto 2014/2015

Gabriele Marraccini

<gabriele@maffeimarraccini.it>

Niccolò Scatena

<speedjack95@gmail.com>

Lorenzo Tonelli

<lorenzo.tonelli.d.a@gmail.com>

5 marzo 2016

Indice

1	Analisi delle specifiche	5
1.1	Glossario dei termini	5
1.2	Business rules	9
2	Progettazione concettuale	13
2.1	Procedimento Strutturato	13
2.1.1	Variazioni	14
2.2	Rappresentazione concettuale	16
2.2.1	Entità	16
2.2.2	Generalizzazioni	16
2.2.3	Relazioni	18
2.3	Diagramma entità-relazione	24
3	Ristrutturazione del diagramma E-R	27
3.1	Eliminazione delle generalizzazioni	27
3.2	Eliminazione di attributi multivalore	31
3.3	Accorpamenti	31
3.4	Diagramma E-R ristrutturato	31
4	Progettazione logica	33
4.1	Schema logico	33
4.2	Vincoli di integrità referenziale	35
4.3	Vincoli di integrità generici con MySQL	36
5	Normalizzazione	57
5.1	Dipendenze funzionali	57
5.2	Sede e Account	61
5.3	Confezione	62
5.4	ModificaFase	62
5.5	QuestionarioSvolto	63

6	Tabelle utili	65
6.1	Tabelle di Log	65
6.2	Materialized View	67
7	Operazioni	71
7.1	Implementazione delle operazioni	72
8	Analisi delle prestazioni	77
8.1	Tavola dei volumi	77
8.2	Tavole degli accessi	81
9	Introduzione di ridondanze	85
9.1	MV_ClientiPrenotazione	85
9.2	Punteggio recensioni	86
9.3	Nuove operazioni	87
9.4	Nuove tavole degli accessi	89
10	Area Analytics	93
10.1	Magazzino intelligente	93
10.2	Analisi multidimensionale del business	94
10.3	Fornitura automatizzata del magazzino	99
10.4	Analisi dei consumi e degli sprechi	101
10.5	Qualità del take-away	103
11	Implementazione MySQL	107

Capitolo 1

Analisi delle specifiche

In questo Capitolo viene presentato il risultato della fase di *analisi delle specifiche*. Il paragrafo 1.1 contiene il *glossario dei termini* individuati nelle specifiche. Il paragrafo 1.2 a pagina 9 contiene una lista di *business rules*, alcune delle quali dettate dalle specifiche e le altre (ove permesso) scelte liberamente.

L'*area analytics* non richiede progettazione, per cui sarà trattata solo nel Capitolo 10 a pagina 93.

1.1 Glossario dei termini

Termine	Descrizione	Sinonimi	Collegamenti
Area Gestione			
Magazzino	Contiene la materia prima (ingredienti) in forma di confezioni.		Materia prima, Ingrediente, Sede, Confezione, Menu
Materia prima	Insieme di ingredienti contenuti in confezioni. Appartiene a un magazzino.	Insieme di ingredienti	Magazzino, Ingrediente, Confezione, Menu

Termine	Descrizione	Sinonimi	Collegamenti
Ingrediente	Compone le pietanze. Viene mantenuto nel magazzino all'interno di confezioni.	Componente	Magazzino, Materia prima, Confezione, Ricetta, Procedimento, Fase, Piatto, Menu, Proposta
Sede	Una sede della catena di ristorazione. Contiene uno o più magazzini. Possiede dei pony. Ha una cucina. È divisa in sale. Ha un questionario compilabile online.	Ristorante	Magazzino, Cucina, Menu, Comanda, Prenotazione, Tavolo, Sala, Pony, Recensione, Questionario, Allestimento
Confezione	Contiene una certa quantità di un certo ingrediente. Viene caricata o scaricata dal magazzino. Può venir danneggiata nei trasporti. Può essere in stoccaggio o in ordine presso un magazzino.		Magazzino, Materia prima, Ingrediente
Cucina	Si trova in una sede. È composta da macchinari e attrezzature.		Sede, Attrezzatura, Macchinario
Attrezzatura	Usata in cucina.	Attrezzo	Cucina, Procedimento, Fase
Macchinario	Usato in cucina.	Macchina	Cucina, Procedimento, Fase

Termine	Descrizione	Sinonimi	Collegamenti
Ricetta	Costituita da fasi di preparazione e ingredienti. Elencata in menu. Lo chef permette variazioni possibili per ogni ricetta. Può essere recensita.	Pietanza	Ingrediente, Procedimento, Fase, Piatto, Menu, Variazione, Recensione, Suggerimento
Procedimento	Insieme di fasi di una ricetta.	Procedimento strutturato, Procedura	Ingrediente, Ricetta, Fase, Variazione, Suggerimento
Fase	Fase del procedimento di una ricetta. Può rappresentare l'aggiunta di un ingrediente o l'uso di un macchinario o di un attrezzo.		Ingrediente, Attrezzatura, Macchinario, Ricetta, Procedimento, Variazione, Suggerimento
Piatto	Ricetta in essere con possibili variazioni. Preparato dai cuochi in seguito a una comanda. Ordinato dai clienti al tavolo o online (take-away) con consegna.		Ingrediente, Ricetta, Comanda, Variazione, Tavolo, Consegna, Recensione
Menu	Lista delle ricette disponibili. Ogni sede ha menu diverso. Cambia con cadenza non regolare. Devono comparire solo ricette producibili con ingredienti presenti in magazzino.		Magazzino, Materia prima, Ingrediente, Sede, Ricetta
Comanda	Lista di piatti ordinati da un tavolo di una certa sede o tramite consegna take-away.	Ordine, Ordinazione	Sede, Piatto, Variazione, Tavolo, Consegna, Account

Termine	Descrizione	Sinonimi	Collegamenti
Variazione	Modifica di una fase del procedimento di una ricetta. Scelte dai clienti nelle comande.	Modifica	Ricetta, Procedimento, Fase, Piatto, Comanda
Prenotazione	Prenotazione di un tavolo di una sede da parte di un cliente. Può essere fatta anche da sito web. Può contenere anche allestimenti.		Sede, Tavolo, Sala, Account, Allestimento
Tavolo	Tavolo di una sala in una sede.		Sede, Piatto, Comanda, Prenotazione, Sala
Sala	Area contenente tavoli di una sede. Possono essere prenotate per allestimenti.		Sede, Prenotazione, Tavolo, Allestimento
Pony	Incaricato ad effettuare consegne a domicilio.		Sede, Comanda, Consegna
Consegna	La consegna effettuata da un pony relativamente ad una comanda take-away.		Piatto, Comanda, Pony
Area Clienti			
Account	Account di un cliente sul sito.	Profilo, Utente	Comanda, Prenotazione, Recensione, Valutazione, Proposta, Suggerimento, Gradimento, Allestimento
Recensione	Recensione con questionario da parte di un account su un piatto o sede.		Sede, Ricetta, Piatto, Account, Questionario, Valutazione

Termine	Descrizione	Sinonimi	Collegamenti
Questionario	Insieme di domande scelte dal direttore e chieste ai clienti durante le recensioni.		Sede, Recensione, Domanda, Risposta
Domanda	Domanda di questionario con un range di risposte possibili.		Questionario, Risposta
Risposta	Risposta a una domanda.		Questionario, Domanda
Valutazione	Valutazione di una recensione.		Account, Recensione
Proposta	Proposta di una nuova ricetta in forma di lista di ingredienti.		Ingrediente, Account Gradimento
Suggerimento	Suggerimento di modifica delle fasi di una ricetta.	Consiglio	Ricetta, Procedimento, Fase, Account, Gradimento
Gradimento	Valutazione di un suggerimento o di una proposta.		Account, Proposta, Suggerimento
Allestimento	Prenotazione di una sala di una sede da parte di un cliente per una serata a tema. Deve essere approvato. Salvato come prenotazione speciale.		Sede, Prenotazione, Sala, Account

1.2 Business rules

Elenchiamo di seguito alcune delle *business rules*:

- (BR01) Ogni sede ha a disposizione più magazzini nei quali avviene lo stoccaggio degli ingredienti su varie scaffalature.
- (BR02) Un ingrediente può rivestire un ruolo principale in un piatto solo se l'aspetto non è stato rovinato durante il trasporto.
- (BR03) Il procedimento strutturato deve consentire l'individuazione dell'esatto ordine in cui gli ingredienti devono essere uniti, il loro dosaggio, le manovre da compiere su ogni ingrediente e quali sono i macchinari e le attrezzature usate in ogni fase.

- (BR04) Ogni sede può avere un menu diverso.
- (BR05) In ogni sede deve essere applicato un solo menu alla volta.
- (BR06) Il menu applicato in una sede cambia con cadenza non regolare.
- (BR07) Il database deve mantenere le informazioni sui menu passati.
- (BR08) La data di fine dell'applicazione di un menu deve essere inserita al momento dell'inserimento del menu nel database.
- (BR09) Una pietanza deve comparire nel menu se e solo se ci sono ingredienti a sufficienza (la quantità viene stimata da funzioni di back-end) per la sua produzione, nel magazzino o in arrivo (in tal caso, la pietanza compare nel menu solo se l'ingrediente arriva al max. 3 giorni prima dell'entrata in vigore del menu).
- (BR10) Un cliente può ordinare i piatti anche in più comande successive.
- (BR11) Ogni variazione deve poter modificare anche più di una fase del procedimento strutturato.¹
- (BR12) Un cliente può apportare un massimo di 3 variazioni per ogni piatto, scelte da una lista di variazioni possibili creata dagli chef.
- (BR13) Le comande evase (completate) devono essere mantenute nel database.
- (BR14) Una prenotazione può essere fatta telefonicamente o mediante il sito Web, previa creazione di un account.
- (BR15) Il cliente, al momento della prenotazione mediante il sito web, può scegliere un particolare tavolo. Ulteriori informazioni che il cliente deve specificare al momento della prenotazione sono giorno e orario di prenotazione, e il numero di persone (se la prenotazione è effettuata telefonicamente, deve lasciare anche il recapito telefonico). Se il cliente non seleziona un particolare tavolo, allora gli sarà assegnato automaticamente da una funzionalità di back-end.
- (BR16) Se non ci sono tavoli disponibili con un numero sufficiente di posti, la prenotazione non può essere effettuata.
- (BR17) Una prenotazione, una volta eseguita, è rettificabile con un anticipo minimo di 48 ore.
- (BR18) L'annullamento di una prenotazione è possibile fino a 72 ore prima.

¹Il motivo di questa business rule è spiegato nel paragrafo 2.1 a pagina 13.

- (BR19) Se il cliente che ha effettuato una prenotazione mediante il sito web non si presenta, l'area del sito web nella quale si effettuano le prenotazioni diviene non fruibile per tale cliente.
- (BR20) Un pony può prendersi in carico una consegna se e solo se il suo stato è *libero* (non è in viaggio per un'altra consegna).
- (BR21) Quando tutti i piatti della comanda take-away entrano nello stato *servizio* deve esserle assegnato automaticamente un pony *libero*.
- (BR22) Un utente, quando rilascia una recensione, deve rispondere anche alle domande di un questionario. Per ogni domanda può dare solo una risposta tra una lista di risposte predefinite.
- (BR23) Le serate a tema possono essere organizzate solo se a tali iniziative partecipa un numero di clienti superiore ad una determinata soglia, stabilita dalla direzione.

Capitolo 2

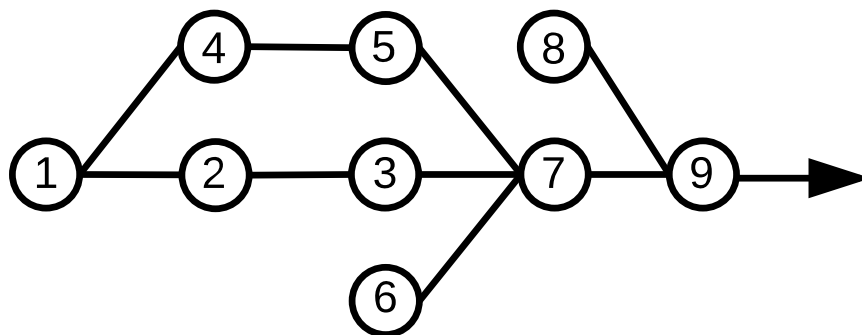
Progettazione concettuale

In questo Capitolo viene presentato il risultato della fase di *progettazione concettuale*. Il paragrafo 2.1 descrive l'implementazione del *procedimento strutturato* e delle *variazioni*. Il paragrafo 2.2 a pagina 16 elenca e descrive le entità, le generalizzazioni e le relazioni individuate durante la progettazione. Infine, il paragrafo 2.3 a pagina 24 riporta il *diagramma concettuale* prodotto.

2.1 Procedimento Strutturato

Prima di poter procedere è necessario decidere come realizzare il **procedimento strutturato**. Esso dovrà anche integrarsi al sistema delle *variazioni* e dei *suggerimenti* dei clienti.

Il modo più *flessibile* di rappresentare un procedimento di preparazione di un piatto in fasi è quello di una struttura a *branches* — ossia rappresentare il procedimento di preparazione di una ricetta tramite una struttura come la seguente:



Dove ogni *nodo numerato* rappresenta una *fase*. Ogni fase (nodo) ha zero o più fasi che la *precedono* e zero o più fasi che la *seguono*. Le uniche regole da imporre sono: deve esserci *una ed una sola* fase (F_9) che non ha alcuna fase successiva (un solo *exit-node*), altrimenti si verrebbe a produrre più di un piatto (che è assurdo); non devono esserci *cicli*, altrimenti la produzione del piatto non terminerebbe mai (altrettanto assurdo).

Con una struttura del genere è possibile ad esempio imporre che per iniziare una fase, tutte le fasi precedenti ad essa devono essere portate a termine. Ad esempio nella struttura precedente: F_5 può essere iniziata solo dopo che F_4 (e di conseguenza anche F_1) è stata completata. F_7 può essere iniziata solo dopo che F_3 (e di conseguenza anche F_2 e F_1), F_5 (e di conseguenza anche F_4) e F_6 sono state completate.

Possiamo inoltre *parallelizzare* la produzione del piatto. Ad esempio F_4 e F_5 possono essere svolte da un cuoco mentre un altro cuoco sta svolgendo F_2 e F_3 (l'importante è che F_1 sia stata portata a termine prima di iniziare F_2 e F_4). Nel mentre un altro cuoco ancora può svolgere F_6 (in questo caso F_6 può iniziare anche prima di F_1) e un altro ancora può svolgere F_8 .

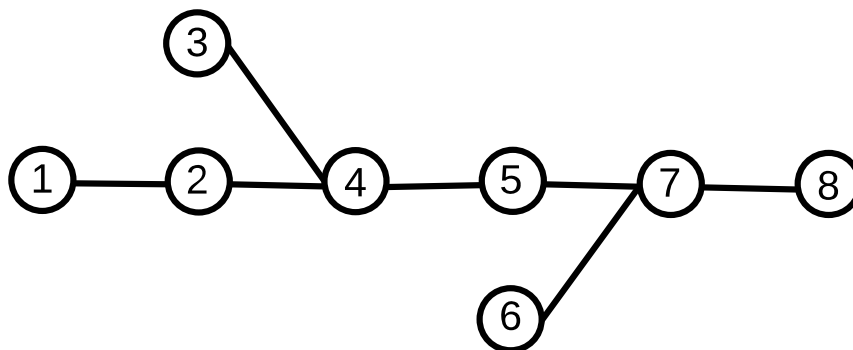
Per poter realizzare una struttura come questa in un database relazionale, sarà necessario introdurre una *relazione ricorsiva*: l'entità che rappresenterà una fase dovrà essere messa in una *relazione molti-a-molti* con se stessa. Infatti ogni fase deve essere messa in relazione con le sue fasi precedenti (nell'immagine precedente: ogni collegamento tra due fasi è una relazione). La relazione (**Precedente**) che permette la realizzazione di tale struttura è mostrata nel *diagramma entità-relazione* a pagina 25.

2.1.1 Variazioni

Con la struttura mostrata nel paragrafo 2.1 nella pagina precedente è possibile anche gestire le *variazioni*.

Le variazioni devono poter: aggiungere nuove fasi; rimuovere fasi.

Prendiamo ad esempio la seguente struttura che rappresenta il procedimento di una pizza al prosciutto:



Dove:

F_1 : fase di stesura della pasta;

F_2 : fase di aggiunta della pasta al piatto;

F_3 : fase di produzione del sugo di pomodoro;

F_4 : fase di aggiunta del pomodoro lavorato (sugo);

F_5 : fase di aggiunta della mozzarella;

F_6 : fase di taglio del prosciutto in fette;

F_7 : fase di aggiunta del prosciutto lavorato (fette di prosciutto);

F_8 : fase di cottura.

Si noti che con tale struttura ci sono più modi per rappresentare lo stesso procedimento di preparazione di un piatto: ad esempio si potrebbe scambiare F_1 con F_2 senza cambiare il senso del procedimento; oppure si potrebbe spostare la fase F_7 a prima della fase F_6 e inserire una nuova fase al posto vuoto lasciato da F_7 che indica di mettere insieme i due semilavorati (la pizza col pomodoro e la mozzarella e le fette di prosciutto).

Facciamo l'ipotesi che il cliente voglia come variazione la rimozione del pomodoro (pizza bianca). In tal caso devono essere rimosse due fasi (anche se la variazione è una sola!): F_3 e F_4 . Se invece quello precedente fosse stato il procedimento di una pizza margherita e il cliente avesse indicato come variazione l'aggiunta del prosciutto, si sarebbero dovute aggiungere due fasi: F_6 e F_7 .

Questo dimostra che è necessario che ogni variazione possa effettuare anche più di un'operazione elementare (aggiunta o rimozione) sul procedimento strutturato.

L'entità **ModificaFase** (con le sue entità figlie) è stata aggiunta al *diagramma entità-relazione* (disponibile a pagina 25) allo scopo di permettere ad ogni variazione di *modificare* anche più fasi: ogni variazione ha più **ModificaFase** e ogni **ModificaFase** rappresenta la modifica (aggiunta, eliminazione, sostituzione) di una e una sola fase (la sostituzione coinvolge però due fasi: quella da togliere e quella da aggiungere).

Le fasi che fanno parte di variazioni (come le fasi F_6 e F_7 nel caso del procedimento della pizza margherita) non vengono confuse dal database con le fasi della ricetta *normale*: infatti le fasi che fanno parte di variazioni si troveranno come *chiavi esterne* all'interno dell'entità **AggiuntaFase** (o **SostituzioneFase**) per mezzo della relazione **Nuova**, e quindi il database saprà distinguerle da quelle della ricetta normale (vedere il diagramma a pagina 25).

2.2 Rappresentazione concettuale

2.2.1 Entità

Dalle specifiche di progetto e dal glossario dei termini prodotto nel paragrafo 1.1 a pagina 5 del Capitolo precedente, si individuano facilmente le seguenti entità:

- | | | |
|----------------------|---------------|-----------------------------|
| • Sede | • Magazzino | • Confezione |
| • Ingrediente | • Macchinario | • Attrezzatura |
| • Menu | • Ricetta | • Fase |
| • Piatto | • Variazione | • ModificaFase ¹ |
| • Comanda | • Tavolo | • Sala |
| • Consegna | • Pony | • Prenotazione |
| • Account | • Proposta | • Suggerimento |
| • Gradimento | • Recensione | • Valutazione |
| • Questionario | • Domanda | • Risposta |
| • QuestionarioSvolto | | |

Queste entità sono state individuate procedendo con una strategia **inside-out** partendo da **Sede** per l'area gestione e da **Account** per l'area clienti. Come si può infatti notare dalla lista dei collegamenti nel glossario dei termini, queste entità sono fondamentali e contengono un alto numero di collegamenti: risulta quindi facile muoversi *a macchia d'olio* partendo da esse. Le entità della lista precedente sono poste nell'esatto ordine in cui sono state aggiunte al diagramma durante la fase di progettazione concettuale.

2.2.2 Generalizzazioni

- Le due entità **Attrezzatura** e **Macchinario** sono entrambi strumenti da cucina. Condividono molte caratteristiche e per tal motivo è utile introdurre un'entità padre: **Strumento**. La generalizzazione è *totale*, in quanto in cucina si possono avere solo attrezzature o macchinari, ed *esclusiva*.
- Ogni **Menu** che è stato sostituito da uno nuovo deve essere comunque mantenuto nel database. Per tale motivo è utile aggiungere l'entità **MenuPassato** che rappresenta un menu non più in uso. In questo modo **MenuPassato** è un sottoinsieme di **Menu** e quindi è una generalizzazione *parziale*.

¹Il motivo dell'introduzione di questa entità è spiegato nel paragrafo 2.1.1 a pagina 14.

- La **Fase** può essere di due tipi: **FaseIngrediente**, ossia di aggiunta di ingrediente, oppure **FaseManovra**, ossia di lavorazione (con o senza strumento). La generalizzazione è *totale*. In teoria si potrebbe trattare come generalizzazione *sovrapposta*, ossia che una fase possa contenere sia l'aggiunta di un ingrediente sia una lavorazione (ad esempio: *affetta il prosciutto e aggiungilo al piatto*). Preferiamo però rendere le fasi quanto più elementari possibili. L'esempio precedente si può vedere anche come diviso in due fasi: *affetta il prosciutto; aggiungi il prosciutto*.
- **ModificaFase** può essere di due tipi: **AggiuntaFase** oppure **EliminazioneFase**. La generalizzazione è *totale* e *sovrapposta* (la sostituzione di una fase con un'altra, infatti, altro non è che l'eliminazione di una fase e l'aggiunta di un'altra fase). Introduciamo quindi, per rendere *esclusiva* la generalizzazione, un'ulteriore entità: **SostituzioneFase**.
- La **Comanda** può essere di due tipi: **ComandaTavolo** oppure **ComandaTakeAway**. La generalizzazione è *totale* ed *esclusiva*.
- La **Prenotazione** può essere di due tipi: una **PrenotazioneOnline** (dal sito) oppure una **PrenotazioneTelefonica**. La generalizzazione è *totale* ed *esclusiva*. Una **PrenotazioneOnline** a sua volta può essere una semplice **PrenotazioneOnline** oppure un **Allestimento**. In questo caso la generalizzazione è *parziale* e **Allestimento** è un sottoinsieme di **PrenotazioneOnline**.
- **Variazione** e **Suggerimento** sono due entità molto simili tra loro: entrambe modificano una o più fasi del procedimento strutturato di una certa ricetta. Può quindi essere comodo generalizzarle in un'unica entità: rinominiamo quindi l'entità **Variazione** in **VariazionePiatto** così da poter utilizzare il nome **Variazione** per l'entità padre di **VariazionePiatto** e **Suggerimento**. La generalizzazione è *totale* ed *esclusiva*.
- Un **Gradimento** può riferirsi a un **Suggerimento** o a una **Proposta**. Risulta quindi comodo aggiungere le entità **GradimentoSuggerimento** e **GradimentoProposta** come figlie. La generalizzazione è *totale* ed *esclusiva*.

Devono essere quindi aggiunte le seguenti entità:

- | | |
|--------------------|----------------------|
| • Strumento | • MenuPassato |
| • FaseIngrediente | • FaseManovra |
| • AggiuntaFase | • EliminazioneFase |
| • SostituzioneFase | • ComandaTavolo |
| • ComandaTakeAway | • PrenotazioneOnline |

- PrenotazioneTelefonica
- VariazionePiatto
- GradimentoProposta
- Allestimento
- GradimentoSuggerimento

2.2.3 Relazioni

Nella seguente tabella sono elencate e descritte tutte le *relazioni* (o *associazioni*), inserite nell'ordine in cui sono state individuate.

Relazione	Entità A (Cardinalità)	Entità B (Cardinalità)	Descrizione
Disponibilità	Sede (1,N)	Magazzino (1,1)	Una sede può avere uno o più magazzini. Un magazzino appartiene solo a una sede.
Stoccaggio	Magazzino (0,N)	Confezione (0,1)	Le confezioni sono mantenute all'interno dei magazzini. Una confezione può anche essere in ordine (e quindi non in stoccaggio).
InOrdine	Magazzino (0,N)	Confezione (0,1)	Una confezione può essere in ordine o in stoccaggio presso un magazzino. Un magazzino può avere in ordine molte confezioni.
Contenuto	Confezione (1,1)	Ingrediente (0,N)	Una confezione contiene un solo ingrediente. Un certo ingrediente può essere contenuto in più confezioni (nessuna nel caso di ingrediente esaurito).
Cucina	Sede (0,N)	Strumento (0,N)	In una sede ci possono essere più tipologie di strumenti. Un tipo di strumento può essere utilizzato in più sedi.
Applicazione	Sede (1,N)	Menu (1,1)	Una sede applica un menu alla volta. I menu passati vengono comunque mantenuti ma un menu appartiene solo ad una sede.

Relazione	Entità A (Cardinalità)	Entità B (Cardinalità)	Descrizione
Elenco	Menu (1,N)	Ricetta (0,N)	Una ricetta appartiene a zero o più menu e un menu possiede più ricette (almeno una).
Procedimento	Ricetta (1,N)	Fase (1,1)	Ogni ricetta è formata da almeno una fase. Ogni fase appartiene a una sola ricetta.
Precedente	Fase (0,N)	Fase (0,N)	Questa relazione permette di rappresentare la sequenza delle fasi. Una fase può precedere zero o più fasi e avere zero o più fasi precedenti.
Aggiunta	Fase (1,1)	Ingrediente (0,N)	Un ingrediente può essere usato da zero o più fasi. Una fase aggiunge al massimo un ingrediente.
Utilizzo	Fase (0,1)	Strumento (0,N)	Uno strumento può essere utilizzato in zero o più fasi. Una fase può utilizzare al massimo uno strumento.
Produzione	Ricetta (0,N)	Piatto (1,1)	Un piatto è prodotto secondo una sola ricetta. Ogni ricetta può essere scelta anche più volte da tanti clienti (e per ogni scelta viene prodotto un piatto).
Modifica	Piatto (0,N)	VariazionePiatto (0,N)	Un piatto può applicare da zero a tre variazioni. Una variazione può essere scelta in più piatti (della stessa ricetta).

Relazione	Entità A (Cardinalità)	Entità B (Cardinalità)	Descrizione
Procedura	Variazione (1,N)	ModificaFase (1,1)	Una variazione prevede la modifica di almeno una fase. Ogni modifica di fase appartiene a una sola variazione.
Nuova	AggiuntaFase (1,1)	Fase (0,N)	Un'aggiunta di fase aggiunge una fase. Una fase può essere oggetto di modifica da parte di più variazioni.
Vecchia	EliminazioneFase (1,1)	Fase (0,N)	Un'eliminazione di fase rimuove una fase. Una fase può essere oggetto di modifica da parte di più variazioni.
Nuova	SostituzioneFase (1,1)	Fase (0,N)	Una sostituzione di fase aggiunge una fase. Una fase può essere oggetto di modifica da parte di più variazioni.
Vecchia	SostituzioneFase (1,1)	Fase (0,N)	Una sostituzione di fase rimuove una fase. Una fase può essere oggetto di modifica da parte di più variazioni.
Ordine	Comanda (1,N)	Piatto (1,1)	Una comanda può ordinare uno o più piatti. Ogni piatto è ordinato da una sola comanda.
Gestione	Sede (0,N)	Comanda (1,1)	Una sede può aver ricevuto zero o più comande. Una comanda è inviata a una sola sede.
Suddivisione	Sede (1,N)	Sala (1,1)	Una sede è suddivisa in più sale, minimo una. Ogni sala appartiene a una Sede.

Relazione	Entità A (Cardinalità)	Entità B (Cardinalità)	Descrizione
Disposizione	Sala (1,N)	Tavolo (1,1)	Una sala può contenere uno o più tavoli. Ogni tavolo è sempre situato in una sala.
Mittente	Tavolo (0,N)	ComandaTavolo (1,1)	Un tavolo può aver inviato zero o più comande. Una comanda è inviata da un solo tavolo.
Richiesta	Comanda– TakeAway (0,1)	Consegna (1,1)	Ogni consegna corrisponde a una sola comanda. A una comanda viene associata una consegna solo quando è pronta.
Flotta	Sede (0,N)	Pony (1,1)	Un pony lavora presso una sola sede. Ogni flotta è composta anche da più pony.
Trasporto	Consegna (1,1)	Pony (0,N)	Un pony può aver effettuato zero o più consegne. Ogni consegna è gestita da un solo pony.
Riserva	Prenotazione (0,1)	Tavolo (0,N)	Un tavolo può essere stato riservato da più prenotazioni (in momenti diversi). Una prenotazione può riservare solo un tavolo.
InvioPre	Account (0,N)	Prenotazione– Online (1,1)	Con un account è possibile effettuare prenotazioni online. Una prenotazione può essere effettuata da un solo account.
Ordinazione	Account (0,N)	Comanda– TakeAway (1,1)	Da un account è possibile effettuare ordinazioni. Ogni comanda take-away è inviata da un solo account.
InvioPro	Account (0,N)	Proposta (1,1)	Con un account è possibile inviare delle proposte. Ogni proposta è inviata da un solo account.

Relazione	Entità A (Cardinalità)	Entità B (Cardinalità)	Descrizione
Composizione	Proposta (1,N)	Ingrediente (0,N)	Una proposta è composta da almeno un ingrediente. Un ingrediente può essere presente in zero o più proposte.
InvioSug	Account (0,N)	Suggerimento (1,1)	Un cliente che ha un account può rilasciare suggerimenti per le ricette. Un suggerimento può essere inviato da un solo account.
InvioGra	Account (0,N)	Gradimento (1,1)	Con un account è possibile inviare zero o più gradimenti. Ogni gradimento è inviato da un solo account.
RiferimentoP	Gradimento– Proposta (1,1)	Proposta (0,N)	Ogni gradimento si riferisce solo a una proposta. Una proposta può essere valutata da zero o più gradimenti.
RiferimentoS	Gradimento– Suggerimento (1,1)	Suggerimento (0,N)	Un gradimento valuta un solo suggerimento. Un suggerimento può essere valutato da zero o più gradimenti.
Rilascio	Account (0,N)	Recensione (1,1)	Con un account è possibile rilasciare più recensioni. Ogni recensione è rilasciata da un solo account.
InvioVal	Account (0,N)	Valutazione (1,1)	Con un account è possibile inviare valutazioni. Ogni valutazione è rilasciata da un solo account.
RelativaA	Valutazione (1,1)	Recensione (0,N)	Una recensione può avere nessuna o molte valutazioni. Ogni valutazione valuta una sola recensione.

Relazione	Entità A (Cardinalità)	Entità B (Cardinalità)	Descrizione
Consiglio	Recensione (1,1)	Ricetta (0,N)	Ogni recensione si riferisce a una sola ricetta. Una ricetta può essere stata recensita zero o più volte.
Relazione	Sede (1,1)	Questionario (1,1)	Ogni sede ha uno e un solo questionario. Ogni questionario appartiene a una sola sede.
Quiz	Questionario (1,N)	Domanda (1,1)	Un questionario è composto da una o più domande. Una domanda appartiene a un solo questionario.
Opzione	Domanda (1,N)	Risposta (1,1)	Una domanda può avere una o più risposte (di solito almeno due). Ogni risposta si riferisce a una sola domanda.
IstanzaDi	Questionario (0,N)	Questionario– Svolto (1,1)	Un questionario svolto è istanza di un solo questionario. Un questionario può essere svolto da diverse persone in diverse recensioni.
Compilazione	Questionario– Svolto (1,N)	Risposta (0,1)	Un questionario svolto contiene le risposte alle domande. Una risposta può essere stata data in zero o più questionari svolti.
Appartenenza	Questionario– Svolto (1,1)	Recensione (1,1)	Un questionario svolto appartiene a una e una sola recensione. Una recensione ha un solo questionario svolto.
SerataTema	Allestimento (1,1)	Sala (0,N)	Un allestimento prenota una sola sala per una serata a tema. In una sala possono essere state svolte zero o più serate a tema.

2.3 Diagramma entità-relazione

Il *diagramma concettuale* finale è mostrato nella pagina successiva.

La spiegazione di come sono stati realizzati il *procedimento strutturato* e le *variazioni* (e quindi anche i *suggerimenti*) è disponibile nel paragrafo 2.1 a pagina 13.

Capitolo 3

Ristrutturazione del diagramma E-R

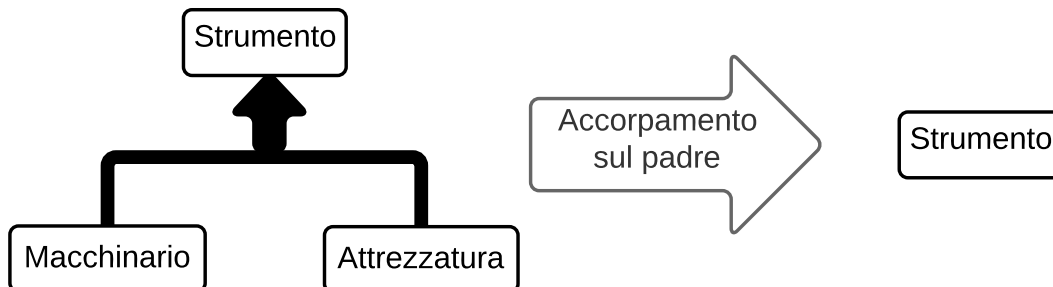
In questo Capitolo viene presentato il risultato della fase di *ristrutturazione del diagramma entità-relazione*. Il paragrafo 3.1 descrive come sono state eliminate le *generalizzazioni*. Il paragrafo 3.2 a pagina 31 descrive come sono stati eliminati gli *attributi multivalore*. Il paragrafo 3.3 a pagina 31 elenca gli altri *accorpamenti* effettuali per semplificare il diagramma. Infine, il paragrafo 3.4 a pagina 31 riporta il *diagramma ristrutturato* finale.

