Basi di Dati

Progetto A.A. 2021/2022

CHAT MULTICANALE

0268603

Marta Fraioli

Indice

1. Descrizione del Minimondo

Si vuole realizzare un sistema informativo per consentire ai lavoratori di una azienda di scambiare messaggi legati ai progetti a cui stanno attualmente lavorando.

Il sistema prevede tre livelli di utenza: gli amministratori, i capi progetto, i dipendenti.

Gli amministratori hanno la possibilità di gestire quali utenti sono capi progetto.

I capi progetto possono creare un numero arbitrario di canali di comunicazione ed invitare al loro interno tutti i dipendenti che cooperano sulle attività del progetto.

All'interno di un canale gli utenti possono inviare messaggi che possono essere letti da tutti gli altri appartenenti al canale.

I messaggi sono organizzati in pagine e gli utenti possono visualizzare, una per una, le pagine della conversazione.

Un utente del sistema ha la possibilità di rispondere pubblicamente in un canale.

In questa risposta, può decider di riferire un messaggio precedentemente inviato, così che il suo messaggio appaia come risposta ad una parte specifica della comunicazione.

Allo stesso modo, partendo da un qualsiasi messaggio, l'utilizzatore può decidere di rispondere in modo privato.

Questa risposta privata aprirà un canale di discussione privato tra lui e il mittente del messaggio cui si sta rispondendo.

I project manager possono sempre accedere (in sola lettura) a tutte le discussioni private nate nei canali dei progetti di cui sono responsabili.

Un utente può accedere in qualsiasi momento a tutti i canali di cui fa parte, e a tutte le conversazioni private (organizzate per canale) che ha creato o in cui è stato coinvolto.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
7-11	Utente	Dipendente Capoprogetto	La facoltà di scrivere messaggi riguarda solamente dipendenti e capiprogetto e non l'utenza in generale che comprende anche gli amministratori

Specifica disambiguata

Si vuole realizzare un sistema informativo per consentire ai lavoratori di una azienda di scambiare messaggi legati ai progetti a cui stanno attualmente lavorando.

Il sistema prevede tre livelli di utenza: gli amministratori, i capi progetto, i dipendenti.

Gli amministratori hanno la possibilità di gestire quali utenti sono capi progetto.

I capi progetto possono creare un numero arbitrario di canali di comunicazione ed invitare al loro interno tutti i dipendenti che cooperano sulle attività del progetto.

All'interno di un canale dipendenti e capiprogetto possono inviare messaggi che possono essere letti da tutti gli altri appartenenti al canale.

I messaggi sono organizzati in pagine e gli utenti possono visualizzare, una per una, le pagine della conversazione.

Dipendenti e capiprogetto hanno la possibilità di rispondere pubblicamente in un canale.

In questa risposta, può decider di riferire un messaggio precedentemente inviato, così che il suo messaggio appaia come risposta ad una parte specifica della comunicazione.

Allo stesso modo, partendo da un qualsiasi messaggio, l'utilizzatore può decidere di rispondere in modo privato.

Questa risposta privata aprirà un canale di discussione privato tra lui e il mittente del messaggio cui si sta rispondendo.

I project manager possono sempre accedere (in sola lettura) a tutte le discussioni private nate nei

canali dei progetti di cui sono responsabili.

Un utente può accedere in qualsiasi momento a tutti i canali di cui fa parte, e a tutte le conversazioni private (organizzate per canale) che ha creato o in cui è stato coinvolto.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Lavoratore	Impiegato dell'azienda	Utente	Messaggio, progetto
Amministratore	Impiegato incaricato di coordinare i capiprogetto		Lavoratore, capoprogetto
Capoprogetto	Impiegato incaricato di coordinare tutte le attività afferenti a determinati progetti	Utilizzatore, Project Manager	Lavoratore, Canale di comunicazione, Messaggio, Progetto
Dipendente	Impiegato incaricato di svolgere determinate attività dei progetti a cui è stato assegnato	Utilizzatore	Lavoratore, Canale di comunicazione, Messaggio, Progetto
Canale di comunicazione	Raccolta di messaggi di un numero arbitrario di dipendenti che lavorano su un determinato progetto	Conversazione, Discussione	Messaggio, Progetto, Lavoratore
Messaggio	Testo di lunghezza arbitraria che può essere scambiato tra dipendenti che lavorano su un determinato progetto	Risposta	Capoprogetto, Dipendente, Progetto
Progetto	Complesso di attività assegnate ad un gruppo di dipendenti		Capoprogetto, Dipendente, Canale di comunicazione