3.1 Eliminazione delle generalizzazioni

Di seguito viene mostrato come le generalizzazioni sono state eliminate per produrre il *diagramma E-R ristrutturato*.

Nella totalità dei casi le generalizzazioni sono state eliminate **accorpando le entità figlie sull'entità padre**.

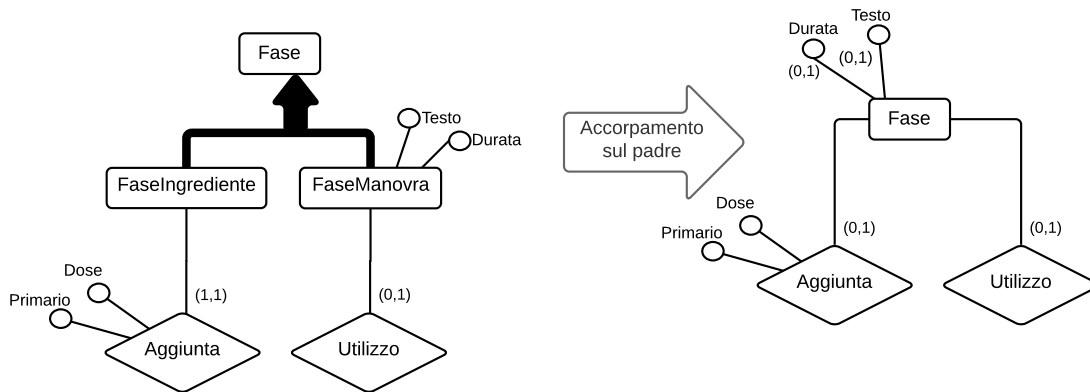
Nel caso della generalizzazione che coinvolge le entità **Strumento**, **Macchinario** e **Attrezzatura**, risulta comodo l'accorpamento sul padre in quanto le entità figlie non hanno attributi che il padre non possiede e non hanno associazioni con altre entità.



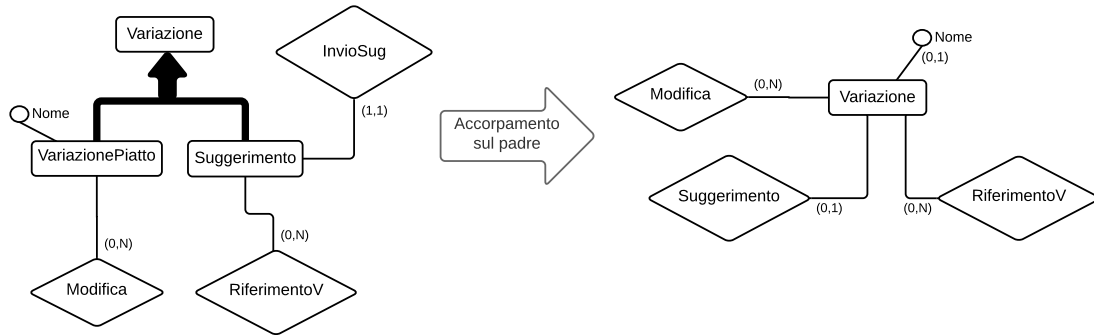
Nel caso della generalizzazione che coinvolge le entità **Menu** e **MenuPassato**, l'accorpamento sul padre non porta all'aggiunta di alcun attributo opzionale (l'attributo **DataFine** è specificato al momento dell'inserimento del menu nel database, come specificato dalla business rule (BR08) — grazie ad essa si evita l'aggiunta di un attributo *nullable*).



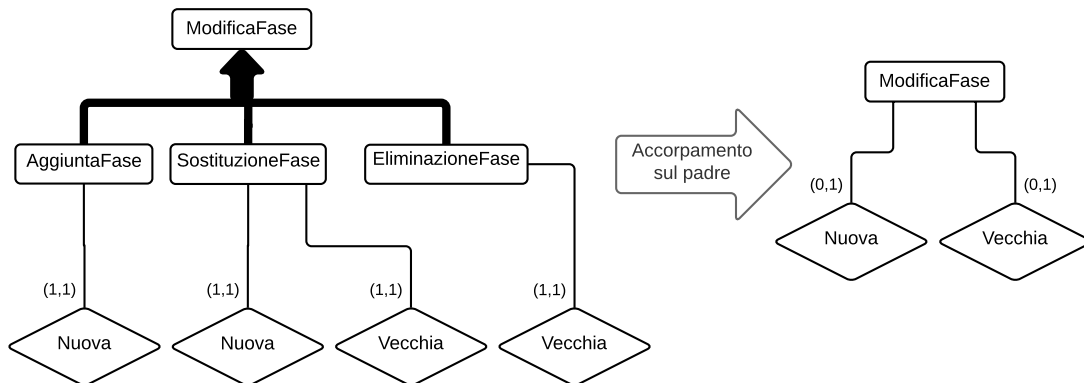
Nel caso della generalizzazione che coinvolge le entità **Fase**, **FaseIngrediente** e **FaseManovra**, l'accorpamento sul padre porta all'introduzione di diversi attributi opzionali: **Durata**, **Testo** ma anche **Dose** e **Primario** in quanto l'associazione **Aggiunta** diventa opzionale. Non è necessario inserire qui un attributo **Tipo** per riconoscere il tipo di fase in quanto può essere facilmente dedotto dalla presenza o meno dell'associazione **Aggiunta**.



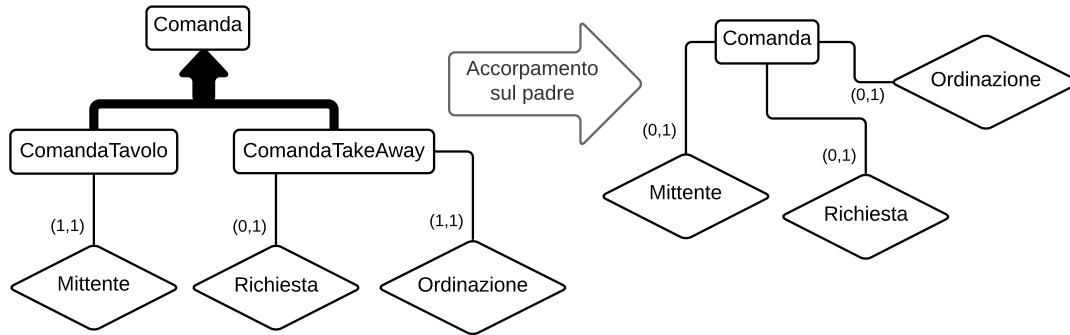
Nel caso della generalizzazione che coinvolge le entità **Variazione**, **VariazionePiatto** e **Suggerimento**, l'accorpamento sul padre porta all'aggiunta dell'attributo opzionale **Nome**. L'associazione **InvioSug** diventa opzionale (e la rinominiamo in **Suggerimento**). Non è necessario inserire qui un attributo **Tipo** per riconoscere il tipo di variazione in quanto può essere facilmente dedotto dalla presenza o meno dell'associazione **Suggerimento** (ex-InvioSug).



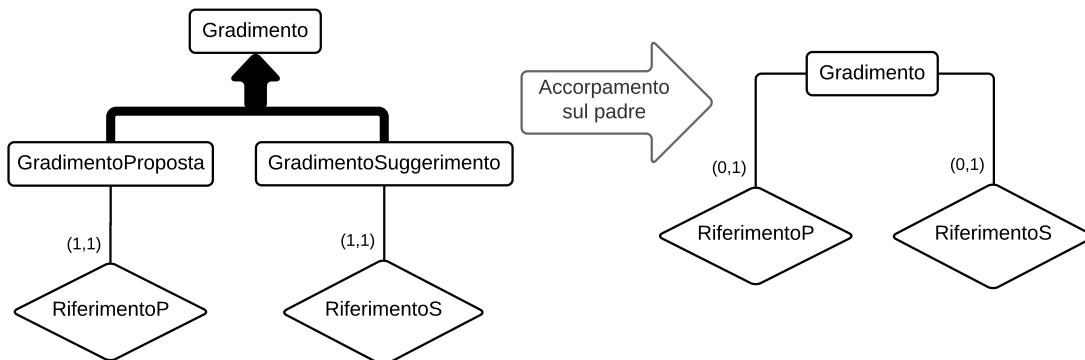
Nel caso della generalizzazione che coinvolge le entità **ModificaFase**, **AggiuntaFase**, **EliminazioneFase** e **SostituzioneFase**, l'accorpamento sul padre porta all'eliminazione di due associazioni (una **Nuova** e una **Vecchia**). Non è necessario introdurre qui un attributo **Tipo** per riconoscere il tipo di modifica di fase in quanto può essere facilmente dedotto dalla presenza o meno delle associazioni **Nuova** e **Vecchia**: **Nuova** = **AggiuntaFase**; **Vecchia** = **EliminazioneFase**; **Nuova** + **Vecchia** = **SostituzioneFase**.



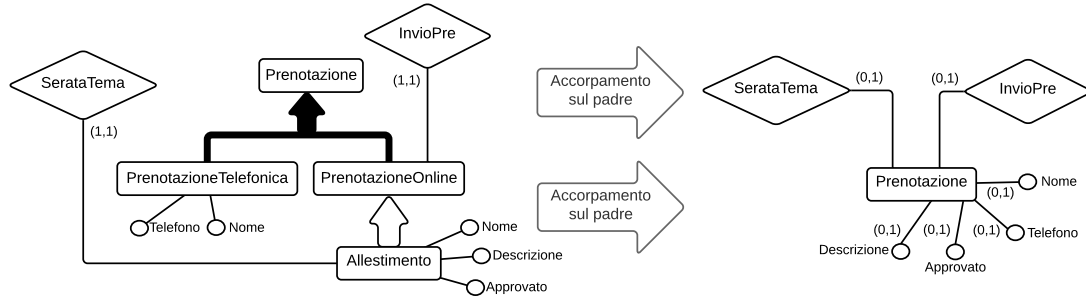
Nel caso della generalizzazione che coinvolge le entità **Comanda**, **ComandaTavolo** e **ComandaTakeAway**, l'accorpamento sul padre non porta all'aggiunta di attributi opzionali. Ma le associazioni **Mittente** e **Ordinazione** diventano opzionali (**Richiesta** è già opzionale). Non è necessario inserire qui un attributo **Tipo** per riconoscere il tipo di comanda in quanto può essere facilmente dedotto dalla presenza o meno dell'associazione **Ordinazione**.



Nel caso della generalizzazione che coinvolge le entità **Gradimento**, **GradimentoProposta** e **GradimentoSuggerimento**, l'accorpamento sul padre non porta all'aggiunta di attributi opzionali. Ma le associazioni **RiferimentoP** e **RiferimentoS** diventano opzionali. Non è necessario inserire qui un attributo **Tipo** per riconoscere il tipo di comanda in quanto può essere facilmente dedotto dall'associazione presente (se **RiferimentoP** o **RiferimentoS**).

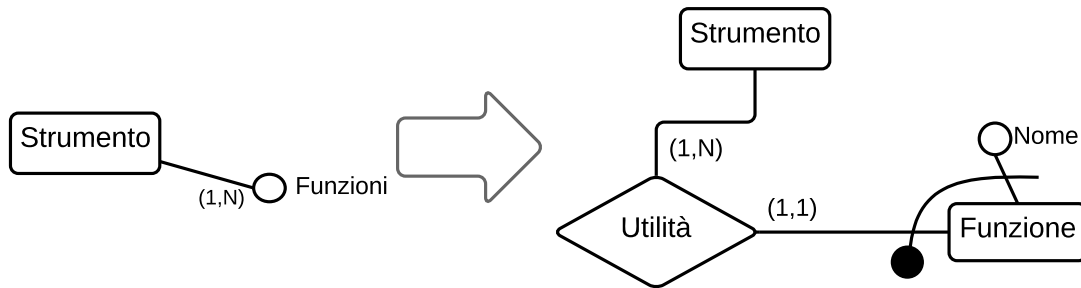


Nel caso delle due generalizzazioni che coinvolgono le entità **Prenotazione**, **PrenotazioneTelefonica**, **PrenotazioneOnline** e **Allestimento** si può procedere con due accorpamenti sul padre: prima l'accorpamento di **Allestimento** su **PrenotazioneOnline**; poi l'accorpamento di **PrenotazioneTelefonica** e **PrenotazioneOnline** su **Prenotazione**. Tutti gli attributi di **PrenotazioneTelefonica** e **Allestimento** diventano opzionali, così anche le associazioni **InvioPre** e **SerataTema**. Non è necessario inserire qui un attributo **Tipo** per riconoscere il tipo di prenotazione in quanto può essere facilmente dedotto dalla presenza o meno delle associazioni **InvioPre**, **SerataTema** e **Riserva**.



3.2 Eliminazione di attributi multivalore

L'unico *attributo multivalore* presente nel diagramma è l'attributo **Funzione** di **Strumento** con cardinalità $(1,N)$. Questo attributo può essere facilmente eliminato aggiungendo una nuova entità **Funzione** legata a **Strumento** tramite un'associazione *uno-a-molti* come mostrato in figura:



3.3 Accorpamenti

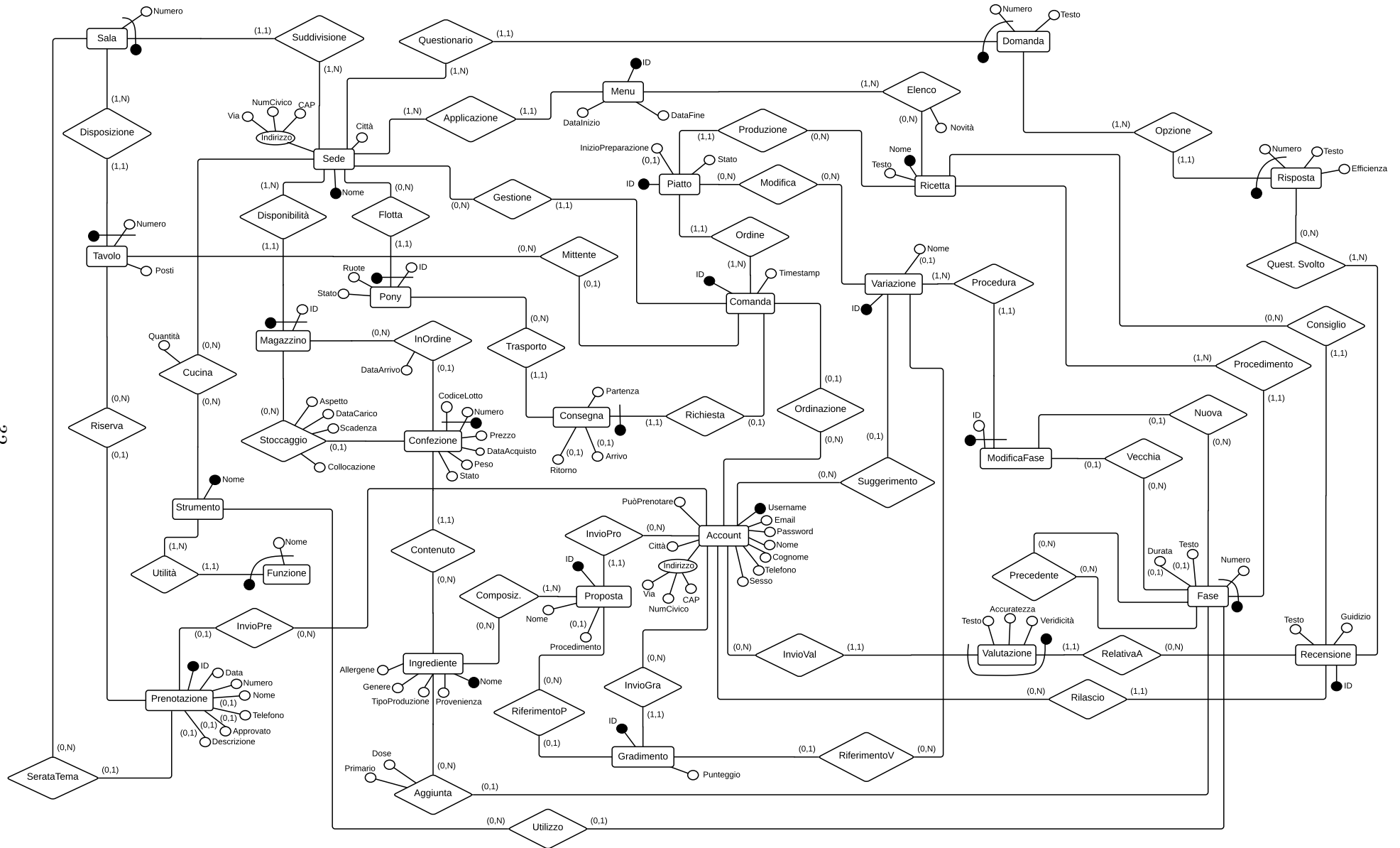
Possiamo effettuare altri accorpamenti al diagramma.

L'entità **Questionario** e l'associazione **Relazione** che la collega all'entità **Sede**, possono essere accorpate in un'unica associazione **Questionario** che lega **Sede** a **Domanda** in quanto un questionario è identificato dalla sede alla quale appartiene e non ha altri attributi.

Può essere fatto un ragionamento analogo al precedente anche per l'entità **QuestionarioSvolto** e l'associazione **Appartenenza** che la collega all'entità **Recensione**. Viene quindi rimossa anche l'associazione **IstanzaDi**.

3.4 Diagramma E-R ristrutturato

Il *diagramma ristrutturato* finale è mostrato nella pagina successiva.



Capitolo 4

Progettazione logica

In questo Capitolo viene presentato il risultato della fase di *progettazione logica*. Il paragrafo 4.1 riporta lo *schema logico* prodotto. Il paragrafo 4.2 a pagina 35 elenca le *chiavi esterne* individuate. Il paragrafo 4.3 a pagina 36 contiene le *implementazioni MySQL* dei *trigger* necessari per gestire i *vincoli di integrità generici*.

4.1 Schema logico

Di seguito è mostrato lo *schema logico* risultante dalla fase di *progettazione logica*.

Si noti che le entità **Comanda** e **Consegna** potevano venir anche accorpate in un'unica entità. Abbiamo però deciso di lasciarle separate così da evitare valori NULL in più (nel caso di comanda da tavolo).

L'associazione **Precedente** poteva anche essere tradotta con una tabella che riportasse la fase *successiva* (invece della precedente). Non c'è alcuna differenza tra l'una e l'altra rappresentazione — decidiamo quindi di rappresentare la fase *precedente* così da rimanere fedeli al nome dell'associazione.

Per tutte le *associazioni uno-a-molti con partecipazione opzionale*, ossia del tipo (0,1) -- (X,N) (ad esempio: **Utilizzo**, **Aggiunta**, **Riserva**, **Mittente**, ecc...) si è sempre deciso di **non** creare una tabella aggiuntiva per l'associazione e di inserire invece attributi opzionali. Questa scelta aumenta il numero di valori NULL ma riduce il numero di associazioni, e quindi anche il numero di accessi per le operazioni che coinvolgono tali tabelle. Cerchiamo così di ottimizzare le prestazioni della base di dati.

Per il resto, la traduzione dal *diagramma entità-relazione ristrutturato* di pagina 32 allo *schema logico* mostrato in questo paragrafo è del tutto ovvia e immediata. Non sono quindi necessarie ulteriori spiegazioni.

Le *chiavi primarie* sono mostrate sottolineate mentre gli *attributi opzionali* sono rappresentati in *corsivo*.

SEDE(Nome, Città, CAP, Via, NumeroCivico)
MAGAZZINO(Sede, ID)
CONFEZIONE(CodiceLotto, Numero, Ingrediente, Peso, Prezzo, DataAcquisto,
DataArrivo, *DataCarico*, Sede, Magazzino, *Collocazione*, *Scadenza*,
Aspetto, Stato)
INGREDIENTE(Nome, Provenienza, TipoProduzione, Genere, Allergene)
CUCINA(Sede, Strumento, Quantità)
STRUMENTO(Nome)
FUNZIONE(Strumento, Nome)
MENU(ID, Sede, DataInizio, DataFine)
ELENCO(Menu, Ricetta, Novità)
RICETTA(Nome, Testo)
FASE(Ricetta, Numero, *Ingrediente*, *Dose*, *Primario*, *Strumento*, *Testo*,
Durata)
SEQUENZAFASI(Ricetta, Fase, FasePrecedente)
PIATTO(ID, Comanda, Ricetta, *InizioPreparazione*, Stato)
MODIFICA(Piatto, Variazione)
VARIAZIONE(ID, Nome, Account)
MODIFICAFASE(Variazione, ID, Ricetta, *FaseVecchia*, *FaseNuova*)
COMANDA(ID, Timestamp, Sede, *Sala*, *Tavolo*, Account)
TAVOLO(Sede, Sala, Numero, Posti)
SALA(Sede, Numero)
CONSEGNA(Comanda, Sede, Pony, *Partenza*, *Arrivo*, *Ritorno*)
PONY(Sede, ID, Ruote, Stato)
PRENOTAZIONE(ID, Sede, Data, Numero, Account, Nome, Telefono, Sala,
Tavolo, *Descrizione*, *Approvato*)
ACCOUNT(Username, Email, Password, Nome, Cognome, Sesso, Città, CAP,
Via, NumeroCivico, Telefono, PuòPrenotare)
PROPOSTA(ID, Account, Nome, *Procedimento*)
COMPOSIZIONE(Proposta, Ingrediente)
GRADIMENTO(ID, Account, *Proposta*, *Suggerimento*, Punteggio)
RECENSIONE(ID, Account, Ricetta, Testo, Giudizio)
VALUTAZIONE(Account, Recensione, Veridicità, Accuratezza, Testo)
DOMANDA(Sede, Numero, Testo)
RISPOSTA(Sede, Domanda, Numero, Testo, Efficienza)
QUESTIONARIOSVOLTO(Recensione, Sede, Domanda, Risposta)

4.2 Vincoli di integrità referenziale

Di seguito riportiamo tutti i *vincoli di integrità referenziale* derivati dalla traduzione delle associazioni nello schema logico.

$R1(A) \rightarrow R2(B)$ indica che l'attributo **A** della relazione **R1** è **chiave esterna** dell'attributo **B** della relazione **R2** — ossia che l'attributo **A** può assumere solo uno dei valori assunti dall'attributo **B** (e il valore **NULL** se l'attributo è opzionale). Talvolta **A** e **B** possono essere anche due *insiemi di attributi* con lo stesso numero di elementi, ognuno dei quali separati da virgola.

Nella totalità dei casi, l'attributo (o l'insieme di attributi) **B** è la **chiave primaria** della relazione **R2**.

- $MAGAZZINO(Sede) \rightarrow SEDE(Nome)$
- $CONFEZIONE(Sede, Magazzino) \rightarrow MAGAZZINO(Sede, ID)$
- $CONFEZIONE(Ingrediente) \rightarrow INGREDIENTE(Nome)$
- $CUCINA(Sede) \rightarrow SEDE(Nome)$
- $CUCINA(Strumento) \rightarrow STRUMENTO(Nome)$
- $FUNZIONE(Strumento) \rightarrow STRUMENTO(Nome)$
- $MENU(Sede) \rightarrow SEDE(Nome)$
- $ELENCO(Menu) \rightarrow MENU(ID)$
- $ELENCO(Ricetta) \rightarrow RICETTA(Nome)$
- $FASE(Ricetta) \rightarrow RICETTA(Nome)$
- $FASE(Ingrediente) \rightarrow INGREDIENTE(Nome)$
- $FASE(Strumento) \rightarrow STRUMENTO(Nome)$
- $SEQUENZAFASI(Ricetta, Fase) \rightarrow FASE(Ricetta, Numero)$
- $SEQUENZAFASI(Ricetta, FasePrecedente) \rightarrow FASE(Ricetta, Numero)$
- $PIATTO(Comanda) \rightarrow COMANDA(ID)$
- $PIATTO(Ricetta) \rightarrow RICETTA(Nome)$
- $MODIFICA(Piatto) \rightarrow PIATTO(ID)$
- $MODIFICA(Variazione) \rightarrow VARIAZIONE(ID)$
- $VARIAZIONE(Account) \rightarrow ACCOUNT(Username)$
- $MODIFICAFASE(Variazione) \rightarrow VARIAZIONE(ID)$
- $MODIFICAFASE(Ricetta, FaseVecchia) \rightarrow FASE(Ricetta, Numero)$
- $MODIFICAFASE(Ricetta, FaseNuova) \rightarrow FASE(Ricetta, Numero)$

- `COMANDA(Sede) → SEDE(Nome)`
- `COMANDA(Sede, Sala, Tavolo) → TAVOLO(Sede, Sala, Numero)`
- `COMANDA(Account) → ACCOUNT Username)`
- `TAVOLO(Sede, Sala) → SALA(Sede, Numero)`
- `SALA(Sede) → SEDE(Nome)`
- `CONSEGNA(Comanda) → COMANDA(ID)`
- `CONSEGNA(Sede, Pony) → PONY(Sede, ID)`
- `PONY(Sede) → SEDE(Nome)`
- `PRENOTAZIONE(Sede, Sala, Tavolo) → TAVOLO(Sede, Sala, Numero)`
- `PRENOTAZIONE(Sede, Sala) → SALA(Sede, Numero)`
- `PRENOTAZIONE(Account) → ACCOUNT Username)`
- `PROPOSTA(Account) → ACCOUNT Username)`
- `COMPOSIZIONE(Proposta) → PROPOSTA(ID)`
- `COMPOSIZIONE(Ingrediente) → INGREDIENTE(Nome)`
- `GRADIMENTO(Account) → ACCOUNT Username)`
- `GRADIMENTO(Proposta) → PROPOSTA(ID)`
- `GRADIMENTO(Suggerimento) → VARIAZIONE(ID)`
- `RECENSIONE(Account) → ACCOUNT Username)`
- `RECENSIONE(Ricetta) → RICETTA(Nome)`
- `VALUTAZIONE(Account) → ACCOUNT Username)`
- `VALUTAZIONE(Recensione) → RECENSIONE(ID)`
- `DOMANDA(Sede) → SEDE(Nome)`
- `RISPOSTA(Sede, Domanda) → DOMANDA(Sede, Numero)`
- `QUESTIONARIOSVOLTO(Recensione) → RECENSIONE(ID)`
- `QUESTIONARIOSVOLTO(Sede, Domanda, Risposta)
→ RISPOSTA(Sede, Domanda, Numero)`

4.3 Vincoli di integrità generici con MySQL

Il seguente listato contiene i *trigger* necessari a effettuare tutti i controlli di integrità generici per le tabelle del database. Le descrizioni dei trigger sono inserite in blocchi di commento sopra al trigger al quale si riferiscono.

```

DELIMITER $$

/*****
 * nuova_confezione controlla che DataCarico, se presente, non sia precedente *
 * a DataAcquisto. Inoltre controlla che siano specificati tutti gli      *
 * attributi necessari.                                                    *
 *****/
CREATE TRIGGER nuova_confezione
BEFORE INSERT
ON Confezione
FOR EACH ROW
BEGIN
    IF NEW.DataCarico IS NOT NULL THEN
        IF NEW.DataCarico < NEW.DataAcquisto THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'DataCarico precedente a DataAcquisto.';
        END IF;
        IF NEW.Collocazione IS NULL THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'L\'attributo Collocazione non può essere NULL.';
        END IF;
        IF NEW.Aspetto IS NULL THEN
            SET NEW.Aspetto = TRUE; -- Default: nessun danno
        END IF;
        IF NEW.Stato IS NULL THEN
            SET NEW.Stato = 'completa'; -- Default
        END IF;
    ELSE
        IF (NEW.Stato <> 'in ordine') THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'L\'attributo DataCarico non può essere NULL.';
        ELSEIF (NEW.Collocazione IS NOT NULL
            OR NEW.Aspetto IS NOT NULL) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'DataCarico, Collocazione e Aspetto devono '
                               'essere tutti NULL o tutti non-NULL.';
        END IF;
    END IF;
END;

/*****
 * aggiorna_confezione controlla che DataCarico, se presente, non sia     *
 * precedente a DataAcquisto. Inoltre controlla che siano specificati tutti *
 * gli attributi necessari.                                                *
 *****/
CREATE TRIGGER aggiorna_confezione

```

```

BEFORE UPDATE
ON Confezione
FOR EACH ROW
50 BEGIN
    IF NEW.DataCarico IS NOT NULL THEN
        IF NEW.DataCarico < NEW.DataAcquisto THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'DataCarico precedente a DataAcquisto.';
55     END IF;
        IF NEW.Collocazione IS NULL THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'L\'attributo Collocazione non può essere NULL.';
60     END IF;
        IF NEW.Aspetto IS NULL THEN
            SET NEW.Aspetto = TRUE; -- Default: nessun danno
        END IF;
    ELSE
        IF (NEW.Stato <> 'in ordine') THEN
65         SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'L\'attributo DataCarico non può essere NULL.';
        ELSEIF (NEW.Collocazione IS NOT NULL
            OR NEW.Aspetto IS NOT NULL) THEN
70         SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'DataCarico, Collocazione e Aspetto devono '
                                'essere tutti NULL o tutti non-NULL.';
        END IF;
    END IF;
END; $$
75
/*****
* nuovo_menu controlla che il periodo di applicazione del nuovo menu inserito *
* non sia in conflitto con il periodo di applicazione di un altro menu nella *
* stessa sede (ogni sede applica un solo menu alla volta). Inoltre controlla *
80 * che DataFine sia successiva a DataInizio. *
* Business Rule: (BR05) *
*****/
CREATE TRIGGER nuovo_menu
BEFORE INSERT
85 ON Menu
FOR EACH ROW
BEGIN
    DECLARE MenuAttiviPeriodo BOOL;

90     IF NEW.DataFine <= NEW.DataInizio THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'DataFine precedente a DataInizio.';

```

```

END IF;

95 SET MenuAttiviPeriodo = (SELECT COUNT(*) > 0
                             FROM Menu M
                             WHERE M.Sede = NEW.Sede
                                    AND M.DataFine >= NEW.DataInizio
                                    AND M.DataInizio <= NEW.DataFine);

100 IF MenuAttiviPeriodo THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Un menu è già attivo in questo periodo.';
END IF;

105 END;$$

/*****
 * nuova_fase controlla che la fase inserita sia una FaseIngrediente o una
 * FaseManovra (non entrambe insieme). Controlla anche che siano specificati
110 * tutti gli attributi necessari.
 *****/
CREATE TRIGGER nuova_fase
BEFORE INSERT
ON Fase
115 FOR EACH ROW
BEGIN
    IF NEW.Ingrediente IS NOT NULL THEN
        SET NEW.Durata = NULL;
        SET NEW.Testo = NULL;
120 IF NEW.Strumento IS NOT NULL THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Una fase può impiegare o uno strumento o un '
                                'ingrediente. Non entrambi.';
        END IF;
125 IF NEW.Dose IS NULL THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'L\'attributo Dose deve essere specificato.';
        END IF;
        IF NEW.Primario IS NULL THEN
130 SET NEW.Primario = FALSE; -- Default
        END IF;
    ELSE
        SET NEW.Dose = NULL;
        SET NEW.Primario = NULL;

135 IF NEW.Durata IS NULL THEN
            SET NEW.Durata = '00:00:00'; -- Default
        END IF;
    END IF;
END IF;

```

```

140         IF NEW.Testo IS NULL THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'L\'attributo Testo deve essere specificato.';
        END IF;
    END IF;
END; $$

145 /*****
* aggiorna_fase controlla che la fase modificata sia sempre una
* FaseIngrediente o una FaseManovra (non entrambe insieme). Controlla anche
* che siano specificati tutti gli attributi necessari.
150 *****/
CREATE TRIGGER aggiorna_fase
BEFORE UPDATE
ON Fase
FOR EACH ROW
155 BEGIN
    IF NEW.Ingrediente IS NOT NULL THEN
        SET NEW.Durata = NULL;
        SET NEW.Testo = NULL;
        IF NEW.Strumento IS NOT NULL THEN
160             SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Una fase può impiegare o uno strumento o un '
                                'ingrediente. Non entrambi.';
        END IF;
        IF NEW.Dose IS NULL THEN
165             SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'L\'attributo Dose deve essere specificato.';
        END IF;
        IF NEW.Primario IS NULL THEN
            SET NEW.Primario = FALSE; -- Default
170        END IF;
    ELSE
        SET NEW.Dose = NULL;
        SET NEW.Primario = NULL;

175        IF NEW.Durata = NULL THEN
            SET NEW.Durata = '00:00:00'; -- Default
        END IF;
        IF NEW.Testo = NULL THEN
180             SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'L\'attributo Testo deve essere '
                                'specificato.';
        END IF;
    END IF;
END; $$

```



```

185  /*****
      * nuova_sequenza_fasi controlla che le due fasi messe in sequenza      *
      * appartengano alla stessa ricetta.                                    *
      *****/
190  CREATE TRIGGER nuova_sequenza_fasi
      BEFORE INSERT
      ON SequenzaFasi
      FOR EACH ROW
      BEGIN
195      DECLARE StessaRicetta BOOL;
      SET StessaRicetta = (SELECT COUNT(*) > 0
                           FROM (SELECT F1.ID, F1.Ricetta
                                FROM Fase F1
                                WHERE F1.ID = NEW.Fase) AS Fase1
                           INNER JOIN
                                (SELECT F2.ID, F2.Ricetta
                                 FROM Fase F2
                                 WHERE F2.ID = NEW.FasePrecedente) AS Fase2
                           ON Fase1.Ricetta = Fase2.Ricetta);

200
205      IF NOT StessaRicetta THEN
          SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'Le due fasi non appartengono alla stessa ricetta.';
      END IF;
210  END;$$

      /*****
      * aggiorna_sequenza_fasi controlla che le due fasi messe in sequenza      *
      * appartengano alla stessa ricetta.                                    *
      *****/
215  CREATE TRIGGER aggiorna_sequenza_fasi
      BEFORE UPDATE
      ON SequenzaFasi
      FOR EACH ROW
      BEGIN
220      DECLARE StessaRicetta BOOL;
      SET StessaRicetta = (SELECT COUNT(*) > 0
                           FROM (SELECT F1.ID, F1.Ricetta
                                FROM Fase F1
                                WHERE F1.ID = NEW.Fase) AS Fase1
                           INNER JOIN
                                (SELECT F2.ID, F2.Ricetta
                                 FROM Fase F2
                                 WHERE F2.ID = NEW.FasePrecedente) AS Fase2
                           ON Fase1.Ricetta = Fase2.Ricetta);

225
230

```

```

235     IF NOT StessaRicetta THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Le due fasi non appartengono alla stessa ricetta.';
    END IF;
END; $$

/*****
* nuova_comanda controlla che la comanda inserita sia da tavolo o take-away
* e non entrambe insieme.
*****/
CREATE TRIGGER nuova_comanda
BEFORE INSERT
ON Comanda
245 FOR EACH ROW
BEGIN
    IF (NEW.Account IS NOT NULL
        AND (NEW.Tavolo IS NOT NULL OR NEW.Sala IS NOT NULL))
    OR (NEW.Account IS NULL AND NEW.Tavolo IS NULL
250         AND NEW.Sala IS NULL) THEN

        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Una comanda deve essere o da tavolo o take-away. '
                           'Non entrambe.';

    END IF;
255 END; $$

/*****
* aggiorna_comanda controlla che la comanda modificata sia da tavolo o
* take-away e non entrambe insieme.
*****/
260 CREATE TRIGGER aggiorna_comanda
BEFORE UPDATE
ON Comanda
FOR EACH ROW
265 BEGIN
    IF (NEW.Account IS NOT NULL
        AND (NEW.Tavolo IS NOT NULL OR NEW.Sala IS NOT NULL))
    OR (NEW.Account IS NULL AND NEW.Tavolo IS NULL
        AND NEW.Sala IS NULL) THEN
270
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Una comanda deve essere o da tavolo o take-away. '
                           'Non entrambe.';

    END IF;
275 END; $$

```

```

/*****
 * aggiorna_piatto evita che l'attributo venga aggiornato al timestamp
 * attuale se il piatto non sta passando dallo stato 'attesa' a 'in
 * preparazione'.
 *****/
280 CREATE TRIGGER aggiorna_piatto
    BEFORE UPDATE
    ON Piatto
    FOR EACH ROW
285 BEGIN
    IF NEW.Stato != 'in preparazione' OR OLD.Stato != 'attesa' THEN
        NEW.Stato = OLD.Stato;
    END IF;
END;$$

/*****
 * nuova_variazione controlla che la variazione inserita sia un Suggerimento
 * o una VariazionePiatto (non entrambi insieme).
 *****/
290
295 CREATE TRIGGER nuova_variazione
    BEFORE INSERT
    ON Variazione
    FOR EACH ROW
    BEGIN
300     IF (NEW.Nome IS NOT NULL AND NEW.Account IS NOT NULL)
        OR (NEW.Nome IS NULL AND NEW.Account IS NULL) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Una variazione deve essere una variazione '
                                'dagli chef (con nome) o un suggerimento inviato '
                                'da un account utente (senza nome).';
305     END IF;
END;$$

/*****
 * aggiorna_variazione controlla che la variazione modificata sia un
 * Suggerimento o una VariazionePiatto (non entrambi insieme).
 *****/
310
CREATE TRIGGER aggiorna_variazione
    BEFORE UPDATE
    ON Variazione
315 FOR EACH ROW
    BEGIN
        IF (NEW.Nome IS NOT NULL AND NEW.Account IS NOT NULL)
            OR (NEW.Nome IS NULL AND NEW.Account IS NULL) THEN
320             SIGNAL SQLSTATE '45000'

```

```

        SET MESSAGE_TEXT = 'Una variazione deve essere una variazione '
                           'dagli chef (con nome) o un suggerimento inviato '
                           'da un account utente (senza nome).';

    END IF;
325 END; $$

/*****
 * nuova_modificafase che la ModificaFase contenga almeno o una FaseVecchia o *
 * una FaseNuova. Inoltre controlla che le fasi modificate appartengano alla *
330 * stessa ricetta (ossia quella a cui appartiene la variazione). Infine      *
 * controlla che una fase inserita in FaseNuova (FaseVecchia) non compaia in *
 * nessuna FaseVecchia (FaseNuova).                                         *
 *****/
CREATE TRIGGER nuova_modificafase
335 BEFORE INSERT
    ON ModificaFase
    FOR EACH ROW
    BEGIN
        DECLARE RicettaFaseNuova VARCHAR(45);
        DECLARE RicettaFaseVecchia VARCHAR(45);
340 DECLARE RicettaVariazione VARCHAR(45);
        DECLARE FaseInAggiunta BOOL;
        DECLARE FaseInEliminazione BOOL;

        IF NEW.FaseNuova IS NULL AND NEW.FaseVecchia IS NULL THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'FaseNuova o FaseVecchia devono essere specificati.';
        END IF;

350 SET RicettaVariazione = (SELECT V.Ricetta FROM Variazione V
                           WHERE V.ID = NEW.Variazione);

        IF NEW.FaseNuova IS NOT NULL THEN
            SET RicettaFaseNuova = (SELECT F.Ricetta FROM Fase F
355                                WHERE F.ID = NEW.FaseNuova);

            IF RicettaFaseNuova <> RicettaVariazione THEN
                SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'La ricetta di FaseNuova deve corrispondere '
360                                'alla ricetta della variazione.';
            END IF;

            SET FaseInEliminazione = (SELECT COUNT(*) > 0
365                                FROM ModificaFase MF
                                WHERE MF.FaseVecchia = NEW.FaseNuova);

```

```

370     IF FaseInEliminazione THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La fase inserita come FaseNuova viene '
                                'già eliminata da un\'altra ModificaFase. Una '
                                'fase può essere solo aggiunta o rimossa dalle '
                                'ModificaFase.';
    END IF;
END IF;
375
IF NEW.FaseVecchia IS NOT NULL THEN
    SET RicettaFaseVecchia = (SELECT F.Ricetta FROM Fase F
                                WHERE F.ID = NEW.FaseVecchia);

380    IF RicettaFaseVecchia <> RicettaVariazione THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La ricetta di FaseVecchia deve corrispondere '
                                'alla ricetta della variazione.';
    ELSEIF RicettaFaseNuova IS NOT NULL
385        AND RicettaFaseNuova <> RicettaFaseVecchia THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La ricetta di FaseVecchia e quella di '
                                'FaseNuova devono corrispondere.';
    END IF;
390
    SET FaseInAggiunta = (SELECT COUNT(*) > 0
                            FROM ModificaFase MF
                            WHERE MF.FaseNuova = NEW.FaseVecchia);

395    IF FaseInAggiunta THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La fase inserita come FaseVecchia viene '
                                'già aggiunta da un\'altra ModificaFase. Una '
                                'fase può essere solo aggiunta o rimossa dalle '
400        'ModificaFase.';
    END IF;
END IF;
END;$$

405 /*****
* aggiorna_modificafase che la ModificaFase contenga almeno o una *
* FaseVecchia o una FaseNuova. Inoltre controlla che le fasi modificate *
* appartengano alla stessa ricetta (ossia quella a cui appartiene la *
* variazione). *
410 *****/
CREATE TRIGGER aggiorna_modificafase
BEFORE UPDATE

```

```

ON ModificaFase
FOR EACH ROW
415 BEGIN
    DECLARE RicettaFaseNuova VARCHAR(45);
    DECLARE RicettaFaseVecchia VARCHAR(45);
    DECLARE RicettaVariazione VARCHAR(45);
    DECLARE FaseInAggiunta BOOL;
420 DECLARE FaseInEliminazione BOOL;

    IF NEW.FaseNuova IS NULL AND NEW.FaseVecchia IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'FaseNuova o FaseVecchia devono essere specificati.';
425 END IF;

    SET RicettaVariazione = (SELECT V.Ricetta FROM Variazione V
                             WHERE V.ID = NEW.Variazione);

430 IF NEW.FaseNuova IS NOT NULL THEN
    SET RicettaFaseNuova = (SELECT F.Ricetta FROM Fase F
                             WHERE F.ID = NEW.FaseNuova);

    IF RicettaFaseNuova <> RicettaVariazione THEN
435 SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La ricetta di FaseNuova deve corrispondere '
                           'alla ricetta della variazione.';
    END IF;

440 SET FaseInEliminazione = (SELECT COUNT(*) > 0
                             FROM ModificaFase MF
                             WHERE MF.FaseVecchia = NEW.FaseNuova);

    IF FaseInEliminazione THEN
445 SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La fase inserita come FaseNuova viene '
                           'già eliminata da un\'altra ModificaFase. Una '
                           'fase può essere solo aggiunta o rimossa dalle '
                           'ModificaFase.';
    END IF;
450 END IF;
END IF;

IF NEW.FaseVecchia IS NOT NULL THEN
    SET RicettaFaseVecchia = (SELECT F.Ricetta FROM Fase F
                              WHERE F.ID = NEW.FaseVecchia);
455 IF RicettaFaseVecchia <> RicettaVariazione THEN
        SIGNAL SQLSTATE '45000'

```

```

460         SET MESSAGE_TEXT = 'La ricetta di FaseVecchia deve corrispondere '
                                'alla ricetta della variazione.';
ELSEIF RicettaFaseNuova IS NOT NULL
    AND RicettaFaseNuova <> RicettaFaseVecchia THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'La ricetta di FaseVecchia e quella di '
465         'FaseNuova devono corrispondere.';
END IF;

SET FaseInAggiunta = (SELECT COUNT(*) > 0
                      FROM ModificaFase MF
470                      WHERE MF.FaseNuova = NEW.FaseVecchia);

IF FaseInAggiunta THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'La fase inserita come FaseVecchia viene '
475         'già aggiunta da un\'altra ModificaFase. Una '
        'fase può essere solo aggiunta o rimossa dalle '
        'ModificaFase.';
END IF;
END IF;
480 END;$$

/*****
 * nuova_modifica controlla che, per il piatto sul quale si sta applicando la *
 * variazione, non siano già state scelte 3 variazioni. Inoltre controlla che *
485 * la ricetta del piatto e quella della variazione corrispondano.          *
 * Business Rule: (BR12)                                                    *
 *****/
CREATE TRIGGER nuova_modifica
BEFORE INSERT
490 ON Modifica
FOR EACH ROW
BEGIN
    DECLARE Suggerimento BOOL;
    DECLARE NumVariazioni INT;
    DECLARE StessaRicetta BOOL;
495

    SET Suggerimento = (SELECT V.Account IS NOT NULL
                        FROM Variazione V
                        WHERE V.ID = NEW.Variazione);

500
    IF Suggerimento THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Solo le variazioni selezionate dagli chef possono '
                            'essere selezionate, non i suggerimenti.';

```

```

505     END IF;

    SET NumVariazioni = (SELECT COUNT(*)
                        FROM Modifica M
                        WHERE M.Piatto = NEW.Piatto);

510     IF NumVariazioni >= 3 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Ci sono già 3 variazioni su questo piatto.';
    END IF;

515     SET StessaRicetta = (SELECT COUNT(*) > 0
                        FROM (SELECT P.ID, P.Ricetta
                            FROM Piatto P
                            WHERE P.ID = NEW.Piatto) AS Pi
                        INNER JOIN
                        (SELECT V.ID, V.Ricetta
                            FROM Variazione V
                            WHERE V.ID = NEW.Variazione) AS Va
                        ON Pi.Ricetta = Va.Ricetta);

520

525     IF NOT StessaRicetta THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La variazione selezionata non è applicabile al '
                        'piatto scelto.';
    END IF;

530     END; $$

/*****
* aggiorna_consegna si assicura che l'arrivo registrato sia sempre successivo*
* alla partenza e che il ritorno sia sempre successivo all'arrivo.          *
*****/
535 CREATE TRIGGER aggiorna_consegna
BEFORE UPDATE
ON Consegna
540 FOR EACH ROW
BEGIN
    IF NEW.Arrivo IS NOT NULL AND NEW.Arrivo < NEW.Partenza THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Arrivo precedente a partenza.';
    ELSEIF NEW.Arrivo IS NULL AND NEW.Ritorno IS NOT NULL THEN
545        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Se Ritorno viene specificato anche Arrivo deve '
                        'essere specificato.';
    END IF;

550

```



```

555     IF NEW.Ritorno IS NOT NULL THEN
        IF OLD.Arrivo IS NOT NULL AND OLD.Arrivo = NEW.Arrivo THEN
            -- Evita ON UPDATE CURRENT_TIMESTAMP
            SET NEW.Arrivo = OLD.Arrivo;
        END IF;
        IF NEW.Ritorno < NEW.Arrivo THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Ritorno precedente a arrivo.';
560     END IF;
    END IF;
END;$$

/*****
565 * nuova_prenotazione controlla: se l'account (in caso di prenotazione
* online) è abilitato a prenotare; se il tavolo da prenotare è libero; se la
* sala e tutti i tavoli che contiene sono liberi per un allestimento.
* Inoltre controlla che siano specificati tutti gli attributi necessari.
* Business Rule: (BR15), (BR16) e (BR19)
570 *****/
CREATE TRIGGER nuova_prenotazione
BEFORE INSERT
ON Prenotazione
FOR EACH ROW
575 BEGIN
    DECLARE PrenotazioniAbilitate BOOL;
    DECLARE PostiTavolo INT;
    DECLARE TavoloLibero BOOL;
    DECLARE SalaLibera BOOL;

580
    IF CURRENT_DATETIME > (NEW.'Data' - INTERVAL 1 DAY) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Una prenotazione deve essere effettuata almeno un '
        'giorno prima della data scelta.';
585    END IF;

    IF NEW.Tavolo IS NOT NULL THEN
        SET PostiTavolo = (SELECT T.Posti
                           FROM Tavolo T
590                          WHERE T.ID = NEW.Tavolo
                           AND T.Sala = NEW.Sala
                           AND T.Sede = NEW.Sede);
    END IF;

595    IF NEW.Account IS NOT NULL THEN
        SET PrenotazioniAbilitate = (SELECT A.PuoPrenotare

```

```

                                FROM Account A
                                WHERE A.Username = NEW.Account);

600 IF NOT PrenotazioniAbilitate THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Prenotazioni disabilitate per l\'account.';
    END IF;

605 SET NEW.Telefono = NULL;
    IF NEW.Tavolo IS NOT NULL THEN
        SET NEW.Nome = NULL;
        SET NEW.Descrizione = NULL;
        SET NEW.Approvato = NULL;

610 IF PostiTavolo > NEW.Numero + 3 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Il tavolo scelto non ha un numero adeguato '
                            'di posti.';

615 END IF;
    ELSEIF NEW.Nome IS NULL THEN
        SET NEW.Descrizione = NULL;
        SET NEW.Approvato = NULL;

620 -- Assegna Tavolo (riportato solo nello script finale nel Cap. 11)

    ELSEIF NEW.Descrizione IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Descrizione deve essere specificato per gli '
                            'allestimenti.';

625 ELSEIF NEW.Approvato IS NULL THEN
        SET NEW.Approvato = FALSE; -- Default
    END IF;
ELSE
630 SET NEW.Descrizione = NULL;
    SET NEW.Approvato = NULL;
    IF NEW.Nome IS NULL OR NEW.Telefono IS NULL OR NEW.Tavolo IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Nome, Telefono e Tavolo devono essere '
                            'specificati per le prenotazioni telefoniche.';

635 END IF;
END IF;

-- NOTA: L'uso di espressioni booleane all'interno di SUM() è possibile solo in
640 -- MySQL (che converte l'espressione booleana in int). In altri DBMS si
-- può utilizzare COUNT() e spostare l'espressione booleana nel WHERE.
SET SalaLibera = (SELECT SUM(DATE(P.'Data') = DATE(NEW.'Data')) > 0

```

```

        FROM Prenotazione P
        WHERE P.Sala = NEW.Sala
        AND P.Sede = NEW.Sede
        AND P.Tavolo = NULL);

645
    IF SalaLibera THEN
        SIGNAL SQLSTATE '45000'
650        SET MESSAGE_TEXT = 'La sala scelta è già prenotata per un '
            'allestimento.';
    END IF;

    IF NEW.Tavolo IS NOT NULL THEN
655
        IF PostiTavolo < NEW.Numero THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il tavolo scelto non ha un numero adeguato '
                'di posti.';
660
        END IF;

        SET TavoloLibero = (SELECT SUM(P.'Data' >
                                (NEW.'Data' - INTERVAL 2 HOUR)
                                AND P.'Data' <
                                (NEW.'Data' + INTERVAL 2 HOUR)) = 0
                                FROM Prenotazione P
                                WHERE P.Tavolo = NEW.Tavolo
                                AND P.Sala = NEW.Sala
                                AND P.Sede = NEW.Sede);
665

        IF NOT TavoloLibero THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il tavolo scelto è già prenotato.';
        END IF;
670
    ELSE
675
        SET SalaLibera = (SELECT SUM(DATE(P.'Data') = DATE(NEW.'Data')) = 0
                                FROM Prenotazione P
                                WHERE P.Sala = NEW.Sala
                                AND P.Sede = NEW.Sede);
680

        IF NOT SalaLibera THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'La sala contiene tavoli già prenotati.';
        END IF;
685
    END IF;
END; $$

/*****

```

```

690  * aggiorna_prenotazione controlla che la rettifica della prenotazione venga *
    * fatta al max. 48 ore prima della data della prenotazione. Inoltre *
    * che la nuova prenotazione specifichi tutti gli attributi necessari e *
    * che sia valida. *
    * Business Rule: (BR15), (BR16) e (BR17) *
    *****/
695 CREATE TRIGGER aggiorna_prenotazione
    BEFORE UPDATE
    ON Prenotazione
    FOR EACH ROW
    BEGIN
700     DECLARE PostiTavolo INT;
        DECLARE TavoloLibero BOOL;
        DECLARE SalaLibera BOOL;

        IF CURRENT_DATETIME > (OLD.'Data' - INTERVAL 2 DAY) THEN
705             SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Non è possibile modificare la prenotazione.';
        END IF;

        IF NEW.Tavolo IS NOT NULL THEN
710             SET PostiTavolo = (SELECT T.Posti
                                    FROM Tavolo T
                                    WHERE T.ID = NEW.Tavolo
                                           AND T.Sala = NEW.Sala
                                           AND T.Sede = NEW.Sede);
715         END IF;

        IF NEW.Account IS NOT NULL THEN
            SET NEW.Telefono = NULL;
            IF NEW.Tavolo IS NOT NULL THEN
720                 SET NEW.Nome = NULL;
                    SET NEW.Descrizione = NULL;
                    SET NEW.Approvato = NULL;

                    IF PostiTavolo > NEW.Numero + 3 THEN
725                         SIGNAL SQLSTATE '45000'
                                SET MESSAGE_TEXT = 'Il tavolo scelto non ha un numero adeguato '
                                                            'di posti.';
                    END IF;
                ELSEIF NEW.Nome IS NULL OR NEW.Descrizione IS NULL THEN
730                     SIGNAL SQLSTATE '45000'
                                SET MESSAGE_TEXT = 'Nome e Descrizione devono essere specificati '
                                                            'per gli allestimenti.';
                ELSEIF NEW.Approvato IS NULL THEN
                    SET NEW.Approvato = FALSE; -- Default

```

```

735     END IF;
ELSE
    SET NEW.Descrizione = NULL;
    SET NEW.Approvato = NULL;
    IF NEW.Nome IS NULL OR NEW.Telefono IS NULL OR NEW.Tavolo IS NULL THEN
740        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Nome, Telefono e Tavolo devono essere '
                            'specificati per le prenotazioni telefoniche.';
    END IF;
END IF;

745 SET SalaLibera = (SELECT SUM(DATE(P.'Data') = DATE(NEW.'Data')) > 0
                    FROM Prenotazione P
                    WHERE P.Sala = NEW.Sala
                          AND P.Sede = NEW.Sede
                          AND P.Tavolo = NULL);

    IF SalaLibera THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La sala scelta è già prenotata per un '
                            'allestimento.';
755    END IF;

    IF NEW.Tavolo IS NOT NULL THEN
        IF PostiTavolo < NEW.Numero THEN
760            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il tavolo scelto non ha un numero adeguato '
                                'di posti.';
        END IF;

765    SET TavoloLibero = (SELECT SUM(P.'Data' >
                                    (NEW.'Data' - INTERVAL 2 HOUR)
                                    AND P.'Data' <
                                    (NEW.'Data' + INTERVAL 2 HOUR)) = 0
                        FROM Prenotazione P
                        WHERE P.Tavolo = NEW.Tavolo
                              AND P.Sala = NEW.Sala
                              AND P.Sede = NEW.Sede);

770    IF NOT TavoloLibero THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Il tavolo scelto è già prenotato.';
775    END IF;
ELSE
    SET SalaLibera = (SELECT SUM(DATE(P.'Data') = DATE(NEW.'Data')) = 0
                    FROM Prenotazione P

```

```

                                WHERE P.Sala = NEW.Sala
                                AND P.Sede = NEW.Sede);

    IF NOT SalaLibera THEN
785        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La sala contiene tavoli già prenotati.';
    END IF;
END IF;
END; $$

790
/*****
* elimina_prenotazione controlla che l'annullamento della prenotazione
* venga fatta al max. 72 ore prima della data della prenotazione.
* Business Rule: (BR18)
795 *****/
CREATE TRIGGER elimina_prenotazione
BEFORE DELETE
ON Prenotazione
FOR EACH ROW
800 BEGIN
    IF CURRENT_DATETIME > (OLD.'Data' - INTERVAL 3 DAY) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Non è possibile annullare la prenotazione.';
    END IF;
805 END; $$

/*****
* I seguenti trigger controllano che gli attributi che indicano un punteggio
* siano compresi tra 1 e 5.
810 *****/
CREATE TRIGGER nuovo_gradimento
BEFORE INSERT
ON Gradimento
FOR EACH ROW
815 BEGIN
    IF (NEW.Proposta IS NOT NULL AND NEW.Suggerimento IS NOT NULL)
    OR (NEW.Proposta IS NULL AND NEW.Suggerimento IS NULL) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Un gradimento deve riferirsi a una proposta o a '
820                                'un suggerimento. Non ad entrambi.';
    END IF;

    IF NEW.Punteggio < 1 OR NEW.Punteggio > 5 THEN
825        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Punteggio deve essere compreso tra 1 e 5';
    END IF;
END;

```

```

      END IF;
END;$$

CREATE TRIGGER nuova_recensione
830 BEFORE INSERT
    ON Recensione
    FOR EACH ROW
    BEGIN
      IF NEW.Giudizio < 1 OR NEW.Giudizio > 5 THEN
835        SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'Giudizio deve essere compreso tra 1 e 5';
      END IF;
END;$$

840 CREATE TRIGGER nuova_valutazione
    BEFORE INSERT
    ON Valutazione
    FOR EACH ROW
    BEGIN
845      IF NEW.Veridicita < 1 OR NEW.Veridicita > 5 THEN
          SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'Veridicita deve essere compreso tra 1 e 5';
      END IF;
      IF NEW.Accuratezza < 1 OR NEW.Accuratezza > 5 THEN
850        SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'Accuratezza deve essere compreso tra 1 e 5';
      END IF;
END;$$

855 CREATE TRIGGER nuova_risposta
    BEFORE INSERT
    ON Risposta
    FOR EACH ROW
    BEGIN
860      IF NEW.Efficienza < 1 OR NEW.Efficienza > 5 THEN
          SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'Efficienza deve essere compreso tra 1 e 5';
      END IF;
END;$$

865 DELIMITER ;

```

I vincoli di integrità intrarelazionali semplici (UNIQUE, UNSIGNED, NOT NULL, ecc...) sono riportati solo nel Capitolo 11 a pagina 107.

Capitolo 5

Normalizzazione

In questo Capitolo viene presentato il risultato della fase di *normalizzazione*.

Nel paragrafo 5.1 è presentata la lista di tutte le *dipendenze funzionali non banali*. Nei paragrafi ad esso successivi viene riportata la normalizzazione di tutte le relazioni che non rispettano la **Forma Normale di Boyce-Codd (BCNF)**.

Vediamo ora tutte le *dipendenze funzionali non banali*.

5.1 Dipendenze funzionali

$$\text{Sede} \left\{ \begin{array}{l} \text{Nome} \rightarrow \text{Città, Via, NumeroCivico} \\ \text{Città, Via} \rightarrow \text{CAP} \end{array} \right.$$

Poiché (Città, Via) non è superchiave, Sede non è in BCNF.¹

$$\text{Magazzino} \left\{ \emptyset \right.$$

Poiché non ha alcuna dipendenza funzionale non banale, Magazzino è in BCNF.

$$\text{Confezione} \left\{ \begin{array}{l} \text{CodiceLotto} \rightarrow \text{Ingrediente, Scadenza} \\ \text{CodiceLotto, Numero} \rightarrow \text{Peso, Prezzo, DataAcquisto, DataArrivo,} \\ \text{DataCarico, Sede, Magazzino, Collocazione,} \\ \text{Aspetto, Stato} \end{array} \right.$$

Poiché CodiceLotto (da sé) non è superchiave, Confezione non è in BCNF.²

¹Si veda il paragrafo 5.2 a pagina 61.

²Si veda il paragrafo 5.3 a pagina 62.

$\text{Ingrediente} \left\{ \text{Nome} \rightarrow \text{Provienienza}, \text{TipoProduzione}, \text{Genere}, \text{Allergene} \right.$

Poiché la parte a sinistra è superchiave, **Ingrediente** è in BCNF.

$\text{Cucina} \left\{ \text{Sede}, \text{Strumento} \rightarrow \text{Quantità} \right.$

Poiché la parte a sinistra è superchiave, **Cucina** è in BCNF.

$\text{Strumento} \left\{ \emptyset \right.$

Poiché non ha alcuna dipendenza funzionale non banale, **Strumento** è in BCNF.

$\text{Funzione} \left\{ \emptyset \right.$

Poiché non ha alcuna dipendenza funzionale non banale, **Funzione** è in BCNF.

$\text{Menu} \left\{ \text{ID} \rightarrow \text{Sede}, \text{DataInizio}, \text{DataFine} \right.$

Poiché la parte a sinistra è superchiave, **Menu** è in BCNF.

$\text{Elenco} \left\{ \text{Menu}, \text{Ricetta} \rightarrow \text{Novità} \right.$

Poiché la parte a sinistra è superchiave, **Elenco** è in BCNF.

$\text{Ricetta} \left\{ \text{Nome} \rightarrow \text{Testo} \right.$

Poiché la parte a sinistra è superchiave, **Ricetta** è in BCNF.

$\text{Fase} \left\{ \begin{array}{l} \text{Ricetta}, \text{Numero} \rightarrow \text{Ingrediente}, \text{Strumento}, \text{Testo}, \text{Durata} \\ \text{Ricetta}, \text{Numero}, \text{Ingrediente} \rightarrow \text{Dose}, \text{Primario} \end{array} \right.$

Poiché la parte a sinistra è superchiave, **Fase** è in BCNF.

$\text{SequenzaFasi} \left\{ \emptyset \right.$

Poiché non ha alcuna dipendenza funzionale non banale, **SequenzaFasi** è in BCNF.

$\text{Piatto} \left\{ \text{ID} \rightarrow \text{Comanda}, \text{Ricetta}, \text{InizioPreparazione}, \text{Stato} \right.$

Poiché la parte a sinistra è superchiave, **Piatto** è in BCNF.

$\text{Modifica} \left\{ \emptyset \right.$

Poiché non ha alcuna dipendenza funzionale non banale, **Modifica** è in BCNF.

$\text{Variazione} \left\{ \text{ID} \rightarrow \text{Nome}, \text{Account} \right.$

Poiché la parte a sinistra è superchiave, **Variazione** è in BCNF.

$\text{ModificaFase} \left\{ \begin{array}{l} \text{Variazione} \rightarrow \text{Ricetta} \\ \text{Variazione}, \text{ID} \rightarrow \text{FaseVecchia}, \text{FaseNuova} \end{array} \right.$

Poiché **Variazione** (da sé) non è superchiave, **ModificaFase** non è in BCNF.³

$\text{Comanda} \left\{ \text{ID} \rightarrow \text{Timestamp}, \text{Sede}, \text{Sala}, \text{Tavolo}, \text{Account} \right.$

Poiché la parte a sinistra è superchiave, **Comanda** è in BCNF.

$\text{Tavolo} \left\{ \text{Sede}, \text{Sala}, \text{Numero} \rightarrow \text{Posti} \right.$

Poiché la parte a sinistra è superchiave, **Tavolo** è in BCNF.

$\text{Sala} \left\{ \emptyset \right.$

Poiché non ha alcuna dipendenza non banale, **Sala** è in BCNF.

$\text{Consegna} \left\{ \text{Comanda} \rightarrow \text{Sede}, \text{Pony}, \text{Partenza}, \text{Arrivo}, \text{Ritorno} \right.$

Poiché la parte a sinistra è superchiave, **Consegna** non è in BCNF.

³Si veda il paragrafo 5.4 a pagina 62.

Pony $\{ \text{Sede}, \text{ID} \rightarrow \text{Ruote}, \text{Stato}$

Poiché la parte a sinistra è superchiave, Pony è in BCNF.

Prenotazione $\left\{ \begin{array}{l} \text{ID} \rightarrow \text{Sede}, \text{Data}, \text{Numero}, \text{Account}, \text{Nome}, \text{Telefono}, \text{Sala}, \\ \text{Tavolo}, \text{Descrizione}, \text{Approvato} \end{array} \right.$

Poiché la parte a sinistra è superchiave, Prenotazione è in BCNF.

Account $\left\{ \begin{array}{l} \text{Username} \rightarrow \text{Email}, \text{Password}, \text{Nome}, \text{Cognome}, \text{Sesso}, \text{Città}, \\ \text{Via}, \text{NumeroCivico}, \text{Telefono}, \text{PuòPrenotare} \\ \text{Città}, \text{Via} \rightarrow \text{CAP} \end{array} \right.$

Poiché (Città, Via) non è superchiave, Account non è in BCNF.⁴

Proposta $\{ \text{ID} \rightarrow \text{Account}, \text{Nome}, \text{Procedimento}$

Poiché la parte a sinistra è superchiave, Proposta è in BCNF.

Composizione $\{ \emptyset$

Poiché non ha alcuna dipendenza funzionale non banale, Composizione è in BCNF.

Gradimento $\{ \text{ID} \rightarrow \text{Account}, \text{Proposta}, \text{Suggerimento}, \text{Punteggio}$

Poiché la parte a sinistra è superchiave, Gradimento è in BCNF.

Recensione $\{ \text{ID} \rightarrow \text{Account}, \text{Ricetta}, \text{Testo}, \text{Giudizio}$

Poiché la parte a sinistra è superchiave, Recensione è in BCNF.

Valutazione $\{ \text{Account}, \text{Recensione} \rightarrow \text{Veridicità}, \text{Accuratezza}, \text{Testo}$

Poiché la parte a sinistra è superchiave, Valutazione è in BCNF.