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative ai lavoratori
Il sistema prevede tre livelli di utenza: gli amministratori, i capi progetto, i dipendenti. Un utente può accedere in qualsiasi momento a tutti i canali di cui fa parte, e a tutte le conversazioni private (organizzate per canale) che ha creato o in cui è stato coinvolto.

Frasi relative agli amministratori
Gli amministratori hanno la possibilità di gestire quali utenti sono capi progetto.

Frasi relative ai capiprogetto
I capi progetto possono creare un numero arbitrario di canali di comunicazione ed invitare al loro interno tutti i dipendenti che cooperano sulle attività del progetto. All'interno di un canale dipendenti e capiprogetto possono inviare messaggi che possono essere letti da tutti gli altri appartenenti al canale. Dipendenti e capiprogetto hanno la possibilità di rispondere pubblicamente in un canale. Allo stesso modo, partendo da un qualsiasi messaggio, l'utilizzatore può decidere di rispondere in modo privato. I project manager possono sempre accedere (in sola lettura) a tutte le discussioni private nate nei canali dei progetti di cui sono responsabili.

Frasi relative ai dipendenti
All'interno di un canale dipendenti e capiprogetto possono inviare messaggi che possono essere letti da tutti gli altri appartenenti al canale. Dipendenti e capiprogetto hanno la possibilità di rispondere pubblicamente in un canale. Allo stesso modo, partendo da un qualsiasi messaggio, l'utilizzatore può decidere di rispondere in modo privato.

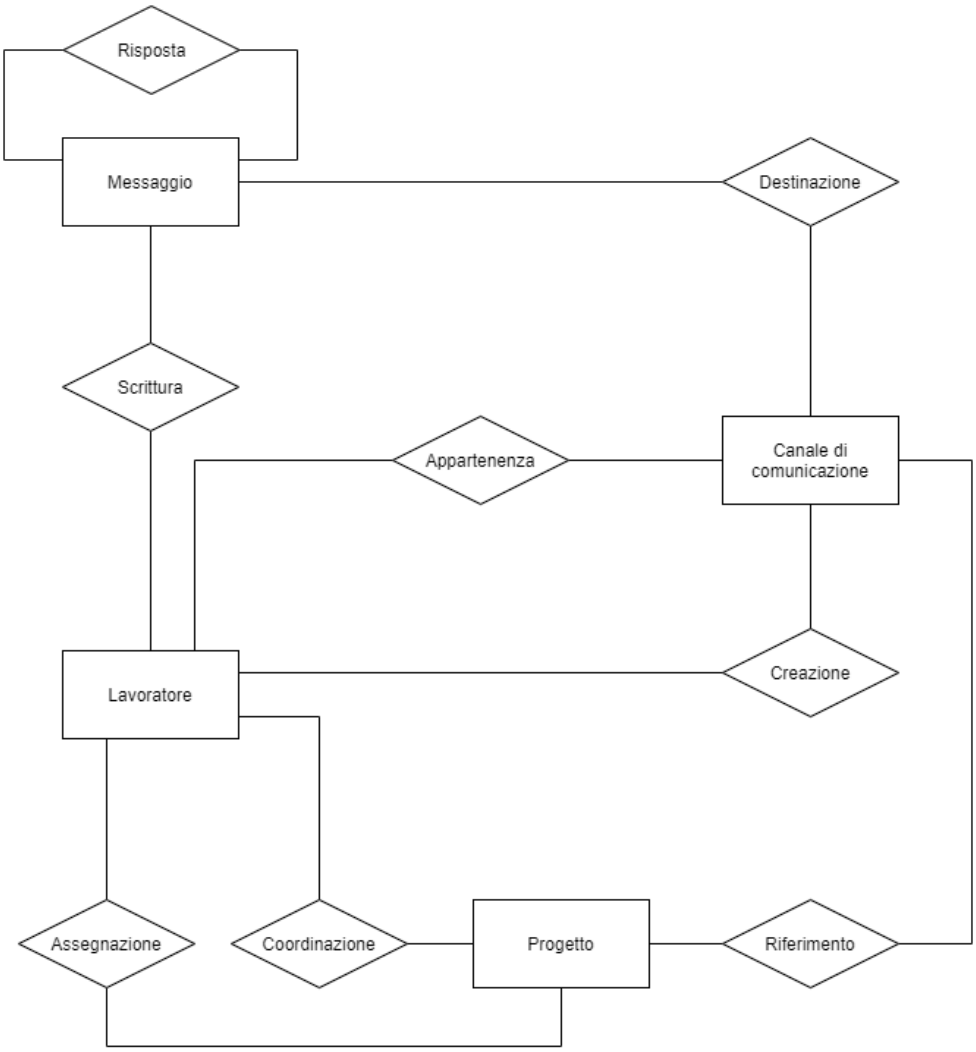
Frasi relative ai messaggi
I messaggi sono organizzati in pagine e gli utenti possono visualizzare, una per una, le pagine della conversazione. Si può decider di riferire un messaggio precedentemente inviato, così che il messaggio appaia come risposta ad una parte specifica della conversazione. Una risposta privata aprirà un canale di discussione privato tra l'utente e il mittente del messaggio cui si sta rispondendo.