⁴Si veda il paragrafo 5.2 a fronte.

$$\text{Domanda} \left\{ \text{Numero}, \text{Sede} \rightarrow \text{Testo} \right.$$

Poiché la parte a sinistra è superchiave, **Domanda** è in BCNF.

$$\text{Risposta} \left\{ \text{Numero}, \text{Domanda}, \text{Sede} \rightarrow \text{Testo}, \text{Efficienza} \right.$$

Poiché la parte a sinistra è superchiave, **Risposta** è in BCNF.

$$\text{QuestionarioSvolto} \left\{ \begin{array}{l} \text{Recensione} \rightarrow \text{Sede} \\ \text{Recensione}, \text{Sede}, \text{Domanda} \rightarrow \text{Risposta} \end{array} \right.$$

Poiché **Recensione** (da sé) non è superchiave, **QuestionarioSvolto** non è in BCNF.⁵

5.2 Sede e Account

Le relazioni **Sede** e **Account** non rispettano la BCNF. A causare il problema sono le seguenti dipendenze funzionali:

- **Sede**: $\text{Città}, \text{Via} \rightarrow \text{CAP}$
- **Account**: $\text{Città}, \text{Via} \rightarrow \text{CAP}$

Queste dipendenze funzionali *non* sono però normalizzabili (a meno di non avere una tabella che metta in relazione tutte le coppie (**Città**, **Via**) con il relativo **CAP**). Si noti inoltre che, per quanto riguarda la relazione **Sede**, non saranno mai presenti due sedi con la stessa coppia (**Città**, **Via**) — è infatti assurdo che una catena di ristorazione possieda più di una sede nella stessa via — e quindi la *decomposizione* di tale dipendenza funzionale non porterebbe a nessun vantaggio. Per quanto riguarda la relazione **Account**, l'introduzione dell'attributo **CAP** non è richiesto dalle specifiche e non è necessario in quanto il *Codice di Avviamento Postale* è utile solo nel caso in cui sia necessario spedire un bene utilizzando i *servizi postali italiani*: le uniche *spedizioni* effettuate dalla catena di ristorazione sono le *consegne a domicilio* effettuate dai *pony*, che non hanno bisogno del **CAP** — volendo quindi normalizzare anche la tabella **Account**, si potrebbe semplicemente eliminare l'attributo **CAP**.⁶

⁵Si veda il paragrafo 5.5 a pagina 63.

⁶In questo progetto l'attributo **CAP** sarà comunque mantenuto sia per **Sede** che per **Account**.

5.3 Confezione

La relazione **Confezione** non rispetta la BCNF. Riportiamo di seguito tutte le dipendenze funzionali non banali della relazione in questione:

$$\text{Confezione} \left\{ \begin{array}{l} \text{CodiceLotto} \rightarrow \text{Ingrediente}, \text{Scadenza} \\ \text{CodiceLotto}, \text{Numero} \rightarrow \text{Peso}, \text{Prezzo}, \text{DataAcquisto}, \text{DataArrivo}, \\ \text{DataCarico}, \text{Sede}, \text{Magazzino}, \text{Collocazione}, \\ \text{Aspetto}, \text{Stato} \end{array} \right.$$

CodiceLotto, da sé, non è infatti *superchiave* della relazione **Confezione**. Per poter normalizzare tale relazione è necessario scomporla in due relazioni:

CONFEZIONE(CodiceLotto, Numero, Peso, Prezzo, DataAcquisto, DataArrivo, DataCarico, Sede, Magazzino, Collocazione, Aspetto, Stato)
 LOTTO(Codice, Ingrediente, Scadenza)

Dobbiamo anche aggiungere il *vincolo di integrità referenziale* tra **CodiceLotto** di **Confezione** e **Codice** di **Lotto**.

Si vede immediatamente che, così facendo, la BCNF è rispettata.

5.4 ModificaFase

La relazione **ModificaFase** non rispetta la BCNF. Riportiamo di seguito tutte le dipendenze funzionali non banali della relazione in questione:

$$\text{ModificaFase} \left\{ \begin{array}{l} \text{Variazione} \rightarrow \text{Ricetta} \\ \text{Variazione}, \text{ID} \rightarrow \text{FaseVecchia}, \text{FaseNuova} \end{array} \right.$$

Variazione, da sé, non è infatti *superchiave* della relazione **ModificaFase**. Per poter normalizzare tale relazione è necessario spostare l'attributo **Ricetta** nella relazione **Variazione**. Questo però ci impone anche di dover trovare un nuovo identificatore per **Fase** — useremo quindi un campo **ID** e toglieremo **Numero** (superfluo in quanto l'ordine delle fasi è dato dalla relazione **SequenzaFasi**). Bisognerà quindi modificare le relazioni coinvolte come segue:

FASE(ID, Ricetta, Ingrediente, Dose, Primario, Strumento, Testo, Durata)
 SEQUENZAFASI(Fase, FasePrecedente)
 VARIAZIONE(ID, Ricetta, Nome, Account)
 MODIFICAFASE(Variazione, ID, FaseVecchia, FaseNuova)

Anche i *vincoli di integrità referenziale* dovranno cambiare di conseguenza: dovrà essere aggiunto il vincolo tra **Ricetta** di **Variazione** e **Nome di Ricetta**; dovranno inoltre essere sistemati tutti i vincoli che si riferiscono alle tuple di **Fase** in quanto adesso vengono identificate dall'unico attributo **ID**.

Si vede immediatamente che, così facendo, la BCNF è rispettata per tutte le relazioni modificate.

5.5 QuestionarioSvolto

La relazione **QuestionarioSvolto** non rispetta la BCNF. Riportiamo di seguito tutte le dipendenze funzionali non banali della relazione in questione:

$$\text{QuestionarioSvolto} \begin{cases} \text{Recensione} \rightarrow \text{Sede} \\ \text{Recensione, Sede, Domanda} \rightarrow \text{Risposta} \end{cases}$$

Recensione, da sé, non è infatti *superchiave* della relazione **QuestionarioSvolto**. Per poter normalizzare tale relazione è necessario spostare l'attributo **Sede** nella relazione **Recensione**. Questo però ci impone anche di dover trovare un nuovo identificatore per le relazioni **Risposta** e **Domanda** — useremo quindi un campo **ID** per **Domanda** e toglieremo **Numero** (superfluo in quanto l'ordine delle domande può essere dato dalla sequenza degli **ID**). Inoltre possiamo (anche se non necessario per la normalizzazione) togliere **Risposta** dall'identificatore di **QuestionarioSvolto** (è possibile in quanto, per ogni recensione, l'utente può dare una sola risposta ad ogni domanda — come specificato dalla business rule (BR22)). Bisognerà quindi modificare le relazioni coinvolte come segue:

```
RECENSIONE(ID, Account, Sede, Ricetta, Testo, Giudizio)
DOMANDA(ID, Sede, Testo)
RISPOSTA(Domanda, Numero, Testo, Efficienza)
QUESTIONARIOSVOLTO(Recensione, Domanda, Risposta)
```

Anche i *vincoli di integrità referenziale* dovranno cambiare di conseguenza: dovrà essere aggiunto il vincolo tra **Sede** di **Recensione** e **Nome di Sede**; dovranno inoltre essere sistemati il vincolo tra **QuestionarioSvolto** e **Risposta** in quanto le tuple di quest'ultima adesso vengono identificate dai soli attributi **Domanda** e **Numero**. Ovviamente deve essere sistemato anche il vincolo tra **Risposta** e **Domanda** in quanto le tuple di quest'ultima adesso vengono identificate dall'unico attributo **ID**.

Si vede immediatamente che, così facendo, la BCNF è rispettata per tutte le relazioni modificate.

Capitolo 6

Tabelle utili

In questo Capitolo vengono aggiunte alcune tabelle non richieste dalle specifiche ma comunque utili al progetto. Nel paragrafo 6.1 viene presentato il *codice MySQL* che implementa e mantiene aggiornate alcune **tabelle di log**. Nel paragrafo 6.2 a pagina 67 viene presentato il *codice MySQL* che implementa e mantiene aggiornate alcune **materialized views**.

6.1 Tabelle di Log

Il seguente listato contiene il *codice MySQL* che implementa due tabelle di log: **Clienti_Log** e **Scarichi_Log**.

La prima tabella mantiene, per ogni sede e per ogni mese, il numero di clienti che si sono presentati *senza prenotazione*. Tale informazione sarà poi utilizzata per stimare il numero di clienti presenti in sala in un dato giorno al fine di calcolare la quantità di ingredienti necessari per produrre le ricette del menu.

Non è necessario registrare il numero di clienti che si sono presentati con prenotazione in quanto tale informazione può essere facilmente ricavata dalla tabella **Prenotazione**.

L'attributo contatore **SenzaPrenotazione** deve essere incrementato manualmente dallo Staff del ristorante ogni qual volta che si presenta un cliente senza prenotazione. Per farlo, è sufficiente chiamare la *stored procedure* **RegistraClienti**(VARCHAR(45), INT).

La seconda tabella mantiene invece le informazioni su tutti gli scarichi effettuati dai magazzini per ogni ingrediente. Tale informazione sarà poi utilizzata per effettuare l'analisi dei consumi e degli sprechi.

```
CREATE TABLE Clienti_Log
(
  Sede          VARCHAR(45) NOT NULL,
  Anno          INT UNSIGNED NOT NULL,
```

```

5      Mese                                INT UNSIGNED NOT NULL,
      SenzaPrenotazione                   INT UNSIGNED NOT NULL DEFAULT 0,
      PRIMARY KEY (Sede, Anno, Mese),
      FOREIGN KEY (Sede)
          REFERENCES Sede(Nome)
10      ON DELETE CASCADE
      ON UPDATE CASCADE
) ENGINE = InnoDB;

CREATE TABLE Scarichi_Log
15 (
    ID                                    INT UNSIGNED NOT NULL AUTO_INCREMENT,
    Sede                                VARCHAR(45) NOT NULL,
    Magazzino                           INT UNSIGNED NOT NULL,
    Ingrediente                         VARCHAR(45) NOT NULL,
20    'Timestamp'                       TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Quantita                           INT UNSIGNED NOT NULL DEFAULT 0,
    PRIMARY KEY (ID),
    FOREIGN KEY (Sede, Magazzino)
        REFERENCES Magazzino(Sede, ID)
25    ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY (Ingrediente)
        REFERENCES Ingrediente(Nome)
    ON DELETE CASCADE
30    ON UPDATE CASCADE
) ENGINE = InnoDB;

DELIMITER $$

35 CREATE TRIGGER nuova_sede
AFTER INSERT
ON Sede
FOR EACH ROW
BEGIN
40     INSERT INTO Clienti_Log(Sede, Anno, Mese) VALUES
        (NEW.Nome, YEAR(CURRENT_DATE), MONTH(CURRENT_DATE));
END; $$

CREATE PROCEDURE RegistraClienti(IN inSede VARCHAR(45), IN numero INT)
45 NOT DETERMINISTIC MODIFIES SQL DATA
BEGIN
    INSERT INTO Clienti_Log(Sede, Anno, Mese, SenzaPrenotazione) VALUES
        (inSede, YEAR(CURRENT_DATE), MONTH(CURRENT_DATE), numero)
    ON DUPLICATE KEY

```

```

50         UPDATE SenzaPrenotazione = SenzaPrenotazione + numero;
END;$$

CREATE TRIGGER aggiorna_Scarichi_Log_update
AFTER UPDATE
55 ON Confezione
FOR EACH ROW
BEGIN
    DECLARE IngScaricato VARCHAR(45);

60     IF OLD.Stato = 'in uso' AND NEW.Stato = 'parziale'
        AND OLD.Peso > NEW.Peso THEN
        SET IngScaricato = (SELECT L.Ingrediente
                            FROM Lotto L
                            WHERE L.Codice = NEW.CodiceLotto);

65     INSERT INTO Scarichi_Log(Sede, Magazzino, Ingrediente, Quantita)
        VALUES (NEW.Sede, NEW.Magazzino, IngScaricato, OLD.Peso - NEW.Peso);
    END IF;
END;$$

70 CREATE TRIGGER aggiorna_Scarichi_Log_delete
AFTER DELETE
ON Confezione
FOR EACH ROW
75 BEGIN
    DECLARE IngScaricato VARCHAR(45);

    IF OLD.Stato = 'in uso' THEN
        SET IngScaricato = (SELECT L.Ingrediente
                            FROM Lotto L
                            WHERE L.Codice = OLD.CodiceLotto);

80     INSERT INTO Scarichi_Log(Sede, Magazzino, Ingrediente, Quantita)
        VALUES (OLD.Sede, OLD.Magazzino, IngScaricato, OLD.Peso);

85     END IF;
END;$$

DELIMITER ;

```

6.2 Materialized View

Il seguente listato contiene il *codice MySQL* che implementa due *materialized view*: MV_OrdiniRicetta e MV_MenuCorrente.

La prima raccoglie, per ogni sede, il numero di volte che una ricetta viene ordinata e il numero di giorni che tale ricetta compare nel menu della sede. Tale informazione sarà utilizzata per stimare il numero di volte che un piatto viene ordinato ogni giorno al fine di calcolare la quantità di ingredienti necessari per produrre le ricette del menu.

La seconda contiene di volta in volta, per ogni sede, il menu selezionabile dall'utente. Nel menu devono infatti comparire solo le ricette per la quale c'è una quantità di ingredienti sufficiente in magazzino. L'evento che aggiorna le materialized view utilizza la funzione `IngredientiDisponibili(VARCHAR(45), VARCHAR(45), BOOL, DATE)` per determinare se c'è una quantità sufficiente di ingredienti (la funzione ritorna TRUE se tale quantità è presente nei magazzini della sede). Tale *stored function* non viene riportata in questa sede: è mostrata come operazione nel Capitolo 7 a pagina 71.

```

SET GLOBAL event_scheduler = on;

CREATE TABLE MV_OrdiniRicetta
(
5   Sede          VARCHAR(45) NOT NULL,
   Ricetta        VARCHAR(45) NOT NULL,
   Comparsa       INT UNSIGNED NOT NULL DEFAULT 1,
   TotOrdini      INT UNSIGNED NOT NULL,
   PRIMARY KEY (Sede, Ricetta),
10  FOREIGN KEY (Sede)
      REFERENCES Sede(Nome)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
   FOREIGN KEY (Ricetta)
15  REFERENCES Ricetta(Nome)
      ON DELETE CASCADE
      ON UPDATE CASCADE
) ENGINE = InnoDB;

20 CREATE TABLE MV_MenuCorrente
(
   Sede          VARCHAR(45) NOT NULL,
   Ricetta        VARCHAR(45) NOT NULL,
   Novita        BOOL NOT NULL DEFAULT FALSE,
25  PRIMARY KEY (Sede, Ricetta),
   FOREIGN KEY (Sede)
      REFERENCES Sede(Nome)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
30  FOREIGN KEY (Ricetta)
      REFERENCES Ricetta(Nome)
      ON DELETE CASCADE
      ON UPDATE CASCADE

```

```

35  ) ENGINE = InnoDB;

DELIMITER $$

40  CREATE EVENT aggiorna_MV_OrdiniRicetta_MenuCorrente
ON SCHEDULE
EVERY 1 DAY
STARTS TIMESTAMP(CURRENT_DATE) + INTERVAL 1 DAY + INTERVAL 2 HOUR
ON COMPLETION PRESERVE
DO
45  BEGIN
    DECLARE cSede VARCHAR(45);
    DECLARE cRicetta VARCHAR(45);
    DECLARE cOrdini INT;
    DECLARE cNovita BOOL;
50  DECLARE Finito BOOL DEFAULT FALSE;
    DECLARE curPiatto CURSOR FOR
        SELECT C.Sede, P.Ricetta, COUNT(*) AS Ordini
        FROM Piatto P INNER JOIN Comanda C ON P.Comanda = C.ID
        WHERE DATE(C.'Timestamp') = (CURRENT_DATE - INTERVAL 1 DAY)
55  GROUP BY C.Sede, P.Ricetta;
    DECLARE curElenco CURSOR FOR
        SELECT M.Sede, E.Ricetta, E.Novita
        FROM Elenco E INNER JOIN Menu M ON E.Menu = M.ID
        WHERE CURRENT_DATE
60  BETWEEN M.DataInizio AND M.DataFine;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET Finito = TRUE;

    OPEN curPiatto;

65  loop_lbl: LOOP
    FETCH curPiatto INTO cSede, cRicetta, cOrdini;
    IF Finito THEN
        LEAVE loop_lbl;
    END IF;

70  INSERT INTO MV_OrdiniRicetta(Sede, Ricetta, TotOrdini) VALUES
        (cSede, cRicetta, cOrdini)
        ON DUPLICATE KEY
            UPDATE Comparsa = Comparsa + 1, TotOrdini = TotOrdini + cOrdini;
75  END LOOP loop_lbl;

    CLOSE curPiatto;

```

```
80      SET Finito = FALSE;
      TRUNCATE TABLE MV_MenuCorrente; -- full refresh
      OPEN curElenco;

85      loop2_lbl: LOOP
          FETCH curElenco INTO cSede, cRicetta, cNovita;
          IF Finito THEN
              LEAVE loop2_lbl;
          END IF;

90          IF IngredientiDisponibili(cSede, cRicetta, cNovita, NULL) THEN
              INSERT INTO MV_MenuCorrente(Sede, Ricetta, Novita) VALUES
                  (cSede, cRicetta, cNovita);
          END IF;

95      END LOOP loop2_lbl;

      CLOSE curElenco;
      END; $$

100 DELIMITER ;
```

Capitolo 7

Operazioni

In questo Capitolo sono presentate alcune *operazioni interessanti*. Successivamente, nel paragrafo 7.1 nella pagina successiva, saranno presentate le *implementazioni MySQL* di tali operazioni.

Le operazioni che andremo ad analizzare sono le seguenti (le stime sul numero di esecuzioni al giorno sono basate anche su alcune delle considerazioni riportate nella *tavola dei volumi* del paragrafo 8.1 a pagina 77):

Operazione	Frequenza
1. Ottenere lo stato di una comanda.	1 500/ <i>giorno</i>
2. Assegnamento automatico di un pony a una comanda take-away mediante trigger.	125/ <i>giorno</i>
3. Aggiunta di una nuova comanda.	675/ <i>giorno</i>
4. Aggiunta di un nuovo piatto.	3 375/ <i>giorno</i>
5. Controllo della disponibilità degli ingredienti in magazzino per la produzione di una ricetta in un certo giorno.	500/ <i>giorno</i>
6. Aggiunta di una prenotazione.	375/ <i>giorno</i>
7. Classifica delle recensioni.	100/ <i>giorno</i>
8. Aggiunta di una valutazione.	3/ <i>giorno</i>

7.1 Implementazione delle operazioni

Di seguito le implementazioni MySQL delle operazioni individuate nel paragrafo precedente.

```

DELIMITER $$

-- OPERAZIONE 1
CREATE FUNCTION StatoComanda(idComanda INT)
5 RETURNS ENUM('nuova', 'in preparazione', 'parziale', 'evasa', 'consegna')
NOT DETERMINISTIC READS SQL DATA
BEGIN
    -- bit 1 set: contiene piatti in attesa
    -- bit 2 set: contiene piatti in preparazione
10 -- bit 3 set: contiene piatti in servizio
    DECLARE Flags INT;

    SET Flags = (SELECT IF(SUM(P.Stato = 'attesa') > 0, 1, 0)
                  + IF(SUM(P.Stato = 'in preparazione') > 0, 2, 0)
15                  + IF(SUM(P.Stato = 'servizio') > 0, 4, 0)
                  FROM Piatto P
                  WHERE P.Comanda = idComanda);

    CASE
20 WHEN Flags = 4 THEN -- tutti i piatti in servizio
        BEGIN
            DECLARE TakeAway BOOL DEFAULT FALSE;
            SET TakeAway = (SELECT C.Account IS NOT NULL
                             FROM Comanda C
25                             WHERE C.ID = idComanda);

            IF TakeAway THEN
                RETURN 'consegna';
            ELSE
                RETURN 'evasa';
30            END IF;
        END;
    WHEN Flags > 4 THEN RETURN 'parziale'; -- alcuni piatti in servizio
    WHEN Flags > 1 THEN RETURN 'in preparazione'; -- alcuni piatti in prep.
    ELSE RETURN 'nuova'; -- tutti i piatti in attesa (Flags = 1)
35 END CASE;
END; $$

-- OPERAZIONE 2
40 CREATE TRIGGER assegna_pony
AFTER UPDATE

```



```

ON Piatto
FOR EACH ROW
BEGIN
45   DECLARE NumeroPiatti INT;
   DECLARE SedeComanda VARCHAR(45);
   DECLARE PonyScelto INT;

   IF StatoComanda(NEW.Comanda) = 'consegna' THEN
50     SET SedeComanda = (SELECT C.Account <> NULL, C.Sede
                        FROM Comanda C
                        WHERE C.ID = NEW.Comanda);

     SET NumeroPiatti = (SELECT COUNT(*)
55                        FROM Piatto P
                        WHERE P.Comanda = NEW.Comanda);

     -- Se i piatti sono 5 o meno scelgo un pony su 2 ruote,
     -- altrimenti su 4 ruote
60     SET PonyScelto = (SELECT P.ID
                        FROM Pony P
                        WHERE P.Sede = SedeComanda
                        AND P.Stato = 'libero'
                        AND Ruote = (NumeroPiatti > 5)
65                        LIMIT 1);

     -- Se non disponibile scelgo un pony qualsiasi
     IF PonyScelto IS NULL THEN
70       SET PonyScelto = (SELECT P.ID
                        FROM Pony P
                        WHERE P.Sede = SedeComanda
                        AND P.Stato = 'libero'
                        LIMIT 1);

     END IF;

75     IF PonyScelto IS NULL THEN
       -- Nessun Pony disponibile
       SIGNAL SQLSTATE '01000' -- Warning
       SET MESSAGE_TEXT = 'Nessun Pony è stato assegnato in '
80       'quanto sono tutti occupati.';

     ELSE
       -- Assegna Pony
       INSERT INTO Consegna(Comanda, Sede, Pony, Arrivo, Ritorno)
       VALUES (NEW.Comanda, SedeComanda, PonyScelto, NULL, NULL);
85     END IF;

   END IF;
END; $$

```

```

90 DELIMITER ;

-- OPERAZIONE 3
INSERT INTO Comanda(Sede, Sala, Tavolo) VALUES ('nome sede', 2, 10);

95 -- OPERAZIONE 4
INSERT INTO Piatto(Comanda, Ricetta) VALUES (1, 'nome ricetta');

DELIMITER $$

100 -- OPERAZIONE 5
CREATE FUNCTION IngredientiDisponibili(cSede VARCHAR(45), cRicetta VARCHAR(45),
                                         cNovita BOOL, cData DATE)
RETURNS BOOL
NOT DETERMINISTIC READS SQL DATA
105 BEGIN
    DECLARE ClientiPrenotazioni INT;
    DECLARE MediaSenzaPrenotazione INT;
    DECLARE StimaClienti INT;
    DECLARE StimaOrdini INT;
110 DECLARE cIngrediente VARCHAR(45);
    DECLARE cDose INT;
    DECLARE cPrimario BOOL;
    DECLARE qtaDisponibile INT;
    DECLARE Finito BOOL DEFAULT FALSE;
115 DECLARE curIngredienti CURSOR FOR
        SELECT F.Ingrediente, SUM(F.Dose), SUM(F.Primario) > 0
        FROM Fase F
        WHERE F.Ricetta = cRicetta AND F.Ingrediente IS NOT NULL
        GROUP BY F.Ingrediente;
120 DECLARE CONTINUE HANDLER FOR NOT FOUND SET Finito = TRUE;

    IF cData IS NULL THEN
        SET cData = CURRENT_DATE; -- Default
    END IF;

125 SET ClientiPrenotazioni = (SELECT COALESCE(SUM(P.Numero), 0)
                                FROM Prenotazione P
                                WHERE P.Sede = cSede
                                    AND DATE(P.'Data') = cData);

130 -- AVG(SenzaPrenotazione) = media di clienti fuori prenotazione per tale
-- mese. Questo viene diviso per il numero di giorni che il mese
-- contiene (= media dei clienti fuori prenotazione in un giorno del mese).
```

```

135  -- [La stima non è precisissima nel caso del mese di febbraio per via degli
-- anni bisestili, ma non è importante: in fondo è pur sempre una stima]
SET MediaSenzaPrenotazione = (SELECT
                                CEIL(COALESCE(AVG(CL.SenzaPrenotazione), 0)/
                                DAY(LAST_DAY(cData)))
                                AS Media
140  FROM Clienti_Log CL
WHERE CL.Sede = cSede
AND CL.Mese = MONTH(cData)
AND CL.Anno <> YEAR(cData)
                                );

145  SET StimaClienti = ClientiPrenotazioni + MediaSenzaPrenotazione;

IF cNovita THEN
    -- 1/3 dei clienti ordina la ricetta
150  SET StimaOrdini = (SELECT CEIL(StimaClienti * 0.33));
ELSE
    -- Stima ordini viene calcolata come la media degli ordini della ricetta
    -- incrementata del 10% dei clienti stimati. L'incremento del 10% sul
    -- numero di clienti stimati serve come margine di sicurezza.
155  SET StimaOrdini = (SELECT (COALESCE(CEIL(MV.TotOrdini / MV.Comparsa), 0)
                                + StimaClienti * 0.1) AS StimaOrdini
                        FROM MV_OrdiniRicetta MV
                        WHERE MV.Sede = cSede AND MV.Ricetta = cRicetta);

END IF;

160  IF StimaOrdini < 5 THEN
    SET StimaOrdini = 5;
END IF;

165  OPEN curIngredienti;

loop_lbl: LOOP
    FETCH curIngredienti INTO cIngrediente, cDose, cPrimario;
    IF Finito THEN
170      LEAVE loop_lbl;
    END IF;

    -- somma il peso delle confezioni di quell'ingrediente che non sono in
    -- ordine o che arrivano con almeno tre giorni di anticipo rispetto a
175  -- cData e che non sono danneggiate (se l'ingrediente è primario in
    -- questa ricetta).
    SET qtaDisponibile = (SELECT SUM(C.Peso)
                            FROM Confezione C INNER JOIN Lotto L
                            ON C.CodiceLotto = L.Codice

```

```

180         WHERE C.Sede = cSede
              AND L.Ingrediente = cIngrediente
              AND (C.Stato <> 'in ordine'
                  OR (C.Stato = 'in ordine'
                      AND C.DataArrivo >=
185                          cData - INTERVAL 3 DAY))
              AND (C.Aspetto OR (NOT cPrimario)));

        IF qtaDisponibile < (cDose * StimaOrdini) THEN
            RETURN FALSE;
190        END IF;
    END LOOP loop_lbl;

    CLOSE curIngredienti;

195    RETURN TRUE;
END; $$

DELIMITER ;

200 -- OPERAZIONE 6
INSERT INTO Prenotazione(Sede, 'Data', Numero, Account, Sala, Tavolo)
VALUES ('nome sede', 'yyyy-mm-dd hh:mm:ss', 5, 'username', 2, 7);

205 -- OPERAZIONE 7
SELECT @row_number := @row_number + 1 AS Posizione, D.*
FROM (SELECT @row_number := 0) AS N,
    (
        SELECT R.ID AS Recensione,
210            SUM(COALESCE(V.Veridicita, 0)) AS VeridicitaTotale,
            SUM(COALESCE(V.Veridicita, 0)) AS AccuratezzaTotale,
            IF(V.Recensione IS NULL, 0, COUNT(*)) AS NumeroValutazioni
        FROM Recensione R LEFT OUTER JOIN Valutazione V ON R.ID = V.Recensione
        GROUP BY R.ID
215    ) AS D
ORDER BY (D.VeridicitaTotale + D.AccuratezzaTotale)/D.NumeroValutazioni DESC;

-- OPERAZIONE 8
220 INSERT INTO Valutazione(Account, Recensione, Veridicita, Accuratezza, Testo)
VALUES ('username', 10, 4, 3, 'testo testo testo');

```

Capitolo 8

Analisi delle prestazioni

In questo Capitolo viene presentato il risultato della fase di *analisi delle prestazioni* di alcune operazioni interessanti. Il paragrafo 8.1 riporta la tavola dei volumi delle *tabelle* del database. Il paragrafo 8.2 a pagina 81 riporta, per ogni operazione individuata nel Capitolo 7 a pagina 71, la *tavola degli accessi*.

8.1 Tavola dei volumi

La *tavola dei volumi* seguente contiene tre colonne: la prima riporta il nome della *tabella* che si considera; la seconda contiene il *volume stimato* della tabella; la terza *descrive* come è stata calcolata la stima.

Ogni qual volta che si riporta il nome di una tabella in una *espressione matematica* si deve intendere il *volume della tabella*.

Tabella	Volume	Commento
Sede	25	Ipotesi: la catena di ristorazione ha 25 sedi.
Magazzino	50	Ogni sede ha in media 2 magazzini: $2 \times Sede = 50$.
Confezione	20 000	Ogni magazzino ha in media 400 confezioni (alcune in ordine): $400 \times Magazzino = 20\,000$.
Ingrediente	150	Ipotesi: ci sono 150 ingredienti possibili.
Lotto	1 500	La catena di ristorazione possiede (nello stesso momento) 10 lotti per ogni ingrediente (eliminiamo i lotti per i quali non sono più presenti confezioni in alcun magazzino).
Cucina	750	$Strumento \times Sede = 750$.
Strumento	50	Ipotesi: ci sono 50 strumenti possibili.

Tabella	Volume	Commento
Funzione	150	Ogni strumento ha in media 3 funzioni: $3 \times Strumento = 150$.
Menu	500	Ogni sede ha applicato (nel tempo) in media 20 menu: $20 \times Sede = 500$. In realtà il numero di menu cresce anche di molto nel tempo (dipende dalla frequenza con cui una sede cambia menu e dal tempo trascorso), ma 500 può essere un'approssimazione adeguata.
Elenco	10 000	Ogni menu elenca in media 20 ricette: $20 \times Menu = 10\,000$.
Ricetta	200	Ipotesi: il ricettario della catena di ristorazione è formato da 200 ricette.
Fase	5 000	Ogni ricetta ha in media dalle 20 alle 25 fasi (quindi circa 22); inoltre in media $\frac{2}{3}$ delle istanze di ModificaFase richiedono l'aggiunta di una nuova fase: $\approx 22 \times Ricetta + \frac{2}{3} ModificaFase = 4\,900 \approx 5\,000$.
SequenzaFasi	7 500	Ogni fase ha in media una o due fasi che la precedono: $1.5 \times Fase = 7\,500$.
Piatto	∞ ($\approx 500\,000$)	Ogni comanda ordina in media 5 piatti: $5 \times Comanda = 500\,000$. Si veda anche la nota a fine tavola (∞).
Modifica	∞ ($\approx 25\,000$)	In media un piatto su 25 applica una variazione (o più di una — al massimo 3). Approssimiamo quindi a $\frac{1}{20} = 0.05$: $\approx 0.05 \times Piatto = 25\,000$. Si veda anche la nota a fine tavola (∞).
Variazione	1 000	Ogni ricetta ha in media due variazioni possibili; inoltre si ipotizza che gli utenti della piattaforma web rilascino 600 suggerimenti: $2 \times Ricetta + 600 = 1\,000$.
ModificaFase	1 500	Ogni variazione in media richiede la modifica di una o due fasi: $1.5 \times Variazione = 1\,500$.

Tabella	Volume	Commento
Comanda	∞ ($\approx 100\,000$)	Ipotesi: ogni giorno la metà dei tavoli di una sede sono occupati (considerando anche che la sede può fare più di un turno); ognuno di questi tavoli, in quel turno, invia una o due comande; inoltre si ipotizza che ogni giorno ogni sede riceva 5 comande take-away: $Tavolo/2 \times 1.5 \approx 550 + 5 \times Sede \approx 675/giorno$. Per il volume totale della tabella approssimiamo quindi ad un numero molto alto: $\approx 100\,000$. Si veda anche la nota a fine tavola (∞).
Tavolo	750	Ogni sala ha in media 15 tavoli: $15 \times Sala = 750$.
Sala	50	Ogni sede ha in media 2 sale: $2 \times Sede = 50$.
Consegna	∞ ($\approx 10\,000$)	Ipotesi: ogni giorno ogni sede riceve 5 comande take-away: $5 \times Sede = 125/giorno$. Per il volume totale della tabella approssimiamo quindi ad un numero alto: $\approx 10\,000$. Si veda anche la nota a fine tavola (∞).
Pony	100	Ogni sede ha in media 4 pony: $4 \times Sede = 100$.
Prenotazione	∞ ($\approx 50\,000$)	Ipotesi: ogni giorno ogni sede riceve 15 prenotazioni: $15 \times Sede = 375/giorno$. Per il volume totale della tabella approssimiamo quindi ad un numero alto: $\approx 50\,000$. Si veda anche la nota a fine tavola (∞).
Account	10 000	Ipotesi: nel tempo si registrano circa 10 000 utenti.
Proposta	1 000	In media <i>meno</i> di un utente su 10 rilascerà una proposta sulla piattaforma web, però qualcuno di questi utenti ne rilascerà più di una. Approssimiamo quindi a $1/10 = 0.1$: $\approx 0.1 \times Account = 1\,000$.
Composizione	7 000	Una ricetta (anche quelle proposte) in media è composta da 7 ingredienti: $7 \times Proposta = 7\,000$.
Gradimento	4 800	Ogni proposta ha in media 3 gradimenti; ogni suggerimento ha in media 3 gradimenti: $3 \times Proposta + 3 \times Suggerimento = 4\,800$.

Tabella	Volume	Commento
Recensione	2 000	In media solo un utente su 10 si preoccuperà di rilasciare recensioni sul sito; ognuno di questi rilascerà in media 2 recensioni: $2 \times 0.1 \times Account = 2\,000$.
Valutazione	4 000	Ogni recensione ha in media 2 valutazioni: $2 \times Recensione = 4\,000$.
Domanda	125	Ogni sede ha un questionario (insieme di domande); ognuno di questi questionari è composto in media da 5 domande: $5 \times Sede = 125$.
Risposta	375	Ogni domanda ha in media 3 risposte possibili: $3 \times Domanda = 375$.
Questionario–Svolto	1 000	La tabella <code>QuestionarioSvolto</code> mette in relazione <code>Recensione</code> e <code>Risposta</code> associando ad ogni recensione le varie risposte date al questionario. Ogni questionario è composto in media da 5 domande: $Recensione \times 5 = 1\,000$.
Clienti_Log	1 500	Se la catena di ristorazione è aperta da 5 anni: $\approx Sede \times 5 \times 12 = 1\,500$.
Scarichi_Log	∞ ($\approx 100\,000$)	Approssimiamo ad un numero alto: $\approx 100\,000$, considerando che l'amministratore del database dovrebbe, ogni tanto, provvedere a ripulire questa tabella dalle informazioni non più importanti. Si veda anche la nota a fine tavola (∞).
MV_Ordini–Ricetta	5 000	$Sede \times Ricetta = 5\,000$.
MV_Menu–Corrente	500	Ogni menu elenca in media 20 ricette: $Sede \times Elenco = 500$.

NOTA (∞):

Per alcune tabelle (`Comanda`, `Piatto`, ecc...) al posto del volume è stato inserito il simbolo di infinito (∞) e, tra parentesi, un'approssimazione del volume. Per queste tabelle non è possibile fare una stima che si possa ritenere *precisa* del volume in quanto dipendente da fattori molto aleatori (come il tempo).

Ad esempio il numero di comande (e quindi anche dei piatti ordinati) aumenta notevolmente via via che il tempo passa: dopo un intero anno il volume della tabella

Comanda può essere anche di *centinaia di migliaia di record*; dopo 5 anni il volume della tabella **Piatto** può essere anche di *qualche milione di record*. Ovviamente non ha senso mantenere per sempre (o molto a lungo) le informazioni sulle comande e sui piatti ordinati e l'amministratore dovrebbe occuparsi di ripulire il database da informazioni non più utili¹. Stimiamo però comunque valori molto alti per il volume di queste tabelle, così da tenere in considerazione il caso in cui l'amministratore non provveda molto frequentemente alla pulizia del database. Il valore così stimato è inserito, nella tavola, tra parentesi tonde sotto il simbolo ∞ . Questo valore sarà quello utilizzato per tutte le stime di qui in poi (ad esempio, nelle *tavole degli accessi* per le operazioni).

8.2 Tavole degli accessi

Vediamo le *tavole degli accessi* delle operazioni individuate nel Capitolo 7 a pagina 71.

Per la prima operazione si nota facilmente che sono necessarie cinque letture su **Piatto**, in quanto ogni comanda ha in media 5 piatti, e una su **Comanda**:

Operazione 1		
Tabella	Accessi	Tipo
Piatto	5	L
Comanda	1	L
Costo totale: $5 + 1 = 6 \times 1500 = 9\,000$ /giorno		

Per la seconda operazione il *trigger assegna_pony* effettua una chiamata a **Stato-Comanda** la quale effettua cinque letture su **Piatto** e una lettura su **Comanda**, poi altre cinque letture su **Piatto** (per contare il numero di piatti), quattro letture su **Pony**, in quanto ogni sede ha in media 4 pony, e una scrittura su **Consegna** per piazzare la nuova consegna. Quest'ultima scrittura causerà a sua volta l'esecuzione di un trigger che provvederà ad aggiornare lo stato del pony su '**occupato**' — quindi effettua una scrittura anche su **Pony**:

¹La base di dati di questo progetto non contiene funzionalità per la pulizia di informazioni ritenute superflue: l'eventuale pulizia del database, se desiderata, è lasciata all'intervento manuale dell'amministratore.

Operazione 2		
Tabella	Accessi	Tipo
Comanda	1	L
Piatto	10	L
Pony	4	L
Consegna	1	S
Pony	1	S
Costo totale: $1 + 10 + 4 + 1 \times 2 + 1 \times 2 = 19 \times 125 = \mathbf{2\,375/giorno}$		

La terza operazione è una scrittura su Comanda:

Operazione 3		
Tabella	Accessi	Tipo
Comanda	1	S
Costo totale: $1 \times 2 = 2 \times 675 = \mathbf{1\,350/giorno}$		

La quarta operazione è una scrittura su Piatto:

Operazione 4		
Tabella	Accessi	Tipo
Piatto	1	S
Costo totale: $1 \times 2 = 2 \times 3\,375 = \mathbf{6\,750/giorno}$		

Per la quinta operazione la funzione `IngredientiDisponibili` effettua 15 letture su `Prenotazione`, in quanto una sede riceve in media 15 prenotazioni al giorno, cinque letture su `Clienti_Log`, ipotizzando che la catena di ristorazione sia aperta da 5 anni, una lettura su `MV_OrdiniRicetta`, sette letture su `Fase`, ipotizzando che una ricetta sia composta in media da 7 fasi che aggiungono ingredienti, 42 letture su `Confezione`, ipotizzando che in media un magazzino contiene tre confezioni di ogni ingrediente e una sede possiede due magazzini:

Operazione 5		
Tabella	Accessi	Tipo
Prenotazione	15	L
Clienti_Log	5	L
MV_OrdiniRicetta	1	L
Fase	7	L
Confezione	42	L
Costo totale: $15 + 5 + 1 + 7 + 42 = 70 \times 500 = \mathbf{35\,000/giorno}$		

La sesta operazione è una scrittura su Prenotazione:

Operazione 6		
Tabella	Accessi	Tipo
Prenotazione	1	S
Costo totale: $1 \times 2 = 2 \times 375 = \mathbf{750/giorno}$		

Per la settima operazione è necessario, per poter stilare la classifica, leggere tutte le istanze di Recensione e tutte quelle di Valutazione:

Operazione 7		
Tabella	Accessi	Tipo
Recensione	2 000	L
Valutazione	4 000	L
Costo totale: $2\,000 + 4\,000 = 6\,000 \times 100 = \mathbf{60\,000/giorno}$		

L'ottava operazione è una scrittura su Valutazione:

Operazione 8		
Tabella	Accessi	Tipo
Valutazione	1	S
Costo totale: $1 \times 2 = 2 \times 3 = \mathbf{6/giorno}$		

Capitolo 9

Introduzione di ridondanze

In questo Capitolo sono introdotte alcune ridondanze utili a rendere il database più performante. I prossimi due paragrafi introducono e implementano le *ridondanze*. Il paragrafo 9.3 a pagina 87 contiene le *nuove implementazioni* di alcune operazioni individuate nel Capitolo 7 a pagina 71. Il paragrafo 9.4 a pagina 89 mostra le *nuove tavole degli accessi* ottenute dall'introduzione delle ridondanze.

9.1 MV_ClientiPrenotazione

La prima ridondanza che introduciamo viene implementata come una nuova tabella. Più precisamente, implementiamo una nuova *materialized view*. Dopotutto una materialized view può essere anche vista come una forma di ridondanza.

Questa nuova tabella conterrà, per ogni sede e per ogni giorno, il numero di clienti che si presentano con prenotazione. Viene mantenuta aggiornata da alcuni trigger su Prenotazione:

```
CREATE TABLE MV_ClientiPrenotazione
(
  Sede                VARCHAR(45) NOT NULL,
  'Data'              DATE NOT NULL,
  Numero              INT UNSIGNED NOT NULL DEFAULT 0,
  PRIMARY KEY (Sede, 'Data'),
  FOREIGN KEY (Sede)
    REFERENCES Sede(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE
) ENGINE = InnoDB;

DELIMITER $$
```

```

15 CREATE TRIGGER aggiorna_MV_ClientiPrenotazione_insert
   AFTER INSERT
   ON Prenotazione
   FOR EACH ROW
   BEGIN
20     INSERT INTO MV_ClientiPrenotazione(Sede, 'Data', Numero)
       VALUES (NEW.Sede, DATE(NEW.'Data'), NEW.Numero)
       ON DUPLICATE KEY
         UPDATE Numero = Numero + NEW.Numero;
   END; $$

25 CREATE TRIGGER aggiorna_MV_ClientiPrenotazione_update
   AFTER UPDATE
   ON Prenotazione
   FOR EACH ROW
   BEGIN
30     IF NEW.Numero <> OLD.Numero THEN
       INSERT INTO MV_ClientiPrenotazione(Sede, 'Data', Numero)
         VALUES (NEW.Sede, DATE(NEW.'Data'), NEW.Numero)
         ON DUPLICATE KEY
35         UPDATE Numero = Numero - OLD.Numero + NEW.Numero;
     END IF;
   END; $$

40 CREATE TRIGGER aggiorna_MV_ClientiPrenotazione_delete
   AFTER DELETE
   ON Prenotazione
   FOR EACH ROW
   BEGIN
45     UPDATE MV_ClientiPrenotazione
       SET Numero = Numero - OLD.Numero
       WHERE Sede = OLD.Sede
         AND 'Data' = DATE(OLD.'Data');
   END; $$

50 DELIMITER ;

```

9.2 Punteggio recensioni

L'ottava operazione individuata nel Capitolo 7 a pagina 71 effettua un gran numero di operazioni elementari. Possiamo cercare di ridurre questo numero, e rendere quindi l'operazione più efficiente, aggiungendo una ridondanza sulla tabella **Recensione** che riporta i punteggi totali di veridicità e accuratezza e il numero di valutazioni date alla recensione:

```

ALTER TABLE Recensione
ADD COLUMN VeridicitaTotale INT UNSIGNED NOT NULL DEFAULT 0 AFTER Giudizio
ADD COLUMN
    AccuratezzaTotale INT UNSIGNED NOT NULL DEFAULT 0 AFTER VeridicitaTotale
5 ADD COLUMN
    NumeroValutazioni INT UNSIGNED NOT NULL DEFAULT 0 AFTER AccuratezzaTotale;

DELIMITER $$

10 CREATE TRIGGER aggiorna_ridondanza_Recensione
    AFTER INSERT
    ON Valutazione
    FOR EACH ROW
    BEGIN
15     UPDATE Recensione R
        SET R.VeridicitaTotale = R.VeridicitaTotale + NEW.Veridicita,
            R.AccuratezzaTotale = R.AccuratezzaTotale + NEW.Accuratezza,
            NumeroValutazioni = NumeroValutazioni + 1
        WHERE R.ID = NEW.Recensione;
20 END;$$

DELIMITER ;

```

9.3 Nuove operazioni

Di seguito le implementazioni MySQL delle operazioni individuate nel Capitolo 7 a pagina 71 ottimizzate con le ridondanze introdotte nel paragrafo precedente (sono presentate solo due operazioni in quanto le altre rimangono invariate):

```

DELIMITER $$

-- OPERAZIONE 5
CREATE FUNCTION IngredientiDisponibili(cSede VARCHAR(45), cRicetta VARCHAR(45),
5                                     cNovita BOOL, cData DATE)
    RETURNS BOOL
    NOT DETERMINISTIC READS SQL DATA
    BEGIN
10     DECLARE ClientiPrenotazioni INT;
        DECLARE MediaSenzaPrenotazione INT;
        DECLARE StimaClienti INT;
        DECLARE StimaOrdini INT;
        DECLARE cIngrediente VARCHAR(45);
        DECLARE cDose INT;
15     DECLARE cPrimario BOOL;
        DECLARE qtaDisponibile INT;

```

```

DECLARE Finito BOOL DEFAULT FALSE;
DECLARE curIngredienti CURSOR FOR
  SELECT F.Ingrediente, SUM(F.Dose), SUM(F.Primario) > 0
20  FROM Fase F
  WHERE F.Ricetta = cRicetta AND F.Ingrediente IS NOT NULL
  GROUP BY F.Ingrediente;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET Finito = TRUE;

25  IF cData IS NULL THEN
    SET cData = CURRENT_DATE;
  END IF;

  -- il conteggio del numero dei clienti è ora immediato
30  SET ClientiPrenotazioni = (SELECT COALESCE(CP.Numero, 0)
    FROM MV_ClientiPrenotazione CP
    WHERE CP.Sede = cSede
    AND CP.'Data' = cData);

35  SET MediaSenzaPrenotazione = (SELECT
    CEIL(COALESCE(AVG(CL.SenzaPrenotazione), 0)/
    DAY(LAST_DAY(cData)))
    AS Media
    FROM Clienti_Log CL
40  WHERE CL.Sede = cSede
    AND CL.Mese = MONTH(cData)
    AND CL.Anno <> YEAR(cData)
    );

45  SET StimaClienti = ClientiPrenotazioni + MediaSenzaPrenotazione;

  IF cNovita THEN
    SET StimaOrdini = (SELECT CEIL(StimaClienti * 0.33));
  ELSE
50  SET StimaOrdini = (SELECT (COALESCE(CEIL(MV.TotOrdini / MV.Comparese), 0)
    + StimaClienti * 0.1) AS StimaOrdini
    FROM MV_OrdiniRicetta MV
    WHERE MV.Sede = cSede AND MV.Ricetta = cRicetta);

  END IF;

55  IF StimaOrdini < 5 THEN
    SET StimaOrdini = 5;
  END IF;

60  OPEN curIngredienti;

  loop_lbl: LOOP

```



```

65     FETCH curIngredienti INTO cIngrediente, cDose, cPrimario;
    IF Finito THEN
        LEAVE loop_lbl;
    END IF;

    SET qtaDisponibile = (SELECT SUM(C.Peso)
                           FROM Confezione C INNER JOIN Lotto L
                                ON C.CodiceLotto = L.Codice
                           WHERE C.Sede = cSede
                                AND L.Ingrediente = cIngrediente
                                AND (C.Stato <> 'in ordine'
                                     OR (C.Stato = 'in ordine'
                                          AND C.DataArrivo >=
                                              cData - INTERVAL 3 DAY))
                                AND (C.Aspetto OR (NOT cPrimario)));

    IF qtaDisponibile < (cDose * StimaOrdini) THEN
80         RETURN FALSE;
    END IF;
    END LOOP loop_lbl;

    CLOSE curIngredienti;
85
    RETURN TRUE;
END;$$

DELIMITER ;

90
-- OPERAZIONE 7
SELECT @row_number := @row_number + 1 AS Posizione, D.*
FROM (SELECT @row_number := 0) AS N,
    (
95         SELECT R.ID AS Recensione,
                R.VeridicitaTotale, R.AccuratezzaTotale, R.NumeroValutazioni
        FROM Recensione R
        GROUP BY R.ID
    ) AS D
100 ORDER BY (D.VeridicitaTotale + D.AccuratezzaTotale)/D.NumeroValutazioni DESC;

```

9.4 Nuove tavole degli accessi

Vediamo come cambiano le tavole degli accessi per alcune operazioni in seguito all'introduzione delle ridondanze individuate in questo Capitolo.

La prima ridondanza impatta sulle operazioni 5 e 6 (la seconda esegue il trigger che aggiorna la ridondanza):

Operazione 5		
Tabella	Accessi	Tipo
MV_ClientiPrenotazione	1	L
Clienti_Log	5	L
MV_OrdiniRicetta	1	L
Fase	7	L
Confezione	2	L
Costo totale: $1 + 5 + 1 + 7 + 42 = 46 \times 500 = \mathbf{23\ 000/giorno}$		

Operazione 6		
Tabella	Accessi	Tipo
Prenotazione	1	S
MV_ClientiPrenotazione	1	S
Costo totale: $1 \times 2 + 1 \times 2 = 4 \times 375 = \mathbf{1\ 500/giorno}$		

In definitiva:

Costo senza ridondanza: $35\ 000 + 750 = \mathbf{35\ 750/giorno}$

Costo con ridondanza: $23\ 000 + 1\ 500 = \mathbf{24\ 500/giorno}$

Gli altri due trigger usati per tenere la ridondanza aggiornata possono essere trascurati in quanto vengono eseguiti molto raramente. La ridondanza risulta conveniente: decidiamo quindi di mantenerla.

La seconda ridondanza impatta sulle operazioni 7 e 8 (di nuovo, la seconda esegue il trigger che aggiorna la ridondanza):

Operazione 7		
Tabella	Accessi	Tipo
Recensione	2 000	L
Costo totale: $2\ 000 \times 100 = \mathbf{20\ 000/giorno}$		

Operazione 8		
Tabella	Accessi	Tipo
Valutazione	1	S
Recensione	1	S
Costo totale: $1 \times 2 + 1 \times 2 = 4 \times 3 = \mathbf{12/giorno}$		

In definitiva:

Costo senza ridondanza: $60\,000 + 6 = \mathbf{60\,006/giorno}$

Costo con ridondanza: $20\,000 + 12 = \mathbf{20\,012/giorno}$

La ridondanza risulta conveniente: decidiamo quindi di mantenerla.

Capitolo 10

Area Analytics

In questo Capitolo sono descritte e implementate tutte le *funzionalità di back-end* richieste per l'*Area Analytics*.

10.1 Magazzino intelligente

La seguente procedura stende, per la sede passata come parametro, una classifica dei primi cinque piatti che dovrebbero essere aggiunti al menu. La classifica si basa sul numero di ingredienti che stanno per scadere (considerando che le confezioni nello stato **parziale** hanno una durata minore del 20% rispetto alle confezioni nello stato **completa**) e sulla classifica dei piatti preferiti realizzata nel paragrafo successivo.

La tabella `Report_PiattiDaAggiungere` contiene il risultato.

```
CREATE TABLE Report_PiattiDaAggiungere
(
  Posizione          INT UNSIGNED NOT NULL,
  Sede               VARCHAR(45) NOT NULL,
  Ricetta            VARCHAR(45) NOT NULL,
  PRIMARY KEY(Posizione),
  UNIQUE KEY (Sede, Ricetta),
  FOREIGN KEY (Sede)
    REFERENCES Sede(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (Ricetta)
    REFERENCES Ricetta(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE
) ENGINE = InnoDB;

CREATE OR REPLACE VIEW IngredientiInScadenza AS
```

```

SELECT C.Sede, L.Ingrediente
20 FROM Lotto L INNER JOIN Confezione C ON L.Codice = C.CodiceLotto
WHERE (C.Stato = 'completa' AND L.Scadenza < CURRENT_DATE + INTERVAL 5 DAY)
    OR (C.Stato = 'parziale' AND FROM_DAYS(TO_DAYS(L.Scadenza) -
        ROUND(TIMESTAMPDIFF(DAY, C.DataAcquisto, L.Scadenza)*0.2)) <
25                                     CURRENT_DATE + INTERVAL 5 DAY)
GROUP BY C.Sede, L.Ingrediente;

DELIMITER $$

CREATE PROCEDURE ConsigliaPiatti(IN nomeSede VARCHAR(45))
30 NOT DETERMINISTIC MODIFIES SQL DATA
BEGIN
    DELETE FROM Report_PiattiDaAggiungere WHERE Sede = nomeSede;

    INSERT INTO Report_PiattiDaAggiungere(Posizione, Sede, Ricetta)
35 SELECT @row_number := @row_number + 1 AS Posizione, nomeSede, D.Ricetta
FROM (SELECT @row_number := 0) AS N,
    (SELECT R.Nome AS Ricetta, COUNT(*) AS InScadenza,
        (SELECT IF(RPP.NumeroRecensioni = 0, 0,
40                     (RPP.GiudizioTotale/RPP.NumeroRecensioni)/10)
FROM Report_PiattiPreferiti RPP
WHERE RPP.Sede = nomeSede
    AND RPP.Ricetta = R.Nome) AS Punteggio
FROM Fase F INNER JOIN Ricetta R ON F.Ricetta = R.Nome
WHERE F.Ingrediente IS NOT NULL
45     AND F.Ingrediente IN (SELECT IIS.Ingrediente
                            FROM IngredientiInScadenza IIS
                            WHERE IIS.Sede = nomeSede)

    GROUP BY R.Nome) AS D
ORDER BY (D.InScadenza + D.Punteggio) DESC
50 LIMIT 5;
END;$$

DELIMITER ;

```

10.2 Analisi multidimensionale del business

Nel seguente listato sono definite le seguenti procedure:

- **AnalizzaRecensioni()** — Produce, per ogni sede, la classifica delle ricette meglio recensite, ponderando il giudizio in base alle valutazioni date alle recensioni. Inserisce il risultato in `Report_PiattiPreferiti`.

- `AnalizzaVendite(TIMESTAMP, TIMESTAMP)` — Produce, per ogni sede, la classifica delle ricette più vendute in un certo periodo di tempo. Inserisce il risultato in `Report_VenditePiatti`.
- `AnalizzaSuggerimenti()` — Produce la classifica dei suggerimenti più apprezzati. Inserisce il risultato in `Report_SuggerimentiMigliori`.
- `AnalizzaProposte()` — Produce la classifica delle proposte più apprezzate. Inserisce il risultato in `Report_ProposteMigliori`.

Infine, l'evento `Analytics_Scheduler` si occupa di eseguire automaticamente e periodicamente tutte queste procedure.

```

CREATE TABLE Report_PiattiPreferiti
(
  Posizione          INT UNSIGNED NOT NULL,
  Sede               VARCHAR(45) NOT NULL,
  Ricetta            VARCHAR(45) NOT NULL,
  GiudizioTotale     INT UNSIGNED NOT NULL,
  NumeroRecensioni   INT UNSIGNED NOT NULL,
  PRIMARY KEY (Posizione),
  UNIQUE KEY (Sede, Ricetta),
  FOREIGN KEY (Sede)
    REFERENCES Sede(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (Ricetta)
    REFERENCES Ricetta(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE
) ENGINE = InnoDB;

CREATE TABLE Report_VenditePiatti
(
  Posizione          INT UNSIGNED NOT NULL,
  Sede               VARCHAR(45) NOT NULL,
  Ricetta            VARCHAR(45) NOT NULL,
  Vendite            INT UNSIGNED NOT NULL,
  PRIMARY KEY (Posizione),
  UNIQUE KEY (Sede, Ricetta),
  FOREIGN KEY (Sede)
    REFERENCES Sede(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (Ricetta)
    REFERENCES Ricetta(Nome)

```

```

35         ON DELETE CASCADE
        ON UPDATE CASCADE
    ) ENGINE = InnoDB;

CREATE TABLE Report_SuggerimentiMigliori
(
40     Posizione            INT UNSIGNED NOT NULL,
    Suggerimento          INT UNSIGNED NOT NULL,
    GradimentoTotale      INT UNSIGNED NOT NULL,
    NumeroGradimenti      INT UNSIGNED NOT NULL,
    PRIMARY KEY (Posizione),
45     UNIQUE KEY (Suggerimento),
    FOREIGN KEY (Suggerimento)
        REFERENCES Variazione(ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
50 ) ENGINE = InnoDB;

CREATE TABLE Report_ProposteMigliori
(
55     Posizione            INT UNSIGNED NOT NULL,
    Proposta              INT UNSIGNED NOT NULL,
    GradimentoTotale      INT UNSIGNED NOT NULL,
    NumeroGradimenti      INT UNSIGNED NOT NULL,
    PRIMARY KEY (Posizione),
    UNIQUE KEY (Proposta),
60     FOREIGN KEY (Proposta)
        REFERENCES Proposta(ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
    ) ENGINE = InnoDB;

65 DELIMITER $$

CREATE PROCEDURE AnalizzaRecensioni()
NOT DETERMINISTIC MODIFIES SQL DATA
70 BEGIN
    TRUNCATE TABLE Report_PiattiPreferiti;

    INSERT INTO Report_PiattiPreferiti(Posizione, Sede, Ricetta, GiudizioTotale,
                                         NumeroRecensioni)
75     SELECT @row_number := @row_number + 1 AS Posizione, D.*
    FROM (SELECT @row_number := 0) AS N,
        (
            SELECT R.Sede, R.Ricetta,
                SUM(R.Giudizio)*IF(SUM(R.NumeroValutazioni) = 0, 6, ROUND(

```



```

80         AVG((R.VeridicitaTotale + R.AccuratezzaTotale)/R.NumeroValutazioni))
           ) AS GiudizioTotale, COUNT(*) AS NumeroRecensioni
           FROM Recensione R
           GROUP BY R.Sede, R.Ricetta
       ) AS D
85   ORDER BY D.GiudizioTotale/D.NumeroRecensioni DESC;
END;$$

CREATE PROCEDURE AnalizzaVendite(IN inizio TIMESTAMP, IN fine TIMESTAMP)
NOT DETERMINISTIC MODIFIES SQL DATA
90 BEGIN
    IF inizio IS NULL THEN
        SET inizio = '1970-01-01 00:00:01';
    END IF;
    IF fine IS NULL THEN
95        SET fine = CURRENT_TIMESTAMP;
    END IF;

    TRUNCATE TABLE Report_VenditePiatti;

100   INSERT INTO Report_VenditePiatti(Posizione, Sede, Ricetta, Vendite)
SELECT @row_number := @row_number + 1 AS Posizione, D.*
FROM (SELECT @row_number := 0) AS N,
    (
        SELECT C.Sede, P.Ricetta, COUNT(*) AS Vendite
105        FROM Piatto P INNER JOIN Comanda C ON P.Comanda = C.ID
        WHERE C.'Timestamp' BETWEEN inizio AND fine
        GROUP BY C.Sede, P.Ricetta
    ) AS D
    ORDER BY D.Vendite DESC;
110 END;$$

CREATE PROCEDURE AnalizzaSuggerimenti()
NOT DETERMINISTIC MODIFIES SQL DATA
BEGIN
115   TRUNCATE TABLE Report_SuggerimentiMigliori;

    INSERT INTO Report_SuggerimentiMigliori(Posizione, Suggerimento,
                                                GradimentoTotale, NumeroGradimenti)
SELECT @row_number := @row_number + 1 AS Posizione, D.*
120 FROM (SELECT @row_number := 0) AS N,
    (
        SELECT G.Suggerimento, SUM(G.Punteggio) AS GradimentoTotale,
                                                COUNT(*) AS NumeroGradimenti
        FROM Gradimento G

```

```

125         WHERE G.Suggerimento IS NOT NULL
           GROUP BY G.Suggerimento
        ) AS D
    ORDER BY D.GradimentoTotale/D.NumeroGradimenti DESC;
END; $$

130 CREATE PROCEDURE AnalizzaProposte()
    NOT DETERMINISTIC MODIFIES SQL DATA
    BEGIN
        TRUNCATE TABLE Report_ProposteMigliori;

135         INSERT INTO Report_ProposteMigliori(Posizione, Proposta, GradimentoTotale,
                                                NumeroGradimenti)

        SELECT @row_number := @row_number + 1 AS Posizione, D.*
        FROM (SELECT @row_number := 0) AS N,

140         (
            SELECT G.Proposta, SUM(G.Punteggio) AS GradimentoTotale,
                                   COUNT(*) AS NumeroGradimenti

            FROM Gradimento G
            WHERE G.Proposta IS NOT NULL
            GROUP BY G.Proposta

145         ) AS D
    ORDER BY D.GradimentoTotale/D.NumeroGradimenti DESC;
END; $$

150 CREATE EVENT Analytics_Scheduler
    ON SCHEDULE
    EVERY 1 MONTH
    STARTS TIMESTAMP(CURRENT_DATE) + INTERVAL 1 DAY + INTERVAL 4 HOUR
155    ON COMPLETION PRESERVE
    DO
    BEGIN
        CALL AnalizzaRecensioni();
        CALL AnalizzaVendite(CURRENT_TIMESTAMP - INTERVAL 1 MONTH, NULL);
160        CALL AnalizzaSuggerimenti();
        CALL AnalizzaProposte();
    END; $$

DELIMITER ;

```

10.3 Fornitura automatizzata del magazzino

Il seguente evento produce una lista di ordinativi da trasmettere ai fornitori. L'evento calcola, per ogni ricetta e per ogni sede, una stima del numero di ordini di tale ricetta nella settimana che viene. Prende poi, per ogni ingrediente che compone la ricetta in esame, la quantità di ingrediente necessario a produrre il numero di piatti stimato. La stima sul numero di ordini della ricetta si basa sulle prenotazioni in essere per la settimana e sulla media degli ordini di tale ricetta in passato nello stesso periodo.

La tabella `Report_Ordinativi` contiene gli ordinativi da trasmettere ai fornitori.

```

CREATE TABLE Report_Ordinativi
(
  Sede          VARCHAR(45) NOT NULL,
  Ingrediente   VARCHAR(45) NOT NULL,
  Quantita      INT UNSIGNED NOT NULL,
  PRIMARY KEY (Sede, Ingrediente),
  FOREIGN KEY (Sede)
    REFERENCES Sede(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (Ingrediente)
    REFERENCES Ingrediente(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
)ENGINE = InnoDB;

DELIMITER $$

CREATE EVENT Elenco_Ordini
ON SCHEDULE
EVERY 1 WEEK
STARTS TIMESTAMP(CURRENT_DATE) + INTERVAL 5 HOUR
ON COMPLETION PRESERVE
DO
BEGIN
  DECLARE ClientiPrenotazioni INT;
  DECLARE MediaSenzaPrenotazione INT;
  DECLARE StimaClienti INT;
  DECLARE StimaOrdini INT;
  DECLARE NomeSede VARCHAR(45);
  DECLARE NomeRicetta VARCHAR(45);
  DECLARE RicettaNovita BOOL;
  DECLARE Finito BOOL DEFAULT FALSE;
  DECLARE curRicetta CURSOR FOR
    SELECT M.Sede, E.Ricetta, E.Novita

```

```

FROM Menu M INNER JOIN Elenco E ON M.ID = E.Menu
WHERE M.DataInizio <= CURRENT_DATE
      AND M.DataFine >= CURRENT_DATE + INTERVAL 6 DAY;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET Finito = TRUE;

40 TRUNCATE TABLE Report_Ordinativi;

OPEN curRicetta;

45 loop_lbl: LOOP
    FETCH curRicetta INTO NomeSede, NomeRicetta, RicettaNovita;
    IF Finito THEN
        LEAVE loop_lbl;
    END IF;

50 SET ClientiPrenotazioni = (SELECT COALESCE(CP.Numero, 0)
                                FROM MV_ClientiPrenotazione CP
                                WHERE CP.Sede = NomeSede
                                      AND CP.'Data' BETWEEN CURRENT_DATE
55                                     AND CURRENT_DATE + INTERVAL 6 DAY);

    SET MediaSenzaPrenotazione = (
        SELECT CEIL(COALESCE(AVG(CL.SenzaPrenotazione), 0)/4) AS Media
        FROM Clienti_Log CL
60        WHERE CL.Sede = NomeSede
              AND CL.Mese = MONTH(CURRENT_DATE)
              AND CL.Anno <> YEAR(CURRENT_DATE)
    );

65 SET StimaClienti = ClientiPrenotazioni + MediaSenzaPrenotazione;

    IF RicettaNovita THEN
        SET StimaOrdini = (SELECT CEIL(StimaClienti * 0.33));
    ELSE
70        SET StimaOrdini = (
            SELECT (COALESCE(CEIL(MV.TotOrdini / MV.Compare), 0)
                    + StimaClienti * 0.1) AS StimaOrdini
            FROM MV_OrdiniRicetta MV
            WHERE MV.Sede = NomeSede AND MV.Ricetta = NomeRicetta
75        );
    END IF;

    IF StimaOrdini < 5 THEN
        SET StimaOrdini = 5;
80 END IF;

```

```

85      INSERT INTO Report_Ordinativi(Sede, Ingrediente, Quantita)
      SELECT NomeSede, F.Ingrediente, SUM(F.Dose)*StimaOrdini AS Qta
      FROM Fase F
      WHERE F.Ricetta = NomeRicetta
      ON DUPLICATE KEY UPDATE Quantita = Quantita + VALUES(Quantita);
      END LOOP loop_lbl;

      CLOSE curRicetta;
90  END;$$
DELIMITER ;

```

10.4 Analisi dei consumi e degli sprechi

La seguente procedura calcola, per ogni sede e per ogni ingrediente, la differenza tra la materia prima scaricata dai magazzini della sede e quella effettivamente impiegata nella produzione dei piatti ordinati dai clienti, in un certo periodo di tempo.

Per far ciò la procedura itera per tutte le sedi e per tutti gli ingredienti e prende dalla tabella `Scarichi_Log` la quantità totale di ingrediente scaricata dai magazzini della sede. Poi fa uso di una query molto complessa per calcolare la quantità di ingrediente usata nella produzione di tutti i piatti ordinati che richiedono tale ingrediente.

La query prende tutti i piatti prodotti dalla sede nel periodo di interesse specificato. Per ogni piatto prende tutte le fasi di preparazione che prevedono l'aggiunta dell'ingrediente preso in analisi. Per ottenere le fasi di preparazione effettivamente impiegate nella produzione del piatto, la query deve anche considerare le variazioni: la prima delle due condizioni nel `WHERE` che coinvolge `F.ID` controlla che la fase non sia presente come `FaseVecchia` all'interno di una qualche `ModificaFase` di una qualche `Variazione` scelta dal cliente per quel piatto; la seconda controlla che la fase non sia presente come `FaseNuova` all'interno di una qualche `ModificaFase` di una qualche `Variazione` **non** scelta dal cliente per quel piatto. Il *result-set* risultante contiene tutte le fasi di aggiunta dell'ingrediente preso in analisi con tutte le informazioni sul piatto, la comanda e la ricetta alla quale si riferiscono. Quindi le stesse fasi che aggiungono tale ingrediente sono ripetute per il numero di piatti prodotti per ogni ricetta. Sommando tutte le dosi si ottiene la quantità totale di ingrediente impiegato nella produzione dei piatti.