3. Progettazione concettuale

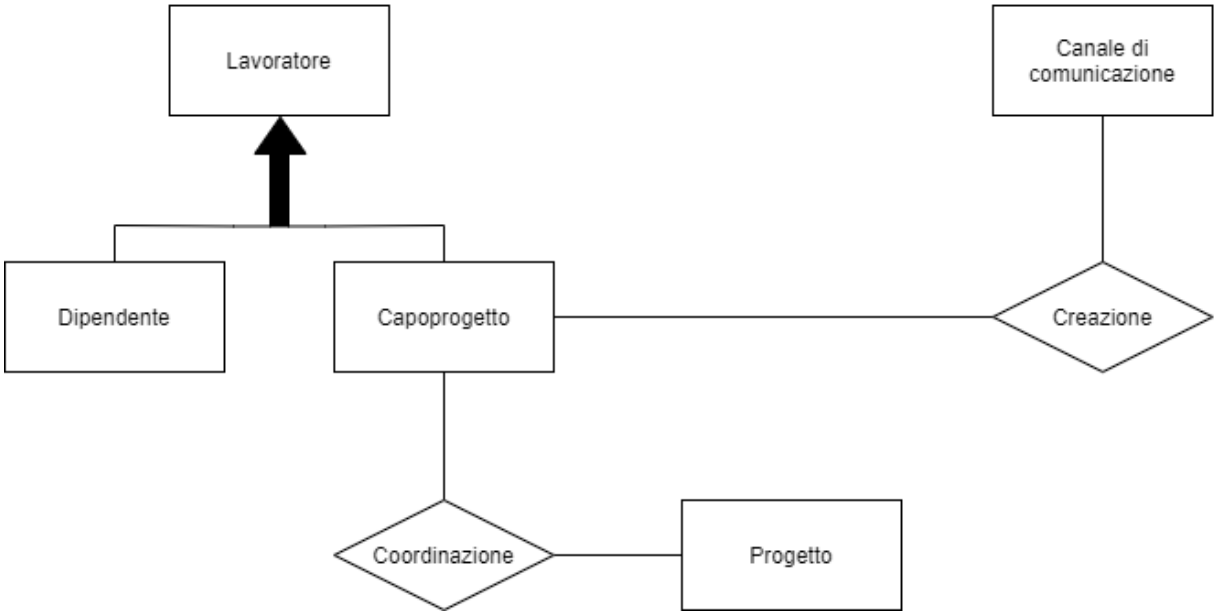
Costruzione dello schema E-R

Dal momento che le richieste del committente si presentano lineari e concise, è stato possibile individuare nell'immediato i concetti da rappresentare con le relazioni che intercorrono tra di loro. Per questo motivo è stato scelto un approccio top-down per costruire lo schema E-R.

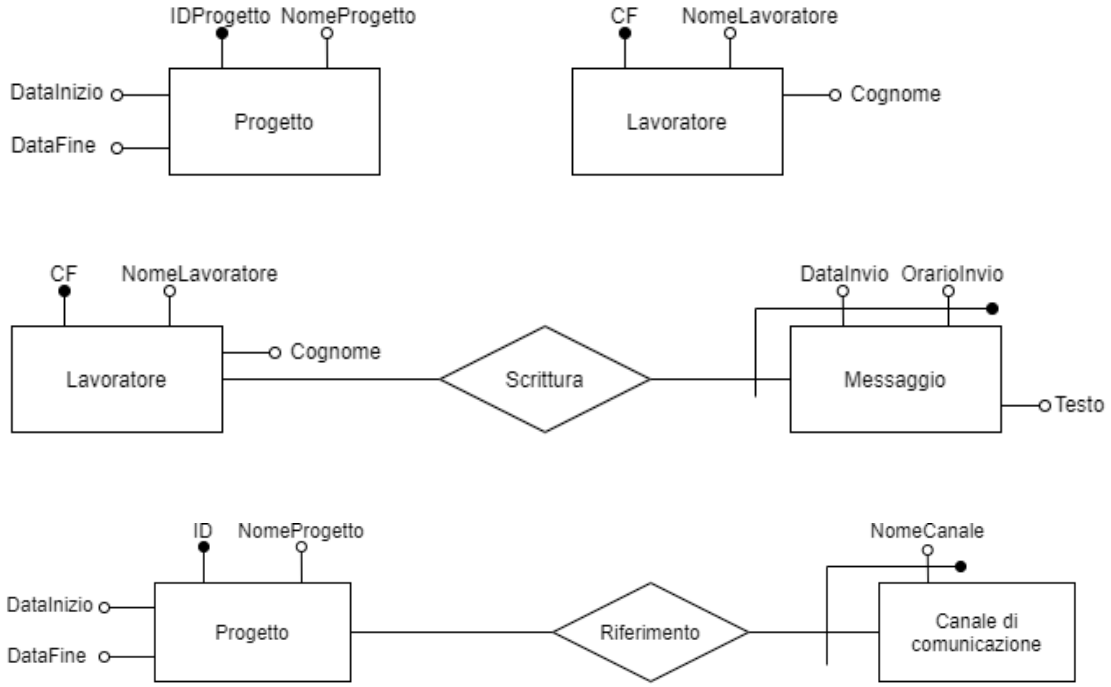
Entità ed associazioni dello schema scheletro sono state denominate a seguito del glossario dei termini nell'analisi dei requisiti e presentate in modo astratto.



Il primo raffinamento è stato sull'entità lavoratore, trasformandola in una gerarchia di generalizzazione per renderere possibile la distinzione dei ruoli che un utente interno all'azienda può ricoprire e delle azioni che quest'ultimo può intraprendere.