La tabella `Report_Sprechi` riporta, per ogni sede e per ogni ingrediente, la quantità di materia prima sprecata.

```

5  CREATE TABLE Report_Sprechi
    (
      Sede                VARCHAR(45) NOT NULL,
      Ingrediente          VARCHAR(45) NOT NULL,
      Spreco               INT UNSIGNED NOT NULL,

```

```

PRIMARY KEY (Sede, Ingrediente),
FOREIGN KEY (Sede)
    REFERENCES Sede(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
10 FOREIGN KEY (Ingrediente)
    REFERENCES Ingrediente(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE
15 ) ENGINE = InnoDB;

DELIMITER $$

CREATE PROCEDURE AnalizzaSprechi(IN inizio TIMESTAMP, IN fine TIMESTAMP)
20 NOT DETERMINISTIC MODIFIES SQL DATA
BEGIN
    IF inizio IS NULL THEN
        SET inizio = '1970-01-01 00:00:01';
    END IF;
25 IF fine IS NULL THEN
        SET fine = CURRENT_TIMESTAMP;
    END IF;

    TRUNCATE TABLE Report_Sprechi;

30 BEGIN
    DECLARE NomeSede VARCHAR(45);
    DECLARE NomeIngrediente VARCHAR(45);
    DECLARE Scaricata INT;
35 DECLARE Quantita INT;
    DECLARE Finito BOOL DEFAULT FALSE;
    DECLARE curScarichi CURSOR FOR
        SELECT SL.Sede, SL.Ingrediente, COALESCE(SUM(SL.Quantita), 0) AS Qta
        FROM Scarichi_Log SL
40 WHERE SL.`Timestamp` BETWEEN inizio AND fine
        GROUP BY SL.Sede, SL.Ingrediente;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET Finito = TRUE;

    OPEN curScarichi;

45 loop_lbl: LOOP
    FETCH curScarichi INTO NomeSede, NomeIngrediente, Scaricata;
    IF Finito THEN
        LEAVE loop_lbl;
50 END IF;

```

```

55      SET Quantita = (
      SELECT COALESCE(SUM(F.Dose), 0) AS Quantita
      FROM Fase F
           INNER JOIN Ricetta R ON F.Ricetta = R.Nome
           INNER JOIN Piatto P ON R.Nome = P.Ricetta
           INNER JOIN Comanda C ON P.Comanda = C.ID
      WHERE C.Sede = NomeSede
           AND F.Ingrediente = NomeIngrediente
60      AND C.'Timestamp' BETWEEN inizio AND fine
           AND F.ID NOT IN (SELECT MF.FaseVecchia
                           FROM ModificaFase MF
                               INNER JOIN Variazione V
                                   ON MF.Variazione = V.ID
                               INNER JOIN
65                               (SELECT M.Variazione
                                FROM Modifica M
                                WHERE M.Piatto = P.ID) AS D
                                   ON V.ID = D.Variazione)
           AND F.ID NOT IN (SELECT MFN.FaseNuova
                           FROM ModificaFase MFN
                               INNER JOIN Variazione VA
                                   ON MFN.Variazione = VA.ID
                               WHERE VA.ID NOT IN (SELECT MO.Variazione
65                                                    FROM Modifica MO
                                                    WHERE MO.Piatto = P.ID))
      );

80      INSERT INTO Report_Sprechi(Sede, Ingrediente, Spreco)
      VALUES (NomeSede, NomeIngrediente, Scaricata - Quantita);

      END LOOP loop_lbl;

      CLOSE curScarichi;
85      END;
END;$$
DELIMITER ;

```

10.5 Qualità del take-away

Il seguente evento calcola, per ogni pony (e quindi anche per ogni sede), la differenza tra i tempi medi di andata e ritorno delle consegne effettuate dal pony e i tempi medi di tutte le consegne effettuate.

La tabella `Report_TakeAway` registra queste informazioni e mantiene anche una *classifica* dei pony più veloci. I pony nelle ultime posizioni sono i più lenti.

```

CREATE TABLE Report_TakeAway
(
  Posizione          INT UNSIGNED NOT NULL,
  Sede               VARCHAR(45) NOT NULL,
  Pony               INT UNSIGNED NOT NULL,
  DeltaTempoAndata    TIME NOT NULL,
  DeltaTempoRitorno   TIME NOT NULL,
  PRIMARY KEY (Posizione),
  UNIQUE KEY (Sede, Pony),
  FOREIGN KEY (Sede, Pony)
    REFERENCES Pony(Sede, ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
) ENGINE = InnoDB;

DELIMITER $$

CREATE EVENT aggiorna_Report_TakeAway
ON SCHEDULE
EVERY 1 WEEK
STARTS TIMESTAMP(CURRENT_DATE) + INTERVAL 1 DAY + INTERVAL 3 HOUR
ON COMPLETION PRESERVE
DO
BEGIN
  DECLARE TempoMedioAndata INT;
  DECLARE TempoMedioRitorno INT;

  TRUNCATE TABLE Report_TakeAway;

  SELECT CEIL(AVG(TIMESTAMPDIFF(SECOND, C.Partenza, C.Arribo))) AS TMAAndata,
         CEIL(AVG(TIMESTAMPDIFF(SECOND, C.Arribo, C.Ritorno))) AS TMRitorno
  INTO TempoMedioAndata, TempoMedioRitorno
  FROM Consegna C
  WHERE C.Ritorno IS NOT NULL;

  IF TempoMedioAndata IS NULL OR TempoMedioRitorno IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Dati insufficienti per la generazione di '
                      'Report_TakeAway.';
  END IF;

  INSERT INTO Report_TakeAway(Posizione, Sede, Pony, DeltaTempoAndata,
                              DeltaTempoRitorno)

```



```
45  SELECT @row_number := @row_number + 1 AS Posizione, D.*
    INTO Report_TakeAway
    FROM (SELECT @row_number := 0) AS N,
        (
            SELECT P.Sede, P.ID AS Pony,
                SEC_TO_TIME(
50             CEIL(AVG(TIMESTAMPDIFF(SECOND, C.Partenza, C.Arrivo))) -
                TempoMedioAndata
            ) AS DeltaTempoAndata,
                SEC_TO_TIME(
55             CEIL(AVG(TIMESTAMPDIFF(SECOND, C.Arrivo, C.Ritorno))) -
                TempoMedioRitorno
            ) AS DeltaTempoRitorno
            FROM Pony P INNER JOIN Consegna C
            WHERE C.Ritorno IS NOT NULL
            GROUP BY P.Sede, P.ID
60        ) AS D
    ORDER BY (D.DeltaTempoAndata + D.DeltaTempoRitorno) ASC;
END; $$
DELIMITER ;
```


Capitolo 11

Implementazione MySQL

In questo Capitolo viene presentato lo *script MySQL che implementa il database*.

Si noti che sono stati aggiunti alcuni *trigger* (rispetto a quelli presentati nel paragrafo 4.3 a pagina 36) e altri sono stati modificati. Sono stati anche aggiunti i codici per *simulare* il comportamento dell'AUTO_INCREMENT. Infatti l'*engine InnoDB* non permette di avere l'AUTO_INCREMENT sull'attributo ID in tabelle come **Magazzino**. Infatti in **Magazzino** le tuple dovrebbero assumere la seguente forma:

```
$ SELECT * FROM Magazzino;
```

```
+-----+-----+
| Sede  | ID  |
+-----+-----+
| SedeA | 1   |
| SedeA | 2   |
| SedeA | 3   |
| SedeB | 1   |
| SedeB | 2   |
| SedeB | 3   |
| SedeC | 1   |
| SedeC | 2   |
+-----+-----+
```

Impostando l'AUTO_INCREMENT sull'attributo ID si avrebbe invece:

```
$ SELECT * FROM Magazzino;
```

```
+----+-----+
| ID | Sede  |
+----+-----+
| 1  | SedeA |
| 2  | SedeA |
```

```

| 3 | SedeA |
| 4 | SedeB |
| 5 | SedeB |
| 6 | SedeB |
| 7 | SedeC |
| 8 | SedeC |
+----+-----+

```

L'engine MyISAM permette di impostare l'AUTO_INCREMENT su una colonna *secondaria*, ma non permette i *vincoli di integrità referenziale* che sono fondamentali nel database di questo progetto. Dobbiamo quindi utilizzare InnoDB e *simulare* l'AUTO_INCREMENT mediante trigger.

Per brevità, gli INSERT INTO che popolano le tabelle non sono riportati in questo documento, ma sono comunque disponibili nello *script in allegato*.

```

SELECT "Creazione database e impostazione variabili."
  AS "***** START - FASE 1 *****";

DROP SCHEMA IF EXISTS unipi_project;
5 CREATE SCHEMA unipi_project DEFAULT CHARACTER SET utf8;
  USE unipi_project;

SET GLOBAL event_scheduler = on;

10
-- TABLES
SELECT "Creazione tabelle." AS "***** FASE 2 *****";

CREATE TABLE Sede
15 (
  Nome          VARCHAR(45) NOT NULL,
  Citta         VARCHAR(45) NOT NULL,
  CAP          INT(5) UNSIGNED ZEROFILL NOT NULL,
  Via          VARCHAR(45) NOT NULL,
20  NumeroCivico INT UNSIGNED NOT NULL,
  PRIMARY KEY (Nome),
  UNIQUE KEY (Citta, Via, NumeroCivico)
) ENGINE = InnoDB;

25 CREATE TABLE Magazzino
  (
    Sede          VARCHAR(45) NOT NULL,
    ID            INT UNSIGNED NOT NULL,
    PRIMARY KEY (Sede, ID),
30  FOREIGN KEY (Sede)

```

```

    REFERENCES Sede(Nome)
    ON DELETE NO ACTION
    ON UPDATE CASCADE
) ENGINE = InnoDB;
35
CREATE TABLE Ingrediente
(
    Nome                VARCHAR(45) NOT NULL,
    Provenienza         VARCHAR(45) NOT NULL,
    40  TipoProduzione   VARCHAR(45) NOT NULL,
    Genere              VARCHAR(45) NOT NULL,
    Allergene           BOOL NOT NULL DEFAULT FALSE,
    PRIMARY KEY (Nome)
) ENGINE = InnoDB;
45
CREATE TABLE Lotto
(
    Codice              VARCHAR(32) NOT NULL,
    Ingrediente         VARCHAR(45) NOT NULL,
    50  Scadenza         DATE NOT NULL,
    PRIMARY KEY (Codice),
    FOREIGN KEY (Ingrediente)
        REFERENCES Ingrediente(Nome)
        ON DELETE NO ACTION
    55  ON UPDATE CASCADE
) ENGINE = InnoDB;

CREATE TABLE Confezione
(
    60  CodiceLotto      VARCHAR(32) NOT NULL,
    Numero             INT UNSIGNED NOT NULL,
    Peso              INT UNSIGNED NOT NULL,
    Prezzo            DECIMAL(8,2) UNSIGNED NOT NULL,
    DataAcquisto       DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    65  DataArrivo       DATETIME,
    DataCarico         DATETIME,
    Sede              VARCHAR(45) NOT NULL,
    Magazzino         INT UNSIGNED NOT NULL,
    Collocazione       VARCHAR(45),
    70  Aspetto         BOOL COMMENT 'TRUE = ok; FALSE = danneggiata',
    Stato             ENUM('completa', 'parziale', 'in uso', 'in ordine')
                    NOT NULL DEFAULT 'in ordine',
    PRIMARY KEY (CodiceLotto, Numero),
    FOREIGN KEY (CodiceLotto)
    75  REFERENCES Lotto(Codice)
    ON DELETE NO ACTION

```

```

        ON UPDATE CASCADE,
FOREIGN KEY (Sede, Magazzino)
REFERENCES Magazzino(Sede, ID)
ON DELETE NO ACTION
ON UPDATE CASCADE
80 ) ENGINE = InnoDB;

CREATE TABLE Strumento
85 (
    Nome VARCHAR(45) NOT NULL,
    PRIMARY KEY (Nome)
) ENGINE = InnoDB;

CREATE TABLE Funzione
90 (
    Strumento VARCHAR(45) NOT NULL,
    Nome VARCHAR(45) NOT NULL,
    PRIMARY KEY (Strumento, Nome),
95 FOREIGN KEY (Strumento)
    REFERENCES Strumento(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE
) ENGINE = InnoDB;

100 CREATE TABLE Cucina
(
    Sede VARCHAR(45) NOT NULL,
    Strumento VARCHAR(45) NOT NULL,
105 Quantita INT UNSIGNED NOT NULL DEFAULT 1,
    PRIMARY KEY (Sede, Strumento),
    FOREIGN KEY (Sede)
        REFERENCES Sede(Nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
110 FOREIGN KEY (Strumento)
        REFERENCES Strumento(Nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
115 ) ENGINE = InnoDB;

CREATE TABLE Menu
(
    ID INT UNSIGNED NOT NULL AUTO_INCREMENT,
120 Sede VARCHAR(45) NOT NULL,
    DataInizio DATE NOT NULL,
    DataFine DATE NOT NULL,

```

```

PRIMARY KEY (ID),
UNIQUE KEY (Sede, DataInizio),
125 UNIQUE KEY (Sede, DataFine),
FOREIGN KEY (Sede)
REFERENCES Sede(Nome)
ON DELETE CASCADE
ON UPDATE CASCADE
130 ) ENGINE = InnoDB;

CREATE TABLE Ricetta
(
Nome VARCHAR(45) NOT NULL,
135 Testo TEXT NOT NULL,
PRIMARY KEY (Nome)
) ENGINE = InnoDB;

CREATE TABLE Elenco
140 (
Menu INT UNSIGNED NOT NULL,
Ricetta VARCHAR(45) NOT NULL,
Novita BOOL NOT NULL DEFAULT TRUE,
PRIMARY KEY (Menu, Ricetta),
145 FOREIGN KEY (Menu)
REFERENCES Menu(ID)
ON DELETE CASCADE
ON UPDATE CASCADE,
FOREIGN KEY (Ricetta)
150 REFERENCES Ricetta(Nome)
ON DELETE CASCADE
ON UPDATE CASCADE
) ENGINE = InnoDB;

155 CREATE TABLE Fase
(
ID INT UNSIGNED NOT NULL AUTO_INCREMENT,
Ricetta VARCHAR(45) NOT NULL,
160 Ingrediente VARCHAR(45),
Dose INT UNSIGNED,
Primario BOOL,
Strumento VARCHAR(45),
Testo TEXT,
Durata TIME,
165 PRIMARY KEY (ID),
FOREIGN KEY (Ricetta)
REFERENCES Ricetta(Nome)
ON DELETE CASCADE

```

```

170         ON UPDATE CASCADE,
        FOREIGN KEY (Ingrediente)
            REFERENCES Ingrediente(Nome)
            ON DELETE SET NULL
            ON UPDATE CASCADE,
175         FOREIGN KEY (Strumento)
            REFERENCES Strumento(Nome)
            ON DELETE SET NULL
            ON UPDATE CASCADE
    ) ENGINE = InnoDB;

180 CREATE TABLE SequenzaFasi
    (
        Fase                                INT UNSIGNED NOT NULL,
        FasePrecedente                    INT UNSIGNED NOT NULL,
        PRIMARY KEY (Fase, FasePrecedente),
185         FOREIGN KEY (Fase)
            REFERENCES Fase(ID)
            ON DELETE CASCADE
            ON UPDATE CASCADE,
        FOREIGN KEY (FasePrecedente)
190         REFERENCES Fase(ID)
            ON DELETE CASCADE
            ON UPDATE CASCADE
    ) ENGINE = InnoDB;

195 CREATE TABLE Sala
    (
        Sede                                VARCHAR(45) NOT NULL,
        Numero                            INT UNSIGNED NOT NULL,
        PRIMARY KEY (Sede, Numero),
200         FOREIGN KEY (Sede)
            REFERENCES Sede(Nome)
            ON DELETE NO ACTION
            ON UPDATE CASCADE
    ) ENGINE = InnoDB;

205 CREATE TABLE Tavolo
    (
        Sede                                VARCHAR(45) NOT NULL,
        Sala                                INT UNSIGNED NOT NULL,
210         Numero                            INT UNSIGNED NOT NULL,
        Posti                            INT UNSIGNED NOT NULL,
        PRIMARY KEY (Sede, Sala, Numero),
        FOREIGN KEY (Sede, Sala)
            REFERENCES Sala(Sede, Numero)
    )

```



```

215         ON DELETE NO ACTION
           ON UPDATE CASCADE
    ) ENGINE = InnoDB;

CREATE TABLE Account
220 (
    Username          VARCHAR(20) NOT NULL,
    Email             VARCHAR(100) NOT NULL,
    'Password'        CHAR(32) NOT NULL,
    Nome              VARCHAR(45) NOT NULL,
225    Cognome          VARCHAR(45) NOT NULL,
    Sesso             CHAR(1) NOT NULL,
    Citta             VARCHAR(45) NOT NULL,
    CAP               INT(5) UNSIGNED ZEROFILL NOT NULL,
    Via               VARCHAR(45) NOT NULL,
230    NumeroCivico     INT UNSIGNED NOT NULL,
    Telefono          VARCHAR(16) NOT NULL,
    PuoPrenotare      BOOL NOT NULL DEFAULT TRUE,
    PRIMARY KEY (Username),
    UNIQUE KEY (Email),
235    UNIQUE KEY (Telefono),
    UNIQUE KEY (Nome, Cognome, Citta, Via, NumeroCivico)
    ) ENGINE = InnoDB;

CREATE TABLE Comanda
240 (
    ID                INT UNSIGNED NOT NULL AUTO_INCREMENT,
    'Timestamp'       TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Sede              VARCHAR(45) NOT NULL,
    Sala              INT UNSIGNED,
245    Tavolo            INT UNSIGNED,
    Account           VARCHAR(20),
    PRIMARY KEY (ID),
    UNIQUE KEY ('Timestamp', Sede, Sala, Tavolo),
    UNIQUE KEY ('Timestamp', Account),
250    FOREIGN KEY (Sede)
        REFERENCES Sede(Nome)
        ON DELETE NO ACTION
        ON UPDATE CASCADE,
    FOREIGN KEY (Sede, Sala, Tavolo)
255    REFERENCES Tavolo(Sede, Sala, Numero)
        ON DELETE NO ACTION
        ON UPDATE CASCADE,
    FOREIGN KEY (Account)
        REFERENCES Account(Username)
260    ON DELETE SET NULL

```

```

        ON UPDATE CASCADE
    ) ENGINE = InnoDB;

CREATE TABLE Piatto
265 (
    ID                INT UNSIGNED NOT NULL AUTO_INCREMENT,
    Comanda           INT UNSIGNED NOT NULL,
    Ricetta           VARCHAR(45) NOT NULL,
    InizioPreparazione  TIMESTAMP NULL DEFAULT NULL
270     ON UPDATE CURRENT_TIMESTAMP,
    Stato             ENUM('attesa', 'in preparazione', 'servizio')
                        NOT NULL DEFAULT 'attesa',

    PRIMARY KEY (ID),
    FOREIGN KEY (Comanda)
275     REFERENCES Comanda(ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (Ricetta)
        REFERENCES Ricetta(Nome)
280     ON DELETE NO ACTION
        ON UPDATE CASCADE
    ) ENGINE = InnoDB;

CREATE TABLE Variazione
285 (
    ID                INT UNSIGNED NOT NULL AUTO_INCREMENT,
    Ricetta           VARCHAR(45) NOT NULL,
    Nome              VARCHAR(45),
    Account           VARCHAR(20),
290     PRIMARY KEY (ID),
    FOREIGN KEY (Ricetta)
        REFERENCES Ricetta(Nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
295     FOREIGN KEY (Account)
        REFERENCES Account(Username)
        ON DELETE SET NULL
        ON UPDATE CASCADE
    ) ENGINE = InnoDB;

300 CREATE TABLE ModificaFase
    (
        Variazione     INT UNSIGNED NOT NULL,
        ID              INT UNSIGNED NOT NULL,
305     FaseVecchia     INT UNSIGNED,
        FaseNuova       INT UNSIGNED,

```

```

PRIMARY KEY (Variazione, ID),
FOREIGN KEY (Variazione)
  REFERENCES Variazione(ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE,
FOREIGN KEY (FaseVecchia)
  REFERENCES Fase(ID)
  ON DELETE NO ACTION
  ON UPDATE CASCADE,
FOREIGN KEY (FaseNuova)
  REFERENCES Fase(ID)
  ON DELETE NO ACTION
  ON UPDATE CASCADE
) ENGINE = InnoDB;

CREATE TABLE Modifica
(
  Piatto          INT UNSIGNED NOT NULL,
  Variazione      INT UNSIGNED NOT NULL,
  PRIMARY KEY (Piatto, Variazione),
  FOREIGN KEY (Piatto)
    REFERENCES Piatto(ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (Variazione)
    REFERENCES Variazione(ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
) ENGINE = InnoDB;

CREATE TABLE Pony
(
  Sede            VARCHAR(45) NOT NULL,
  ID              INT UNSIGNED NOT NULL,
  Ruote           BOOL NOT NULL DEFAULT FALSE
                COMMENT 'TRUE = 4 ruote; FALSE = 2 ruote',
  Stato           BOOL NOT NULL DEFAULT TRUE
                COMMENT 'TRUE = libero; FALSE = occupato',
  PRIMARY KEY (Sede, ID),
  FOREIGN KEY (Sede)
    REFERENCES Sede(Nome)
    ON DELETE NO ACTION
    ON UPDATE CASCADE
) ENGINE = InnoDB;

CREATE TABLE Consegna

```

```

(
  Comanda          INT UNSIGNED NOT NULL,
355  Sede            VARCHAR(45) NOT NULL,
  Pony             INT UNSIGNED NOT NULL,
  Partenza         DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  Arrivo           DATETIME ON UPDATE CURRENT_TIMESTAMP,
  Ritorno          DATETIME,
360  PRIMARY KEY (Comanda),
  UNIQUE KEY (Pony, Partenza),
  FOREIGN KEY (Comanda)
    REFERENCES Comanda(ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
365  FOREIGN KEY (Sede, Pony)
    REFERENCES Pony(Sede, ID)
    ON DELETE NO ACTION
    ON UPDATE CASCADE
370 ) ENGINE = InnoDB;

CREATE TABLE Prenotazione
(
  ID               INT UNSIGNED NOT NULL AUTO_INCREMENT,
375  Sede           VARCHAR(45) NOT NULL,
  'Data'          DATETIME NOT NULL,
  Numero          INT UNSIGNED NOT NULL,
  Account          VARCHAR(45),
  Nome            VARCHAR(45),
380  Telefono       VARCHAR(16),
  Sala            INT UNSIGNED NOT NULL,
  Tavolo          INT UNSIGNED,
  Descrizione     TEXT,
  Approvato       BOOL,
385  PRIMARY KEY (ID),
  UNIQUE KEY (Tavolo, 'Data'),
  FOREIGN KEY (Sede, Sala, Tavolo)
    REFERENCES Tavolo(Sede, Sala, Numero)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
390  FOREIGN KEY (Sede, Sala)
    REFERENCES Sala(Sede, Numero)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
395  FOREIGN KEY (Account)
    REFERENCES Account(Username)
    ON DELETE SET NULL
    ON UPDATE CASCADE

```

```

400 ) ENGINE = InnoDB;

CREATE TABLE Proposta
(
    ID                      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    Account                 VARCHAR(20) NOT NULL,
405 Nome                   VARCHAR(45) NOT NULL,
    Procedimento           TEXT,
    PRIMARY KEY (ID),
    UNIQUE KEY (Account, Nome),
    FOREIGN KEY (Account)
410 REFERENCES Account(Username)
        ON DELETE NO ACTION
        ON UPDATE CASCADE
) ENGINE = InnoDB;

415 CREATE TABLE Composizione
(
    Proposta               INT UNSIGNED NOT NULL,
    Ingrediente            VARCHAR(45) NOT NULL,
    PRIMARY KEY (Proposta, Ingrediente),
420 FOREIGN KEY (Proposta)
        REFERENCES Proposta(ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (Ingrediente)
425 REFERENCES Ingrediente(Nome)
        ON DELETE NO ACTION
        ON UPDATE CASCADE
) ENGINE = InnoDB;

430 CREATE TABLE Gradimento
(
    ID                      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    Account                 VARCHAR(20) NOT NULL,
    Proposta               INT UNSIGNED,
435 Suggestimento          INT UNSIGNED,
    Punteggio              TINYINT(1) UNSIGNED NOT NULL,
    PRIMARY KEY (ID),
    UNIQUE KEY (Account, Proposta, Suggestimento),
    FOREIGN KEY (Account)
440 REFERENCES Account(Username)
        ON DELETE NO ACTION
        ON UPDATE CASCADE,
    FOREIGN KEY (Proposta)
        REFERENCES Proposta(ID)

```

```

445         ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (Suggerimento)
        REFERENCES Variazione(ID)
        ON DELETE CASCADE
450     ON UPDATE CASCADE
) ENGINE = InnoDB;

CREATE TABLE Recensione
(
455     ID                INT UNSIGNED NOT NULL AUTO_INCREMENT,
    Account             VARCHAR(20) NOT NULL,
    Sede                VARCHAR(45) NOT NULL,
    Ricetta             VARCHAR(45) NOT NULL,
    Testo               TEXT NOT NULL,
460     Giudizio          TINYINT(1) UNSIGNED NOT NULL,
    VeridicitaTotale    INT UNSIGNED NOT NULL DEFAULT 0,
    AccuratezzaTotale   INT UNSIGNED NOT NULL DEFAULT 0,
    NumeroValutazioni   INT UNSIGNED NOT NULL DEFAULT 0,
    PRIMARY KEY (ID),
465     UNIQUE KEY (Account, Sede, Ricetta),
    FOREIGN KEY (Account)
        REFERENCES Account(Username)
        ON DELETE NO ACTION
        ON UPDATE CASCADE,
470     FOREIGN KEY (Sede)
        REFERENCES Sede(Nome)
        ON DELETE NO ACTION
        ON UPDATE CASCADE,
    FOREIGN KEY (Ricetta)
475     REFERENCES Ricetta(Nome)
        ON DELETE NO ACTION
        ON UPDATE CASCADE
) ENGINE = InnoDB;

480 CREATE TABLE Valutazione
(
    Account             VARCHAR(20) NOT NULL,
    Recensione          INT UNSIGNED NOT NULL,
    Veridicita          TINYINT(1) UNSIGNED NOT NULL,
485     Accuratezza        TINYINT(1) UNSIGNED NOT NULL,
    Testo               TEXT NOT NULL,
    PRIMARY KEY (Account, Recensione),
    FOREIGN KEY (Account)
        REFERENCES Account(Username)
490     ON DELETE NO ACTION

```

```

        ON UPDATE CASCADE,
        FOREIGN KEY (Recensione)
        REFERENCES Recensione(ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
495 ) ENGINE = InnoDB;

CREATE TABLE Domanda
(
500   ID                      INT UNSIGNED NOT NULL AUTO_INCREMENT,
      Sede                  VARCHAR(45) NOT NULL,
      Testo                 VARCHAR(1024) NOT NULL,
      PRIMARY KEY (ID),
      FOREIGN KEY (Sede)
505   REFERENCES Sede(Nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE = InnoDB;

510 CREATE TABLE Risposta
(
      Domanda               INT UNSIGNED NOT NULL,
      Numero                INT UNSIGNED NOT NULL,
      Testo                 VARCHAR(1024) NOT NULL,
515   Efficienza             TINYINT(1) UNSIGNED NOT NULL,
      PRIMARY KEY (Domanda, Numero),
      FOREIGN KEY (Domanda)
        REFERENCES Domanda(ID)
        ON DELETE CASCADE
520   ON UPDATE CASCADE
) ENGINE = InnoDB;

CREATE TABLE QuestionarioSvolto
(
525   Recensione             INT UNSIGNED NOT NULL,
      Domanda               INT UNSIGNED NOT NULL,
      Risposta              INT UNSIGNED NOT NULL,
      PRIMARY KEY (Recensione, Domanda),
      FOREIGN KEY (Recensione)
530   REFERENCES Recensione(ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
      FOREIGN KEY (Domanda, Risposta)
        REFERENCES Risposta(Domanda, Numero)
535   ON DELETE CASCADE
        ON UPDATE CASCADE

```

```

) ENGINE = InnoDB;

540 -- LOG TABLES
SELECT "Creazione tabelle di log." AS "***** FASE 3 *****";

CREATE TABLE Clienti_Log
(
545     Sede                VARCHAR(45) NOT NULL,
     Anno                INT UNSIGNED NOT NULL,
     Mese                INT UNSIGNED NOT NULL,
     SenzaPrenotazione    INT UNSIGNED NOT NULL DEFAULT 0,
     PRIMARY KEY (Sede, Anno, Mese),
550     FOREIGN KEY (Sede)
         REFERENCES Sede(Nome)
         ON DELETE CASCADE
         ON UPDATE CASCADE
) ENGINE = InnoDB;

555 CREATE TABLE Scarichi_Log
(
     ID                INT UNSIGNED NOT NULL AUTO_INCREMENT,
     Sede                VARCHAR(45) NOT NULL,
560     Magazzino          INT UNSIGNED NOT NULL,
     Ingrediente         VARCHAR(45) NOT NULL,
     'Timestamp'         TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
     Quantita            INT UNSIGNED NOT NULL DEFAULT 0,
     PRIMARY KEY (ID),
565     FOREIGN KEY (Sede, Magazzino)
         REFERENCES Magazzino(Sede, ID)
         ON DELETE CASCADE
         ON UPDATE CASCADE,
     FOREIGN KEY (Ingrediente)
570     REFERENCES Ingrediente(Nome)
         ON DELETE CASCADE
         ON UPDATE CASCADE
) ENGINE = InnoDB;

575 -- REPORT TABLES
SELECT "Creazione tabelle di report." AS "***** FASE 4 *****";

CREATE TABLE Report_PiattiDaAggiungere
580 (
     Posizione          INT UNSIGNED NOT NULL,
     Sede                VARCHAR(45) NOT NULL,

```



```

Ricetta          VARCHAR(45) NOT NULL,
PRIMARY KEY(Posizione),
585 UNIQUE KEY (Sede, Ricetta),
FOREIGN KEY (Sede)
    REFERENCES Sede(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
590 FOREIGN KEY (Ricetta)
    REFERENCES Ricetta(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE
) ENGINE = InnoDB;
595
CREATE TABLE Report_PiattiPreferiti
(
    Posizione          INT UNSIGNED NOT NULL,
    Sede               VARCHAR(45) NOT NULL,
600 Ricetta            VARCHAR(45) NOT NULL,
    GiudizioTotale     INT UNSIGNED NOT NULL,
    NumeroRecensioni   INT UNSIGNED NOT NULL,
    PRIMARY KEY (Posizione),
    UNIQUE KEY (Sede, Ricetta),
605 FOREIGN KEY (Sede)
    REFERENCES Sede(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY (Ricetta)
610 REFERENCES Ricetta(Nome)
    ON DELETE CASCADE
    ON UPDATE CASCADE
) ENGINE = InnoDB;

615 CREATE TABLE Report_VenditePiatti
(
    Posizione          INT UNSIGNED NOT NULL,
    Sede               VARCHAR(45) NOT NULL,
    Ricetta            VARCHAR(45) NOT NULL,
620 Vendite            INT UNSIGNED NOT NULL,
    PRIMARY KEY (Posizione),
    UNIQUE KEY (Sede, Ricetta),
    FOREIGN KEY (Sede)
    REFERENCES Sede(Nome)
625 ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY (Ricetta)
    REFERENCES Ricetta(Nome)

```

```

630         ON DELETE CASCADE
        ON UPDATE CASCADE
    ) ENGINE = InnoDB;

CREATE TABLE Report_SuggerimentiMigliori
(
635     Posizione            INT UNSIGNED NOT NULL,
        Suggerimento        INT UNSIGNED NOT NULL,
        GradimentoTotale    INT UNSIGNED NOT NULL,
        NumeroGradimenti    INT UNSIGNED NOT NULL,
        PRIMARY KEY (Posizione),
640     UNIQUE KEY (Suggerimento),
        FOREIGN KEY (Suggerimento)
            REFERENCES Variazione(ID)
            ON DELETE CASCADE
            ON UPDATE CASCADE
645 ) ENGINE = InnoDB;

CREATE TABLE Report_ProposteMigliori
(
650     Posizione            INT UNSIGNED NOT NULL,
        Proposta            INT UNSIGNED NOT NULL,
        GradimentoTotale    INT UNSIGNED NOT NULL,
        NumeroGradimenti    INT UNSIGNED NOT NULL,
        PRIMARY KEY (Posizione),
        UNIQUE KEY (Proposta),
655     FOREIGN KEY (Proposta)
            REFERENCES Proposta(ID)
            ON DELETE CASCADE
            ON UPDATE CASCADE
    ) ENGINE = InnoDB;

660 CREATE TABLE Report_Ordinativi
(
        Sede                VARCHAR(45) NOT NULL,
        Ingrediente          VARCHAR(45) NOT NULL,
665     Quantita             INT UNSIGNED NOT NULL,
        PRIMARY KEY (Sede, Ingrediente),
        FOREIGN KEY (Sede)
            REFERENCES Sede(Nome)
            ON DELETE CASCADE
670     ON UPDATE CASCADE,
        FOREIGN KEY (Ingrediente)
            REFERENCES Ingrediente(Nome)
            ON DELETE CASCADE
            ON UPDATE CASCADE

```

```

675 | ) ENGINE = InnoDB;

CREATE TABLE Report_Sprechi
(
  Sede                VARCHAR(45) NOT NULL,
680 | Ingrediente         VARCHAR(45) NOT NULL,
  Spreco              INT UNSIGNED NOT NULL,
  PRIMARY KEY (Sede, Ingrediente),
  FOREIGN KEY (Sede)
    REFERENCES Sede(Nome)
685 |     ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (Ingrediente)
    REFERENCES Ingrediente(Nome)
    ON DELETE CASCADE
690 |     ON UPDATE CASCADE
) ENGINE = InnoDB;

CREATE TABLE Report_TakeAway
(
695 | Posizione           INT UNSIGNED NOT NULL,
  Sede                VARCHAR(45) NOT NULL,
  Pony                INT UNSIGNED NOT NULL,
  DeltaTempoAndata    TIME NOT NULL,
  DeltaTempoRitorno   TIME NOT NULL,
700 | PRIMARY KEY (Posizione),
  UNIQUE KEY (Sede, Pony),
  FOREIGN KEY (Sede, Pony)
    REFERENCES Pony(Sede, ID)
    ON DELETE CASCADE
705 |     ON UPDATE CASCADE
) ENGINE = InnoDB;

-- MATERIALIZED VIEWS
710 | SELECT "Creazione materialized views." AS "***** FASE 5 *****";

CREATE TABLE MV_OrdiniRicetta
(
715 | Sede                VARCHAR(45) NOT NULL,
  Ricetta             VARCHAR(45) NOT NULL,
  Compare             INT UNSIGNED NOT NULL DEFAULT 1,
  TotOrdini           INT UNSIGNED NOT NULL,
  PRIMARY KEY (Sede, Ricetta),
  FOREIGN KEY (Sede)
720 |     REFERENCES Sede(Nome)

```

```

        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (Ricetta)
        REFERENCES Ricetta(Nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
725 ) ENGINE = InnoDB;

CREATE TABLE MV_MenuCorrente
730 (
    Sede                VARCHAR(45) NOT NULL,
    Ricetta              VARCHAR(45) NOT NULL,
    Novita               BOOL NOT NULL DEFAULT FALSE,
    PRIMARY KEY (Sede, Ricetta),
735 FOREIGN KEY (Sede)
        REFERENCES Sede(Nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (Ricetta)
740 REFERENCES Ricetta(Nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
    ) ENGINE = InnoDB;

CREATE TABLE MV_ClientiPrenotazione
745 (
    Sede                VARCHAR(45) NOT NULL,
    'Data'              DATE NOT NULL,
    Numero              INT UNSIGNED NOT NULL DEFAULT 0,
750 PRIMARY KEY (Sede, 'Data'),
    FOREIGN KEY (Sede)
        REFERENCES Sede(Nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
755 ) ENGINE = InnoDB;

-- VIEWS
SELECT "Creazione views." AS "**** FASE 6 ****";
760

CREATE OR REPLACE VIEW IngredientiInScadenza AS
SELECT C.Sede, L.Ingrediente
FROM Lotto L INNER JOIN Confezione C ON L.Codice = C.CodiceLotto
WHERE (C.Stato = 'completa' AND L.Scadenza < CURRENT_DATE + INTERVAL 5 DAY)
765 OR (C.Stato = 'parziale' AND FROM_DAYS(TO_DAYS(L.Scadenza) -
    ROUND(TIMESTAMPDIFF(DAY, C.DataAcquisto, L.Scadenza)*0.2)) <

```

```

                                                                    CURRENT_DATE + INTERVAL 5 DAY)
GROUP BY C.Sede, L.Ingrediente;

770 CREATE OR REPLACE VIEW ConsumiUltimaSettimana AS
SELECT SL.Sede, SL.Ingrediente, COALESCE(SUM(SL.Quantita), 0) as Quantita
FROM Scarichi_Log SL
WHERE SL.'Timestamp' BETWEEN CURRENT_DATE - INTERVAL 1 WEEK AND CURRENT_DATE
GROUP BY SL.Sede, SL.Ingrediente;

775
DELIMITER $$

780 -- STORED ROUTINES
SELECT "Creazione stored routines." AS "***** FASE 7 *****";

CREATE PROCEDURE RegistraClienti(IN inSede VARCHAR(45), IN numero INT)
NOT DETERMINISTIC MODIFIES SQL DATA
785 BEGIN
    INSERT INTO Clienti_Log(Sede, Anno, Mese, SenzaPrenotazione) VALUES
        (inSede, YEAR(CURRENT_DATE), MONTH(CURRENT_DATE), numero)
    ON DUPLICATE KEY
        UPDATE SenzaPrenotazione = SenzaPrenotazione + numero;
790 END;$$

CREATE FUNCTION StatoComanda(idComanda INT)
RETURNS ENUM('nuova', 'in preparazione', 'parziale', 'evasa', 'consegna')
NOT DETERMINISTIC READS SQL DATA
795 BEGIN
    -- bit 1 set: contiene piatti in attesa
    -- bit 2 set: contiene piatti in preparazione
    -- bit 3 set: contiene piatti in servizio
    DECLARE Flags INT;

800
    SET Flags = (SELECT IF(SUM(P.Stato = 'attesa') > 0, 1, 0)
        + IF(SUM(P.Stato = 'in preparazione') > 0, 2, 0)
        + IF(SUM(P.Stato = 'servizio') > 0, 4, 0)
        FROM Piatto P
        WHERE P.Comanda = idComanda);

805
    CASE
        WHEN Flags = 4 THEN -- tutti i piatti in servizio
        BEGIN
810
            DECLARE TakeAway BOOL DEFAULT FALSE;
            SET TakeAway = (SELECT C.Account IS NOT NULL

```

```

                                FROM Comanda C
                                WHERE C.ID = idComanda);
    IF TakeAway THEN
815         RETURN 'consegna';
    ELSE
        RETURN 'evasa';
    END IF;
END;
820 WHEN Flags > 4 THEN RETURN 'parziale'; -- alcuni piatti in servizio
    WHEN Flags > 1 THEN RETURN 'in preparazione'; -- alcuni piatti in prep.
    ELSE RETURN 'nuova'; -- tutti i piatti in attesa (Flags = 1)
END CASE;
END; $$
825 CREATE FUNCTION IngredientiDisponibili(cSede VARCHAR(45), cRicetta VARCHAR(45),
                                           cNovita BOOL, cData DATE)
RETURNS BOOL
NOT DETERMINISTIC READS SQL DATA
830 BEGIN
    DECLARE ClientiPrenotazioni INT;
    DECLARE MediaSenzaPrenotazione INT;
    DECLARE StimaClienti INT;
    DECLARE StimaOrdini INT;
835 DECLARE cIngrediente VARCHAR(45);
    DECLARE cDose INT;
    DECLARE cPrimario BOOL;
    DECLARE qtaDisponibile INT;
    DECLARE Finito BOOL DEFAULT FALSE;
840 DECLARE curIngredienti CURSOR FOR
        SELECT F.Ingrediente, SUM(F.Dose), SUM(F.Primario) > 0
        FROM Fase F
        WHERE F.Ricetta = cRicetta AND F.Ingrediente IS NOT NULL
        GROUP BY F.Ingrediente;
845 DECLARE CONTINUE HANDLER FOR NOT FOUND SET Finito = TRUE;

    IF cData IS NULL THEN
        SET cData = CURRENT_DATE;
    END IF;
850 SET ClientiPrenotazioni = (SELECT COALESCE(CP.Numero, 0)
                                FROM MV_ClientiPrenotazione CP
                                WHERE CP.Sede = cSede
                                AND CP.'Data' = cData);

855 SET MediaSenzaPrenotazione = (SELECT
                                CEIL(COALESCE(AVG(CL.SenzaPrenotazione), 0)/

```

```

                                DAY(LAST_DAY(cData)))
                                AS Media
860      FROM Clienti_Log CL
      WHERE CL.Sede = cSede
            AND CL.Mese = MONTH(cData)
            AND CL.Anno <> YEAR(cData)
    );

865  SET StimaClienti = ClientiPrenotazioni + MediaSenzaPrenotazione;

  IF cNovita THEN
      SET StimaOrdini = (SELECT CEIL(StimaClienti * 0.33));
870  ELSE
      SET StimaOrdini = (SELECT (COALESCE(CEIL(MV.TotOrdini / MV.Comparese), 0)
                                + StimaClienti * 0.1) AS StimaOrdini
                        FROM MV_OrdiniRicetta MV
                        WHERE MV.Sede = cSede AND MV.Ricetta = cRicetta);

875  END IF;

  IF StimaOrdini < 5 THEN
      SET StimaOrdini = 5;
  END IF;

880  OPEN curIngredienti;

  loop_lbl: LOOP
      FETCH curIngredienti INTO cIngrediente, cDose, cPrimario;
885      IF Finito THEN
          LEAVE loop_lbl;
      END IF;

      SET qtaDisponibile = (SELECT SUM(C.Peso)
                            FROM Confezione C INNER JOIN Lotto L
                                ON C.CodiceLotto = L.Codice
                            WHERE C.Sede = cSede
                                AND L.Ingrediente = cIngrediente
                                AND (C.Stato <> 'in ordine'
                                    OR (C.Stato = 'in ordine'
                                        AND C.DataArrivo >=
                                            cData - INTERVAL 3 DAY))
                                AND (C.Aspetto OR (NOT cPrimario)));

890
895
900      IF qtaDisponibile < (cDose * StimaOrdini) THEN
          RETURN FALSE;
      END IF;
  END LOOP loop_lbl;

```

```

905     CLOSE curIngredienti;

    RETURN TRUE;
END; $$

910 CREATE PROCEDURE ConsigliaPiatti(IN nomeSede VARCHAR(45))
    NOT DETERMINISTIC MODIFIES SQL DATA
    BEGIN
        DELETE FROM Report_PiattiDaAggiungere WHERE Sede = nomeSede;

915     INSERT INTO Report_PiattiDaAggiungere(Posizione, Sede, Ricetta)
        SELECT @row_number := @row_number + 1 AS Posizione, nomeSede, D.Ricetta
        FROM (SELECT @row_number := 0) AS N,
            (SELECT R.Nome AS Ricetta, COUNT(*) AS InScadenza,
              (SELECT IF(RPP.NumeroRecensioni = 0, 0,
920                      (RPP.GiudizioTotale/RPP.NumeroRecensioni)/10)
              FROM Report_PiattiPreferiti RPP
              WHERE RPP.Sede = nomeSede
              AND RPP.Ricetta = R.Nome) AS Punteggio
            FROM Fase F INNER JOIN Ricetta R ON F.Ricetta = R.Nome
925     WHERE F.Ingrediente IS NOT NULL
            AND F.Ingrediente IN (SELECT IIS.Ingrediente
                                FROM IngredientiInScadenza IIS
                                WHERE IIS.Sede = nomeSede)

            GROUP BY R.Nome) AS D
930     ORDER BY (D.InScadenza + D.Punteggio) DESC
        LIMIT 5;
    END; $$

    CREATE PROCEDURE AnalizzaRecensioni()
935     NOT DETERMINISTIC MODIFIES SQL DATA
    BEGIN
        TRUNCATE TABLE Report_PiattiPreferiti;

        INSERT INTO Report_PiattiPreferiti(Posizione, Sede, Ricetta, GiudizioTotale,
940                                         NumeroRecensioni)
        SELECT @row_number := @row_number + 1 AS Posizione, D.*
        FROM (SELECT @row_number := 0) AS N,
            (
945         SELECT R.Sede, R.Ricetta,
            SUM(R.Giudizio)*IF(SUM(R.NumeroValutazioni) = 0, 6, ROUND(
            AVG((R.VeridicitaTotale + R.AccuratezzaTotale)/R.NumeroValutazioni))
            ) AS GiudizioTotale, COUNT(*) AS NumeroRecensioni
            FROM Recensione R

```



```

950         GROUP BY R.Sede, R.Ricetta
        ) AS D
        ORDER BY D.GiudizioTotale/D.NumeroRecensioni DESC;
END;$$

CREATE PROCEDURE AnalizzaVendite(IN inizio TIMESTAMP, IN fine TIMESTAMP)
955 NOT DETERMINISTIC MODIFIES SQL DATA
BEGIN
    IF inizio IS NULL THEN
        SET inizio = '1970-01-01 00:00:01';
    END IF;
960    IF fine IS NULL THEN
        SET fine = CURRENT_TIMESTAMP;
    END IF;

    TRUNCATE TABLE Report_VenditePiatti;

965    INSERT INTO Report_VenditePiatti(Posizione, Sede, Ricetta, Vendite)
    SELECT @row_number := @row_number + 1 AS Posizione, D.*
    FROM (SELECT @row_number := 0) AS N,
        (
970            SELECT C.Sede, P.Ricetta, COUNT(*) AS Vendite
            FROM Piatto P INNER JOIN Comanda C ON P.Comanda = C.ID
            WHERE C.'Timestamp' BETWEEN inizio AND fine
            GROUP BY C.Sede, P.Ricetta
        ) AS D
    ORDER BY D.Vendite DESC;
975 END;$$

CREATE PROCEDURE AnalizzaSuggerimenti()
NOT DETERMINISTIC MODIFIES SQL DATA
980 BEGIN
    TRUNCATE TABLE Report_SuggerimentiMigliori;

    INSERT INTO Report_SuggerimentiMigliori(Posizione, Suggerimento,
985             GradimentoTotale, NumeroGradimenti)
    SELECT @row_number := @row_number + 1 AS Posizione, D.*
    FROM (SELECT @row_number := 0) AS N,
        (
990            SELECT G.Suggerimento, SUM(G.Punteggio) AS GradimentoTotale,
            COUNT(*) AS NumeroGradimenti
            FROM Gradimento G
            WHERE G.Suggerimento IS NOT NULL
            GROUP BY G.Suggerimento
        ) AS D

```

```

ORDER BY D.GradimentoTotale/D.NumeroGradimenti DESC;
995 END;$$

CREATE PROCEDURE AnalizzaProposte()
NOT DETERMINISTIC MODIFIES SQL DATA
BEGIN
1000 TRUNCATE TABLE Report_ProposteMigliori;

INSERT INTO Report_ProposteMigliori(Posizione, Proposta, GradimentoTotale,
                                     NumeroGradimenti)

1005 SELECT @row_number := @row_number + 1 AS Posizione, D.*
FROM (SELECT @row_number := 0) AS N,
(
    SELECT G.Proposta, SUM(G.Punteggio) AS GradimentoTotale,
                                     COUNT(*) AS NumeroGradimenti
    FROM Gradimento G
1010 WHERE G.Proposta IS NOT NULL
    GROUP BY G.Proposta
) AS D
ORDER BY D.GradimentoTotale/D.NumeroGradimenti DESC;
END;$$

1015 CREATE PROCEDURE AnalizzaSprechi(IN inizio TIMESTAMP, IN fine TIMESTAMP)
NOT DETERMINISTIC MODIFIES SQL DATA
BEGIN
    IF inizio IS NULL THEN
1020 SET inizio = '1970-01-01 00:00:01';
    END IF;
    IF fine IS NULL THEN
        SET fine = CURRENT_TIMESTAMP;
    END IF;

1025 TRUNCATE TABLE Report_Sprechi;

BEGIN
    DECLARE NomeSede VARCHAR(45);
1030 DECLARE NomeIngrediente VARCHAR(45);
    DECLARE Scaricata INT;
    DECLARE Quantita INT;
    DECLARE Finito BOOL DEFAULT FALSE;
    DECLARE curScarichi CURSOR FOR
1035 SELECT SL.Sede, SL.Ingrediente, COALESCE(SUM(SL.Quantita), 0) AS Qta
    FROM Scarichi_Log SL
    WHERE SL.'Timestamp' BETWEEN inizio AND fine
    GROUP BY SL.Sede, SL.Ingrediente;

```

```

1040 DECLARE CONTINUE HANDLER FOR NOT FOUND SET Finito = TRUE;

OPEN curScarichi;

loop_lbl: LOOP
1045   FETCH curScarichi INTO NomeSede, NomeIngrediente, Scaricata;
   IF Finito THEN
       LEAVE loop_lbl;
   END IF;

   SET Quantita = (
1050   SELECT COALESCE(SUM(F.Dose), 0) AS Quantita
   FROM Fase F
       INNER JOIN Ricetta R ON F.Ricetta = R.Nome
       INNER JOIN Piatto P ON R.Nome = P.Ricetta
       INNER JOIN Comanda C ON P.Comanda = C.ID
1055   WHERE C.Sede = NomeSede
       AND F.Ingrediente = NomeIngrediente
       AND C.'Timestamp' BETWEEN inizio AND fine
       AND F.ID NOT IN (SELECT MF.FaseVecchia
                        FROM ModificaFase MF
                        INNER JOIN Variazione V
                        ON MF.Variazione = V.ID
                        INNER JOIN
1060   (SELECT M.Variazione
      FROM Modifica M
      WHERE M.Piatto = P.ID) AS D
                        ON V.ID = D.Variazione)
       AND F.ID NOT IN (SELECT MFN.FaseNuova
                        FROM ModificaFase MFN
                        INNER JOIN Variazione VA
                        ON MFN.Variazione = VA.ID
1065   WHERE VA.ID NOT IN (SELECT MO.Variazione
                        FROM Modifica MO
                        WHERE MO.Piatto = P.ID))

   );

1075   INSERT INTO Report_Sprechi(Sede, Ingrediente, Spreco)
   VALUES (NomeSede, NomeIngrediente, Scaricata - Quantita);

END LOOP loop_lbl;

1080 CLOSE curScarichi;

END;
END;$$

```

```

1085  -- TRIGGERS
      SELECT "Creazione triggers." AS "***** FASE 8 *****";

      CREATE TRIGGER nuova_sede
1090  AFTER INSERT
      ON Sede
      FOR EACH ROW
      BEGIN
          INSERT INTO Clienti_Log(Sede, Anno, Mese) VALUES
1095  (NEW.Nome, YEAR(CURRENT_DATE), MONTH(CURRENT_DATE));
      END; $$

      CREATE TRIGGER nuovo_magazzino
      BEFORE INSERT
1100  ON Magazzino
      FOR EACH ROW
      BEGIN
          -- Simula AUTO_INCREMENT
          IF NEW.ID IS NULL THEN
1105  SET NEW.ID = (SELECT IFNULL(MAX(ID), 0) + 1
                  FROM Magazzino
                  WHERE Sede = NEW.Sede);
          END IF;
      END; $$

1110  CREATE TRIGGER nuova_confezione
      BEFORE INSERT
      ON Confezione
      FOR EACH ROW
1115  BEGIN
          -- Simula AUTO_INCREMENT
          IF NEW.Numero IS NULL THEN
              SET NEW.Numero = (SELECT IFNULL(MAX(Numero), 0) + 1
                                FROM Confezione
                                WHERE CodiceLotto = NEW.CodiceLotto);
1120  END IF;

          IF NEW.DataCarico IS NOT NULL THEN
              IF NEW.DataCarico < NEW.DataAcquisto THEN
1125  SIGNAL SQLSTATE '45000'
              SET MESSAGE_TEXT = 'DataCarico precedente a DataAcquisto.';
          END IF;
          IF NEW.Collocazione IS NULL THEN
              SIGNAL SQLSTATE '45000'

```

```

1130         SET MESSAGE_TEXT = 'L\'attributo Collocazione non può essere NULL.';
        END IF;
        IF NEW.Aspetto IS NULL THEN
            SET NEW.Aspetto = TRUE; -- Default: nessun danno
        END IF;
1135        IF NEW.Stato IS NULL THEN
            SET NEW.Stato = 'completa'; -- Default
        END IF;
    ELSE
        IF (NEW.Stato <> 'in ordine') THEN
1140            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'L\'attributo DataCarico non può essere NULL.';
        ELSEIF (NEW.Collocazione IS NOT NULL
            OR NEW.Aspetto IS NOT NULL) THEN
            SIGNAL SQLSTATE '45000'
1145            SET MESSAGE_TEXT = 'DataCarico, Collocazione e Aspetto devono '
                                'essere tutti NULL o tutti non-NULL.';
        END IF;
    END IF;
END;$$
1150
CREATE TRIGGER aggiorna_confezione
BEFORE UPDATE
ON Confezione
FOR EACH ROW
1155 BEGIN
    IF NEW.DataCarico IS NOT NULL THEN
        IF NEW.DataCarico < NEW.DataAcquisto THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'DataCarico precedente a DataAcquisto.';
1160        END IF;
        IF NEW.Collocazione IS NULL THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'L\'attributo Collocazione non può essere NULL.';
        END IF;
1165        IF NEW.Aspetto IS NULL THEN
            SET NEW.Aspetto = TRUE; -- Default: nessun danno
        END IF;
    ELSE
        IF (NEW.Stato <> 'in ordine') THEN
1170            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'L\'attributo DataCarico non può essere NULL.';
        ELSEIF (NEW.Collocazione IS NOT NULL
            OR NEW.Aspetto IS NOT NULL) THEN
            SIGNAL SQLSTATE '45000'
1175            SET MESSAGE_TEXT = 'DataCarico, Collocazione e Aspetto devono '

```

```

                                'essere tutti NULL o tutti non-NULL.';
        END IF;
    END IF;
END;$$

1180 CREATE TRIGGER elimina_lotto_confezione
    AFTER DELETE
    ON Confezione
    FOR EACH ROW
1185 BEGIN
    DECLARE LottoPresente BOOL;
    DECLARE IngScaricato VARCHAR(45);

    SET LottoPresente = (SELECT COUNT(*) > 0
1190                        FROM Confezione C
                        WHERE C.CodiceLotto = OLD.CodiceLotto);

    IF NOT LottoPresente THEN
        DELETE FROM Lotto WHERE Codice = OLD.CodiceLotto;
1195    END IF;

    IF OLD.Stato = 'in uso' THEN
        SET IngScaricato = (SELECT L.Ingrediente
                            FROM Lotto L
1200                            WHERE L.Codice = OLD.CodiceLotto);

        INSERT INTO Scarichi_Log(Sede, Magazzino, Ingrediente, Quantita)
        VALUES (OLD.Sede, OLD.Magazzino, IngScaricato, OLD.Peso)
        ON DUPLICATE KEY
1205            UPDATE Quantita = Quantita + OLD.Peso;
    END IF;
END;$$

1210 CREATE TRIGGER aggiorna_Scarichi_Log_update
    AFTER UPDATE
    ON Confezione
    FOR EACH ROW
    BEGIN
        DECLARE IngScaricato VARCHAR(45);

1215        IF OLD.Stato = 'in uso' AND NEW.Stato = 'parziale'
            AND OLD.Peso > NEW.Peso THEN
            SET IngScaricato = (SELECT L.Ingrediente
                                FROM Lotto L
1220                                WHERE L.Codice = NEW.CodiceLotto);

```

```

INSERT INTO Scarichi_Log(Sede, Magazzino, Ingrediente, Quantita)
VALUES (NEW.Sede, NEW.Magazzino, IngScaricato, OLD.Peso - NEW.Peso)
ON DUPLICATE KEY
1225     UPDATE Quantita = Quantita + (OLD.Peso - NEW.Peso);
END IF;
END;$$

CREATE TRIGGER nuovo_menu
1230 BEFORE INSERT
ON Menu
FOR EACH ROW
BEGIN
1235     DECLARE MenuAttiviPeriodo BOOL;

    IF NEW.DataFine <= NEW.DataInizio THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'DataFine precedente a DataInizio.';
    END IF;

1240     SET MenuAttiviPeriodo = (SELECT COUNT(*) > 0
                                FROM Menu M
                                WHERE M.Sede = NEW.Sede
                                AND M.DataFine >= NEW.DataInizio
                                AND M.DataInizio <= NEW.DataFine);

1245     IF MenuAttiviPeriodo THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Un menu è già attivo in questo periodo.';
    END IF;
1250 END IF;
END;$$

CREATE TRIGGER nuovo_elenco
1255 BEFORE INSERT
ON Elenco
FOR EACH ROW
BEGIN
    SET NEW.Novita = (SELECT (COUNT(*) = 0) AS PrimaVolta
                      FROM MV_OrdiniRicetta MVOR
1260                      WHERE MVOR.Ricetta = NEW.Ricetta
                      AND MVOR.Sede = (SELECT M.Sede
                                       FROM Menu M
                                       WHERE M.ID = NEW.Menu));
END;$$
1265

```

```

CREATE TRIGGER controllo_ingredienti
AFTER INSERT
ON Elenco
FOR EACH ROW
1270 BEGIN
    DECLARE cSede VARCHAR(45);
    DECLARE cData DATE;

    SELECT M.Sede, M.DataInizio INTO cSede, cData
1275 FROM Menu M
    WHERE M.ID = NEW.Menu;

    IF NOT IngredientiDisponibili(cSede, NEW.Ricetta, NEW.Novita, cData) THEN
        SIGNAL SQLSTATE '01000' -- Warning
1280 SET MESSAGE_TEXT = 'La ricetta potrebbe non comparire nel menu in '
                                'quanto potrebbe non esserci una quantità '
                                'sufficiente di ingredienti.';

    END IF;
END; $$
1285

CREATE TRIGGER nuova_fase
BEFORE INSERT
ON Fase
FOR EACH ROW
1290 BEGIN
    IF NEW.Ingrediente IS NOT NULL THEN
        SET NEW.Durata = NULL;
        SET NEW.Testo = NULL;
        IF NEW.Strumento IS NOT NULL THEN
1295 SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Una fase può impiegare o uno strumento o un '
                                'ingrediente. Non entrambi.';

        END IF;
        IF NEW.Dose IS NULL THEN
1300 SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'L\'attributo Dose deve essere specificato.';
        END IF;
        IF NEW.Primario IS NULL THEN
            SET NEW.Primario = FALSE; -- Default
1305 END IF;
        ELSE
            SET NEW.Dose = NULL;
            SET NEW.Primario = NULL;

1310 IF NEW.Durata IS NULL THEN
            SET NEW.Durata = '00:00:00'; -- Default

```



```

        END IF;
        IF NEW.Testo IS NULL THEN
            SIGNAL SQLSTATE '45000'
1315         SET MESSAGE_TEXT = 'L\'attributo Testo deve essere specificato.';
        END IF;
    END IF;
END;$$

1320 CREATE TRIGGER aggiorna_fase
BEFORE UPDATE
ON Fase
FOR EACH ROW
BEGIN
1325     IF NEW.Ingrediente IS NOT NULL THEN
        SET NEW.Durata = NULL;
        SET NEW.Testo = NULL;
        IF NEW.Strumento IS NOT NULL THEN
            SIGNAL SQLSTATE '45000'
1330         SET MESSAGE_TEXT = 'Una fase può impiegare o uno strumento o un '
                                'ingrediente. Non entrambi.';
        END IF;
        IF NEW.Dose IS NULL THEN
            SIGNAL SQLSTATE '45000'
1335         SET MESSAGE_TEXT = 'L\'attributo Dose deve essere specificato.';
        END IF;
        IF NEW.Primario IS NULL THEN
            SET NEW.Primario = FALSE; -- Default
        END IF;
1340     ELSE
        SET NEW.Dose = NULL;
        SET NEW.Primario = NULL;

        IF NEW.Durata = NULL THEN
1345         SET NEW.Durata = '00:00:00'; -- Default
        END IF;
        IF NEW.Testo = NULL THEN
            SIGNAL SQLSTATE '45000'
1350         SET MESSAGE_TEXT = 'L\'attributo Testo deve essere '
                                'specificato.';
        END IF;
    END IF;
END;$$

1355 CREATE TRIGGER nuova_sequenza_fasi
BEFORE INSERT

```

```

ON SequenzaFasi
FOR EACH ROW
BEGIN
1360   DECLARE StessaRicetta BOOL;
      SET StessaRicetta = (SELECT COUNT(*) > 0
                           FROM (SELECT F1.ID, F1.Ricetta
                                FROM Fase F1
                                WHERE F1.ID = NEW.Fase) AS Fase1
                           INNER JOIN
                           (SELECT F2.ID, F2.Ricetta
                                FROM Fase F2
                                WHERE F2.ID = NEW.FasePrecedente) AS Fase2
                           ON Fase1.Ricetta = Fase2.Ricetta);
1365
1370   IF NOT StessaRicetta THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Le due fasi non appartengono alla stessa ricetta.';
      END IF;
1375 END; $$

CREATE TRIGGER aggiorna_sequenza_fasi
BEFORE UPDATE
ON SequenzaFasi
1380 FOR EACH ROW
BEGIN
      DECLARE StessaRicetta BOOL;
      SET StessaRicetta = (SELECT COUNT(*) > 0
                           FROM (SELECT F1.ID, F1.Ricetta
                                FROM Fase F1
                                WHERE F1.ID = NEW.Fase) AS Fase1
                           INNER JOIN
                           (SELECT F2.ID, F2.Ricetta
                                FROM Fase F2
                                WHERE F2.ID = NEW.FasePrecedente) AS Fase2
                           ON Fase1.Ricetta = Fase2.Ricetta);
1385
1390   IF NOT StessaRicetta THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Le due fasi non appartengono alla stessa ricetta.';
1395   END IF;
      END; $$

CREATE TRIGGER nuova_sala
1400 BEFORE INSERT
ON Sala

```

```

FOR EACH ROW
BEGIN
  -- Simula AUTO_INCREMENT
  IF NEW.Numero IS NULL THEN
    SET NEW.Numero = (SELECT IFNULL(MAX(Numero), 0) + 1
                      FROM Sala
                      WHERE Sede = NEW.Sede);
  END IF;
END;$$

CREATE TRIGGER nuovo_tavolo
BEFORE INSERT
ON Tavolo
FOR EACH ROW
BEGIN
  -- Simula AUTO_INCREMENT
  IF NEW.Numero IS NULL THEN
    SET NEW.Numero = (SELECT IFNULL(MAX(Numero), 0) + 1
                      FROM Tavolo
                      WHERE Sede = NEW.Sede
                      AND Sala = NEW.Sala);
  END IF;
END;$$

CREATE TRIGGER nuova_comanda
BEFORE INSERT
ON Comanda
FOR EACH ROW
BEGIN
  IF (NEW.Account IS NOT NULL
      AND (NEW.Tavolo IS NOT NULL OR NEW.Sala IS NOT NULL))
  OR (NEW.Account IS NULL AND NEW.Tavolo IS NULL
      AND NEW.Sala IS NULL) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Una comanda deve essere o da tavolo o take-away. '
                      'Non entrambe.';
  END IF;
END;$$

CREATE TRIGGER aggiorna_comanda
BEFORE UPDATE
ON Comanda
FOR EACH ROW
BEGIN
  IF (NEW.Account IS NOT NULL

```

```

        AND (NEW.Tavolo IS NOT NULL OR NEW.Sala IS NOT NULL))
    OR (NEW.Account IS NULL AND NEW.Tavolo IS NULL
        AND NEW.Sala IS NULL) THEN
1450  SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Una comanda deve essere o da tavolo o take-away. '
        'Non entrambe.';
END IF;
END; $$
1455
CREATE TRIGGER assegna_pony
AFTER UPDATE
ON Piatto
FOR EACH ROW
1460 BEGIN
    DECLARE NumeroPiatti INT;
    DECLARE SedeComanda VARCHAR(45);
    DECLARE PonyScelto INT;

1465  IF StatoComanda(NEW.Comanda) = 'consegna' THEN
    SET SedeComanda = (SELECT C.Account <> NULL, C.Sede
        FROM Comanda C
        WHERE C.ID = NEW.Comanda);

1470  SET NumeroPiatti = (SELECT COUNT(*)
        FROM Piatto P
        WHERE P.Comanda = NEW.Comanda);

    SET PonyScelto = (SELECT P.ID
1475        FROM Pony P
        WHERE P.Sede = SedeComanda
            AND P.Stato = 'libero'
            AND Ruote = (NumeroPiatti > 5)
            LIMIT 1);

1480  IF PonyScelto IS NULL THEN
    SET PonyScelto = (SELECT P.ID
        FROM Pony P
        WHERE P.Sede = SedeComanda
1485        AND P.Stato = 'libero'
            LIMIT 1);

    END IF;

    IF PonyScelto IS NULL THEN
1490  SIGNAL SQLSTATE '01000' -- Warning
    SET MESSAGE_TEXT = 'Nessun Pony è stato assegnato in '
        'quanto sono tutti occupati.';

```

```

ELSE
    INSERT INTO Consegna(Comanda, Sede, Pony, Arrivo, Ritorno)
1495     VALUES (NEW.Comanda, SedeComanda, PonyScelto, NULL, NULL);
END IF;
END IF;
END;$$

1500 CREATE TRIGGER nuova_variazione
BEFORE INSERT
ON Variazione
FOR EACH ROW
BEGIN
1505     IF (NEW.Nome IS NOT NULL AND NEW.Account IS NOT NULL)
        OR (NEW.Nome IS NULL AND NEW.Account IS NULL) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Una variazione deve essere una variazione '
                                'dagli chef (con nome) o un suggerimento inviato '
                                'da un account utente (senza nome).';
1510     END IF;
END;$$

CREATE TRIGGER aggiorna_variazione
1515 BEFORE UPDATE
ON Variazione
FOR EACH ROW
BEGIN
    IF (NEW.Nome IS NOT NULL AND NEW.Account IS NOT NULL)
1520     OR (NEW.Nome IS NULL AND NEW.Account IS NULL) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Una variazione deve essere una variazione '
                                'dagli chef (con nome) o un suggerimento inviato '
                                'da un account utente (senza nome).';
1525     END IF;
END;$$

CREATE TRIGGER nuova_modificafase
BEFORE INSERT
1530 ON ModificaFase
FOR EACH ROW
BEGIN
    DECLARE RicettaFaseNuova VARCHAR(45);
    DECLARE RicettaFaseVecchia VARCHAR(45);
1535     DECLARE RicettaVariazione VARCHAR(45);
    DECLARE FaseInAggiunta BOOL;
    DECLARE FaseInEliminazione BOOL;

```

```

1540 IF NEW.FaseNuova IS NULL AND NEW.FaseVecchia IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'FaseNuova o FaseVecchia devono essere specificati.';
END IF;

1545 SET RicettaVariazione = (SELECT V.Ricetta FROM Variazione V
    WHERE V.ID = NEW.Variazione);

IF NEW.FaseNuova IS NOT NULL THEN
    SET RicettaFaseNuova = (SELECT F.Ricetta FROM Fase F
    WHERE F.ID = NEW.FaseNuova);

1550 IF RicettaFaseNuova <> RicettaVariazione THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'La ricetta di FaseNuova deve corrispondere '
    'alla ricetta della variazione.';

1555 END IF;

SET FaseInEliminazione = (SELECT COUNT(*) > 0
    FROM ModificaFase MF
    WHERE MF.FaseVecchia = NEW.FaseNuova);

1560 IF FaseInEliminazione THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'La fase inserita come FaseNuova viene '
    'già eliminata da un\'altra ModificaFase. Una '
    'fase può essere solo aggiunta o rimossa dalle '
    'ModificaFase.';

    END IF;
END IF;

1570 IF NEW.FaseVecchia IS NOT NULL THEN
    SET RicettaFaseVecchia = (SELECT F.Ricetta FROM Fase F
    WHERE F.ID = NEW.FaseVecchia);

    IF RicettaFaseVecchia <> RicettaVariazione THEN
1575 SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La ricetta di FaseVecchia deve corrispondere '
        'alla ricetta della variazione.';

    ELSEIF RicettaFaseNuova IS NOT NULL
        AND RicettaFaseNuova <> RicettaFaseVecchia THEN
1580 SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La ricetta di FaseVecchia e quella di '
        'FaseNuova devono corrispondere.';

    END IF;

```

```

1585     SET FaseInAggiunta = (SELECT COUNT(*) > 0
                                FROM ModificaFase MF
                                WHERE MF.FaseNuova = NEW.FaseVecchia);

    IF FaseInAggiunta THEN
1590         SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT = 'La fase inserita come FaseVecchia viene '
                                'già aggiunta da un\'altra ModificaFase. Una '
                                'fase può essere solo aggiunta o rimossa dalle '
                                'ModificaFase.';
1595     END IF;
    END IF;
END;$$

CREATE TRIGGER aggiorna_modificafase
1600 BEFORE UPDATE
    ON ModificaFase
    FOR EACH ROW
    BEGIN
1605     DECLARE RicettaFaseNuova VARCHAR(45);
     DECLARE RicettaFaseVecchia VARCHAR(45);
     DECLARE RicettaVariazione VARCHAR(45);
     DECLARE FaseInAggiunta BOOL;
     DECLARE FaseInEliminazione BOOL;

1610     IF NEW.FaseNuova IS NULL AND NEW.FaseVecchia IS NULL THEN
         SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT = 'FaseNuova o FaseVecchia devono essere specificati.';
     END IF;

1615     SET RicettaVariazione = (SELECT V.Ricetta FROM Variazione V
                                WHERE V.ID = NEW.Variazione);

     IF NEW.FaseNuova IS NOT NULL THEN
1620         SET RicettaFaseNuova = (SELECT F.Ricetta FROM Fase F
                                    WHERE F.ID = NEW.FaseNuova);

         IF RicettaFaseNuova <> RicettaVariazione THEN
1625             SIGNAL SQLSTATE '45000'
             SET MESSAGE_TEXT = 'La ricetta di FaseNuova deve corrispondere '
                                    'alla ricetta della variazione.';
         END IF;

         SET FaseInEliminazione = (SELECT COUNT(*) > 0
                                    FROM ModificaFase MF

```

```

1630                                     WHERE MF.FaseVecchia = NEW.FaseNuova);

    IF FaseInEliminazione THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La fase inserita come FaseNuova viene '
1635                                'già eliminata da un\'altra ModificaFase. Una '
                                'fase può essere solo aggiunta o rimossa dalle '
                                'ModificaFase.';

    END IF;
END IF;

1640 IF NEW.FaseVecchia IS NOT NULL THEN
    SET RicettaFaseVecchia = (SELECT F.Ricetta FROM Fase F
                                WHERE F.ID = NEW.FaseVecchia);

    IF RicettaFaseVecchia <> RicettaVariazione THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La ricetta di FaseVecchia deve corrispondere '
1645                                'alla ricetta della variazione.';

    ELSEIF RicettaFaseNuova IS NOT NULL
        AND RicettaFaseNuova <> RicettaFaseVecchia THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La ricetta di FaseVecchia e quella di '
1650                                'FaseNuova devono corrispondere.';

    END IF;

1655 SET FaseInAggiunta = (SELECT COUNT(*) > 0
                            FROM ModificaFase MF
                            WHERE MF.FaseNuova = NEW.FaseVecchia);

    IF FaseInAggiunta THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La fase inserita come FaseVecchia viene '
1660                                'già aggiunta da un\'altra ModificaFase. Una '
                                'fase può essere solo aggiunta o rimossa dalle '
                                'ModificaFase.';

1665    END IF;
END IF;
END; $$

1670 CREATE TRIGGER nuova_modifica
    BEFORE INSERT
    ON Modifica
    FOR EACH ROW
    BEGIN
1675     DECLARE Suggerimento BOOL;

```



```

DECLARE NumVariazioni INT;
DECLARE StessaRicetta BOOL;

SET Suggerimento = (SELECT V.Account IS NOT NULL
1680                      FROM Variazione V
                      WHERE V.ID = NEW.Variazione);

IF Suggerimento THEN
    SIGNAL SQLSTATE '45000'
1685    SET MESSAGE_TEXT = 'Solo le variazioni selezionate dagli chef possono '
                        'essere selezionate, non i suggerimenti.';
END IF;

SET NumVariazioni = (SELECT COUNT(*)
1690                      FROM Modifica M
                      WHERE M.Piatto = NEW.Piatto);

IF NumVariazioni >= 3 THEN
    SIGNAL SQLSTATE '45000'
1695    SET MESSAGE_TEXT = 'Ci sono già 3 variazioni su questo piatto.';
END IF;

SET StessaRicetta = (SELECT COUNT(*) > 0
1700                      FROM (SELECT P.ID, P.Ricetta
                              FROM Piatto P
                              WHERE P.ID = NEW.Piatto) AS Pi
                      INNER JOIN
                      (SELECT V.ID, V.Ricetta
1705                      FROM Variazione V
                      WHERE V.ID = NEW.Variazione) AS Va
                      ON Pi.Ricetta = Va.Ricetta);

IF NOT StessaRicetta THEN
    SIGNAL SQLSTATE '45000'
1710    SET MESSAGE_TEXT = 'La variazione selezionata non è applicabile al '
                        'piatto scelto.';
END IF;
END;$$

1715 CREATE TRIGGER nuovo_pony
      BEFORE INSERT
      ON Pony
      FOR EACH ROW
      BEGIN
1720          -- Simula AUTO_INCREMENT
          IF NEW.ID IS NULL THEN

```

```

        SET NEW.ID = (SELECT IFNULL(MAX(ID), 0) + 1
                        FROM Pony
                        WHERE Sede = NEW.Sede);
1725     END IF;
END; $$

CREATE TRIGGER nuova_consegna
1730 BEFORE INSERT
ON Consegna
FOR EACH ROW
BEGIN
    UPDATE Pony SET Stato = 'occupato'
1735     WHERE Sede = NEW.Sede
        AND ID = NEW.Pony;

    -- Le nuove consegne non devono avere mai Arrivo e Ritorno.
    SET NEW.Arrivo = NULL;
    SET NEW.Ritorno = NULL;
1740 END; $$

CREATE TRIGGER aggiorna_consegna
BEFORE UPDATE
ON Consegna
1745 FOR EACH ROW
BEGIN
    IF NEW.Arrivo IS NOT NULL AND NEW.Arrivo < NEW.Partenza THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Arrivo precedente a partenza.';
1750 ELSEIF NEW.Arrivo IS NULL AND NEW.Ritorno IS NOT NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Se Ritorno viene specificato anche Arrivo deve '
            'essere specificato.';

    END IF;
1755

    IF NEW.Ritorno IS NOT NULL THEN
        IF OLD.Arrivo IS NOT NULL AND OLD.Arrivo = NEW.Arrivo THEN
            -- Evita ON UPDATE CURRENT_TIMESTAMP
1760             SET NEW.Arrivo = OLD.Arrivo;
        END IF;
        IF NEW.Ritorno < NEW.Arrivo THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Ritorno precedente a arrivo.';
1765        END IF;
    END IF;
END IF;

```

```

END;$$

CREATE TRIGGER libera_pony
1770 AFTER UPDATE
ON Consegna
FOR EACH ROW
BEGIN
    IF NEW.Ritorno IS NOT NULL AND OLD.Ritorno IS NULL THEN
1775 UPDATE Pony SET Stato = 'libero' WHERE Sede = NEW.Sede
AND ID = NEW.Pony;
    END IF;
END;$$

1780 CREATE TRIGGER nuova_prenotazione
BEFORE INSERT
ON Prenotazione
FOR EACH ROW
BEGIN
1785 DECLARE PrenotazioniAbilitate BOOL;
DECLARE PostiTavolo INT;
DECLARE TavoloLibero BOOL;
DECLARE SalaLibera BOOL;
DECLARE TavoloAssegnato INT;
1790 DECLARE SalaAssegnata INT;

IF CURRENT_DATETIME > (NEW.'Data' - INTERVAL 1 DAY) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Una prenotazione deve essere effettuata almeno un '
1795 'giorno prima della data scelta.';
END IF;

IF NEW.Tavolo IS NOT NULL THEN
    SET PostiTavolo = (SELECT T.Posti
1800 FROM Tavolo T
WHERE T.ID = NEW.Tavolo
AND T.Sala = NEW.Sala
AND T.Sede = NEW.Sede);
END IF;

1805 IF NEW.Account IS NOT NULL THEN
    SET PrenotazioniAbilitate = (SELECT A.PuoPrenotare
FROM Account A
WHERE A.Username = NEW.Account);

1810 IF NOT PrenotazioniAbilitate THEN
    SIGNAL SQLSTATE '45000'

```

```

SET MESSAGE_TEXT = 'Prenotazioni disabilitate per l\'account.';
END IF;

1815 SET NEW.Telefono = NULL;
IF NEW.Tavolo IS NOT NULL THEN
    SET NEW.Nome = NULL;
    SET NEW.Descrizione = NULL;
1820 SET NEW.Approvato = NULL;

    IF PostiTavolo > NEW.Numero + 3 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Il tavolo scelto non ha un numero adeguato '
1825                          'di posti.';
    END IF;
ELSEIF NEW.Nome IS NULL THEN
    SET NEW.Descrizione = NULL;
    SET NEW.Approvato = NULL;
1830
    -- Assegna Tavolo
    SELECT T.Numero, T.Sala INTO TavoloAssegnato, SalaAssegnata
    FROM Tavolo T
    WHERE T.Numero NOT IN (
1835        SELECT P1.Tavolo
        FROM Prenotazione P1
        WHERE P1.'Data' >
            (NEW.'Data' - INTERVAL 2 HOUR)
            AND P1.'Data' <
1840            (NEW.'Data' + INTERVAL 2 HOUR))
        AND T.Sala NOT IN (
            SELECT P2.Sala
            FROM Prenotazione P2
            WHERE P2.'Data' <
1845            (NEW.'Data' - INTERVAL 2 HOUR)
            AND P2.'Data' >
                (NEW.'Data' + INTERVAL 2 HOUR)
            AND P2.Tavolo = NULL)
        AND T.Posti BETWEEN NEW.Numero
1850                          AND (NEW.Numero + 3)
    ORDER BY T.Posti ASC
    LIMIT 1;

    IF TavoloAssegnato IS NULL THEN
1855        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Non ci sono tavoli liberi per questo '
                          'numero di persone.';
    END IF;

```

```

1860         SET NEW.Tavolo = TavoloAssegnato;
        SET NEW.Sala = SalaAssegnata;

        ELSEIF NEW.Descrizione IS NULL THEN
            SIGNAL SQLSTATE '45000'
1865         SET MESSAGE_TEXT = 'Descrizione deve essere specificato per gli '
                                'allestimenti.';
        ELSEIF NEW.Approvato IS NULL THEN
            SET NEW.Approvato = FALSE; -- Default
        END IF;
1870 ELSE
        SET NEW.Descrizione = NULL;
        SET NEW.Approvato = NULL;
        IF NEW.Nome IS NULL OR NEW.Telefono IS NULL OR NEW.Tavolo IS NULL THEN
            SIGNAL SQLSTATE '45000'
1875         SET MESSAGE_TEXT = 'Nome, Telefono e Tavolo devono essere '
                                'specificati per le prenotazioni telefoniche.';
        END IF;
    END IF;

1880 SET SalaLibera = (SELECT SUM(DATE(P.'Data') = DATE(NEW.'Data')) > 0
                        FROM Prenotazione P
                        WHERE P.Sala = NEW.Sala
                        AND P.Sede = NEW.Sede
                        AND P.Tavolo = NULL);

1885 IF SalaLibera THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La sala scelta è già prenotata per un '
                                'allestimento.';
1890 END IF;

    IF NEW.Tavolo IS NOT NULL THEN

        IF PostiTavolo < NEW.Numero THEN
1895         SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il tavolo scelto non ha un numero adeguato '
                                'di posti.';
        END IF;

1900 SET TavoloLibero = (SELECT SUM(P.'Data' >
                                (NEW.'Data' - INTERVAL 2 HOUR)
                                AND P.'Data' <
                                (NEW.'Data' + INTERVAL 2 HOUR)) = 0
                        FROM Prenotazione P

```

```

1905         WHERE P.Tavolo = NEW.Tavolo
              AND P.Sala = NEW.Sala
              AND P.Sede = NEW.Sede);

1910     IF NOT TavoloLibero THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Il tavolo scelto è già prenotato.';
    END IF;
ELSE
    SET SalaLibera = (SELECT SUM(DATE(P.'Data') = DATE(NEW.'Data')) = 0
1915        FROM Prenotazione P
        WHERE P.Sala = NEW.Sala
              AND P.Sede = NEW.Sede);

    IF NOT SalaLibera THEN
1920        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La sala contiene tavoli già prenotati.';
    END IF;
END IF;
END; $$

1925 CREATE TRIGGER aggiorna_prenotazione
    BEFORE UPDATE
    ON Prenotazione
    FOR EACH ROW
1930 BEGIN
    DECLARE PostiTavolo INT;
    DECLARE TavoloLibero BOOL;
    DECLARE SalaLibera BOOL;

1935    IF CURRENT_DATETIME > (OLD.'Data' - INTERVAL 2 DAY) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Non è possibile modificare la prenotazione.';
    END IF;

1940    IF NEW.Tavolo IS NOT NULL THEN
        SET PostiTavolo = (SELECT T.Posti
                          FROM Tavolo T
                          WHERE T.ID = NEW.Tavolo
                                AND T.Sala = NEW.Sala
                                AND T.Sede = NEW.Sede);
1945    END IF;

    IF NEW.Account IS NOT NULL THEN
        SET NEW.Telefono = NULL;
1950    IF NEW.Tavolo IS NOT NULL THEN

```

```

SET NEW.Nome = NULL;
SET NEW.Descrizione = NULL;
SET NEW.Approvato = NULL;

1955 IF PostiTavolo > NEW.Numero + 3 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Il tavolo scelto non ha un numero adeguato '
                        'di posti.';

    END IF;
1960 ELSEIF NEW.Nome IS NULL OR NEW.Descrizione IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Nome e Descrizione devono essere specificati '
                        'per gli allestimenti.';

    ELSEIF NEW.Approvato IS NULL THEN
1965     SET NEW.Approvato = FALSE; -- Default
    END IF;
ELSE
    SET NEW.Descrizione = NULL;
    SET NEW.Approvato = NULL;
1970 IF NEW.Nome IS NULL OR NEW.Telefono IS NULL OR NEW.Tavolo IS NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Nome, Telefono e Tavolo devono essere '
                        'specificati per le prenotazioni telefoniche.';

    END IF;
1975 END IF;

SET SalaLibera = (SELECT SUM(DATE(P.'Data') = DATE(NEW.'Data')) > 0
                  FROM Prenotazione P
                  WHERE P.Sala = NEW.Sala
1980                      AND P.Sede = NEW.Sede
                      AND P.Tavolo = NULL);

IF SalaLibera THEN
    SIGNAL SQLSTATE '45000'
1985     SET MESSAGE_TEXT = 'La sala scelta è già prenotata per un '
                        'allestimento.';

    END IF;

IF NEW.Tavolo IS NOT NULL THEN
1990     IF PostiTavolo < NEW.Numero THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Il tavolo scelto non ha un numero adeguato '
                            'di posti.';

        END IF;

1995     SET TavoloLibero = (SELECT SUM(P.'Data' >

```

```

2000         (NEW.'Data' - INTERVAL 2 HOUR)
              AND P.'Data' <
              (NEW.'Data' + INTERVAL 2 HOUR)) = 0
FROM Prenotazione P
WHERE P.Tavolo = NEW.Tavolo
      AND P.Sala = NEW.Sala
      AND P.Sede = NEW.Sede);

2005     IF NOT TavoloLibero THEN
          SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'Il tavolo scelto è già prenotato.';
        END IF;
    ELSE
2010     SET SalaLibera = (SELECT SUM(DATE(P.'Data') = DATE(NEW.'Data')) = 0
                          FROM Prenotazione P
                          WHERE P.Sala = NEW.Sala
                                AND P.Sede = NEW.Sede);

2015     IF NOT SalaLibera THEN
          SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'La sala contiene tavoli già prenotati.';
        END IF;
    END IF;
2020 END; $$

CREATE TRIGGER elimina_prenotazione
BEFORE DELETE
ON Prenotazione
2025 FOR EACH ROW
BEGIN
    IF CURRENT_DATETIME > (OLD.'Data' - INTERVAL 3 DAY) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Non è possibile annullare la prenotazione.';
2030    END IF;
END; $$

CREATE TRIGGER aggiorna_MV_ClientiPrenotazione_insert
AFTER INSERT
2035 ON Prenotazione
FOR EACH ROW
BEGIN
    INSERT INTO MV_ClientiPrenotazione(Sede, 'Data', Numero)
    VALUES (NEW.Sede, DATE(NEW.'Data'), NEW.Numero)
    ON DUPLICATE KEY
2040     UPDATE Numero = Numero + NEW.Numero;

```



```

END;$$

CREATE TRIGGER aggiorna_MV_ClientiPrenotazione_update
2045 AFTER UPDATE
ON Prenotazione
FOR EACH ROW
BEGIN
    IF NEW.Numero <> OLD.Numero THEN
2050        INSERT INTO MV_ClientiPrenotazione(Sede, 'Data', Numero)
        VALUES (NEW.Sede, DATE(NEW.'Data'), NEW.Numero)
        ON DUPLICATE KEY
            UPDATE Numero = Numero - OLD.Numero + NEW.Numero;
    END IF;
2055 END;$$

CREATE TRIGGER aggiorna_MV_ClientiPrenotazione_delete
AFTER DELETE
ON Prenotazione
2060 FOR EACH ROW
BEGIN
    UPDATE MV_ClientiPrenotazione
    SET Numero = Numero - OLD.Numero
    WHERE Sede = OLD.Sede
2065     AND 'Data' = DATE(OLD.'Data');
END;$$

CREATE TRIGGER nuovo_gradimento
BEFORE INSERT
2070 ON Gradimento
FOR EACH ROW
BEGIN
    IF (NEW.Proposta IS NOT NULL AND NEW.Suggerimento IS NOT NULL)
        OR (NEW.Proposta IS NULL AND NEW.Suggerimento IS NULL) THEN
2075        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Un gradimento deve riferirsi a una proposta o a '
        'un suggerimento. Non ad entrambi.';
    END IF;

    IF NEW.Punteggio < 1 OR NEW.Punteggio > 5 THEN
2080        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Punteggio deve essere compreso tra 1 e 5';
    END IF;
2085 END;$$

CREATE TRIGGER nuova_recensione

```

```

BEFORE INSERT
ON Recensione
FOR EACH ROW
2090 BEGIN
    IF NEW.Giudizio < 1 OR NEW.Giudizio > 5 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Giudizio deve essere compreso tra 1 e 5';
    END IF;
2095 END; $$

CREATE TRIGGER nuova_valutazione
BEFORE INSERT
ON Valutazione
2100 FOR EACH ROW
BEGIN
    IF NEW.Veridicita < 1 OR NEW.Veridicita > 5 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Veridicita deve essere compreso tra 1 e 5';
2105 END IF;
    IF NEW.Accuratezza < 1 OR NEW.Accuratezza > 5 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Accuratezza deve essere compreso tra 1 e 5';
    END IF;
2110 END; $$

CREATE TRIGGER aggiorna_ridondanza_Recensione
AFTER INSERT
ON Valutazione
2115 FOR EACH ROW
BEGIN
    UPDATE Recensione R
    SET R.VeridicitaTotale = R.VeridicitaTotale + NEW.Veridicita,
        R.AccuratezzaTotale = R.AccuratezzaTotale + NEW.Accuratezza,
2120     NumeroValutazioni = NumeroValutazioni + 1
    WHERE R.ID = NEW.Recensione;
END; $$

CREATE TRIGGER nuova_risposta
2125 BEFORE INSERT
ON Risposta
FOR EACH ROW
BEGIN
    IF NEW.Efficienza < 1 OR NEW.Efficienza > 5 THEN
2130     SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Efficienza deve essere compreso tra 1 e 5';

```

```

END IF;

-- Simula AUTO_INCREMENT
2135 IF NEW.Numero IS NULL THEN
        SET NEW.Numero = (SELECT IFNULL(MAX(Numero), 0) + 1
                           FROM Risposta
                           WHERE Domanda = NEW.Domanda);
END IF;
2140 END;$$

-- EVENTS
SELECT "Creazione events." AS "**** FASE 9 ****";

2145 CREATE EVENT aggiorna_MV_OrdiniRicetta_MenuCorrente
ON SCHEDULE
EVERY 1 DAY
STARTS TIMESTAMP(CURRENT_DATE) + INTERVAL 1 DAY + INTERVAL 2 HOUR
2150 ON COMPLETION PRESERVE
DO
BEGIN
    DECLARE cSede VARCHAR(45);
    DECLARE cRicetta VARCHAR(45);
2155 DECLARE cOrdini INT;
    DECLARE cNovita BOOL;
    DECLARE Finito BOOL DEFAULT FALSE;
    DECLARE curPiatto CURSOR FOR
        SELECT C.Sede, P.Ricetta, COUNT(*) AS Ordini
2160 FROM Piatto P INNER JOIN Comanda C ON P.Comanda = C.ID
        WHERE DATE(C.'Timestamp') = (CURRENT_DATE - INTERVAL 1 DAY)
        GROUP BY C.Sede, P.Ricetta;
    DECLARE curElenco CURSOR FOR
        SELECT M.Sede, E.Ricetta, E.Novita
2165 FROM Elenco E INNER JOIN Menu M ON E.Menu = M.ID
        WHERE CURRENT_DATE
            BETWEEN M.DataInizio AND M.DataFine;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET Finito = TRUE;

2170 OPEN curPiatto;

loop_lbl: LOOP
    FETCH curPiatto INTO cSede, cRicetta, cOrdini;
    IF Finito THEN
2175 LEAVE loop_lbl;
    END IF;

```

```

        INSERT INTO MV_OrdiniRicetta(Sede, Ricetta, TotOrdini) VALUES
            (cSede, cRicetta, cOrdini)
2180         ON DUPLICATE KEY
            UPDATE Compare = Compare + 1, TotOrdini = TotOrdini + cOrdini;
    END LOOP loop_lbl;

    CLOSE curPiatto;
2185

    SET Finito = FALSE;
    TRUNCATE TABLE MV_MenuCorrente; -- full refresh
2190    OPEN curElenco;

    loop2_lbl: LOOP
        FETCH curElenco INTO cSede, cRicetta, cNovita;
        IF Finito THEN
2195             LEAVE loop2_lbl;
        END IF;

        IF IngredientiDisponibili(cSede, cRicetta, cNovita, NULL) THEN
            INSERT INTO MV_MenuCorrente(Sede, Ricetta, Novita) VALUES
2200                (cSede, cRicetta, cNovita);
        END IF;
    END LOOP loop2_lbl;

    CLOSE curElenco;
2205 END; $$

CREATE EVENT aggiorna_elenco_novita
ON SCHEDULE
EVERY 1 DAY
2210 STARTS TIMESTAMP(CURRENT_DATE) + INTERVAL 1 DAY + INTERVAL '1:30' HOUR_MINUTE
ON COMPLETION PRESERVE
DO
BEGIN
    DECLARE Finito BOOL DEFAULT FALSE;
2215    DECLARE cSede VARCHAR(45);
    DECLARE cRicetta VARCHAR(45);
    DECLARE RimuoviNovita BOOL;
    DECLARE curMenu CURSOR FOR
        SELECT MC.Sede, MC.Ricetta
2220        FROM MV_MenuCorrente MC
        WHERE MC.Novita = TRUE;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET Finito = TRUE;

```

```

OPEN curMenu;
2225
loop_lbl: LOOP
    FETCH curMenu INTO cSede, cRicetta;
    IF Finito THEN
        LEAVE loop_lbl;
2230
    END IF;

    SET RimuoviNovita = (SELECT (COUNT(*) > 4) AS RimuoviNovita
                        FROM MV_OrdiniRicetta MVOR
                        WHERE MVOR.Ricetta = cRicetta
                        AND MVOR.Sede = cSede);
2235

    IF RimuoviNovita THEN
        UPDATE Elenco E INNER JOIN Menu M ON E.Menu = M.ID
        SET E.Novita = FALSE
        WHERE M.Sede = cSede AND E.Ricetta = cRicetta;
2240
    END IF;
END LOOP loop_lbl;

CLOSE curMenu;
END;$$

2245
CREATE EVENT Analytics_Scheduler
ON SCHEDULE
EVERY 1 MONTH
STARTS TIMESTAMP(CURRENT_DATE) + INTERVAL 1 DAY + INTERVAL 4 HOUR
2250
ON COMPLETION PRESERVE
DO
BEGIN
    CALL AnalizzaRecensioni();
    CALL AnalizzaVendite(CURRENT_TIMESTAMP - INTERVAL 1 MONTH, NULL);
2255
    CALL AnalizzaSuggerimenti();
    CALL AnalizzaProposte();
END;$$

CREATE EVENT Elenco_Ordini
2260
ON SCHEDULE
EVERY 1 WEEK
STARTS TIMESTAMP(CURRENT_DATE) + INTERVAL 5 HOUR
ON COMPLETION PRESERVE
DO
2265
BEGIN
    DECLARE ClientiPrenotazioni INT;
    DECLARE MediaSenzaPrenotazione INT;
    DECLARE StimaClienti INT;

```

```

2270 DECLARE StimaOrdini INT;
2270 DECLARE NomeSede VARCHAR(45);
2270 DECLARE NomeRicetta VARCHAR(45);
2270 DECLARE RicettaNovita BOOL;
2270 DECLARE Finito BOOL DEFAULT FALSE;
2270 DECLARE curRicetta CURSOR FOR
2275     SELECT M.Sede, E.Ricetta, E.Novita
        FROM Menu M INNER JOIN Elenco E ON M.ID = E.Menu
        WHERE M.DataInizio <= CURRENT_DATE
            AND M.DataFine >= CURRENT_DATE + INTERVAL 6 DAY;
2280 DECLARE CONTINUE HANDLER FOR NOT FOUND SET Finito = TRUE;

TRUNCATE TABLE Report_Ordinativi;

OPEN curRicetta;

2285 loop_lbl: LOOP
    FETCH curRicetta INTO NomeSede, NomeRicetta, RicettaNovita;
    IF Finito THEN
        LEAVE loop_lbl;
    END IF;

2290     SET ClientiPrenotazioni = (SELECT COALESCE(CP.Numero, 0)
                                FROM MV_ClientiPrenotazione CP
                                WHERE CP.Sede = NomeSede
                                    AND CP.'Data' BETWEEN CURRENT_DATE
2295                                     AND CURRENT_DATE + INTERVAL 6 DAY);

    SET MediaSenzaPrenotazione = (
        SELECT CEIL(COALESCE(AVG(CL.SenzaPrenotazione), 0)/4) AS Media
        FROM Clienti_Log CL
2300     WHERE CL.Sede = NomeSede
            AND CL.Mese = MONTH(CURRENT_DATE)
            AND CL.Anno <> YEAR(CURRENT_DATE)
        );

2305     SET StimaClienti = ClientiPrenotazioni + MediaSenzaPrenotazione;

    IF RicettaNovita THEN
        SET StimaOrdini = (SELECT CEIL(StimaClienti * 0.33));
    ELSE
2310     SET StimaOrdini = (
        SELECT (COALESCE(CEIL(MV.TotOrdini / MV.Comparsa), 0)
            + StimaClienti * 0.1) AS StimaOrdini
        FROM MV_OrdiniRicetta MV
        WHERE MV.Sede = NomeSede AND MV.Ricetta = NomeRicetta

```

```

2315         );
        END IF;

        IF StimaOrdini < 5 THEN
            SET StimaOrdini = 5;
2320        END IF;

        INSERT INTO Report_Ordinativi(Sede, Ingrediente, Quantita)
        SELECT NomeSede, F.Ingrediente, SUM(F.Dose)*StimaOrdini AS Qta
        FROM Fase F
2325        WHERE F.Ricetta = NomeRicetta
        ON DUPLICATE KEY UPDATE Quantita = Quantita + VALUES(Quantita);
    END LOOP loop_lbl;

    CLOSE curRicetta;
2330 END;$$

CREATE EVENT aggiorna_Report_TakeAway
ON SCHEDULE
EVERY 1 WEEK
2335 STARTS TIMESTAMP(CURRENT_DATE) + INTERVAL 1 DAY + INTERVAL 3 HOUR
ON COMPLETION PRESERVE
DO
BEGIN
    DECLARE TempoMedioAndata INT;
2340    DECLARE TempoMedioRitorno INT;

    TRUNCATE TABLE Report_TakeAway;

    SELECT CEIL(AVG(TIMESTAMPDIFF(SECOND, C.Partenza, C.Arrivo))) AS TMAndata,
2345           CEIL(AVG(TIMESTAMPDIFF(SECOND, C.Arrivo, C.Ritorno))) AS TMRitorno
    INTO TempoMedioAndata, TempoMedioRitorno
    FROM Consegna C
    WHERE C.Ritorno IS NOT NULL;

    IF TempoMedioAndata IS NULL OR TempoMedioRitorno IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Dati insufficienti per la generazione di '
        'Report_TakeAway.';
    END IF;
2355

    INSERT INTO Report_TakeAway(Posizione, Sede, Pony, DeltaTempoAndata,
                                DeltaTempoRitorno)
    SELECT @row_number := @row_number + 1 AS Posizione, D.*
    FROM (SELECT @row_number := 0) AS N,
2360    (

```

```
2365         SELECT P.Sede, P.ID AS Pony,
           SEC_TO_TIME(
             CEIL(AVG(TIMESTAMPDIFF(SECOND, C.Partenza, C.Arrivo))) -
             TempoMedioAndata
           ) AS DeltaTempoAndata,
           SEC_TO_TIME(
             CEIL(AVG(TIMESTAMPDIFF(SECOND, C.Arrivo, C.Ritorno))) -
             TempoMedioRitorno
           ) AS DeltaTempoRitorno
2370       FROM Pony P INNER JOIN Consegna C
           WHERE C.Ritorno IS NOT NULL
           GROUP BY P.Sede, P.ID
           ) AS D
           ORDER BY (D.DeltaTempoAndata + D.DeltaTempoRitorno) ASC;
2375 END; $$

DELIMITER ;

2380 -- INSERTS NON RIPORTATI NEL DOCUMENTO

SELECT "Esecuzione script terminata. Bye bye ;)"
       AS "***** END OF SCRIPT *****";
```