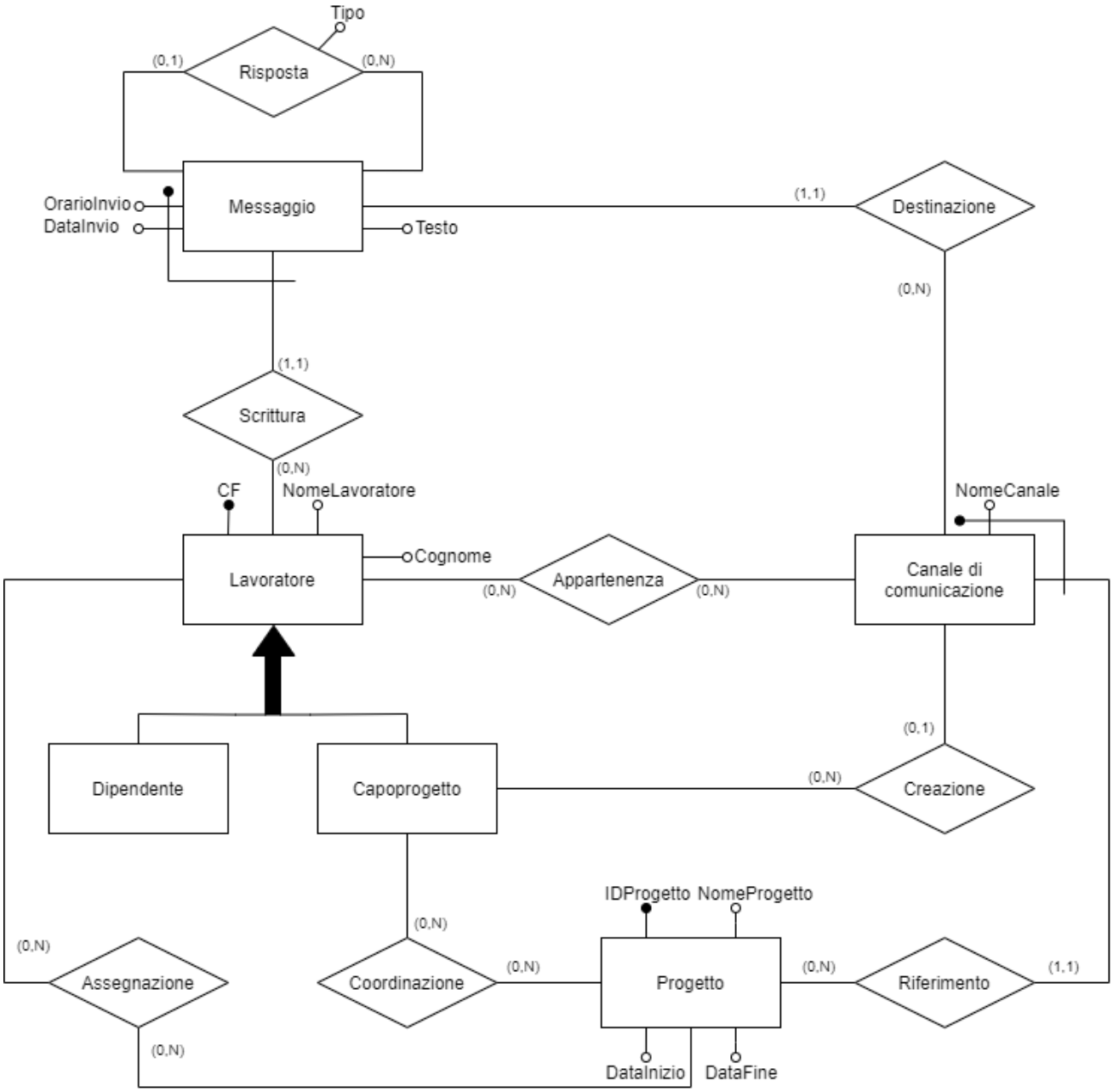


In un secondo momento sono stati definiti eventuali attributi per ogni entità e relazione. Dato che nella descrizione del minimondo affidata non ne erano esplicitamente presenti, ne sono stati scelti quanti bastano ad una identificazione dell'oggetto e al corretto funzionamento dell'applicativo.



Integrazione finale

Poiché ogni concetto era presente dalla prima stesura dello schema non sono stati necessari passi di integrazione, né si sono presentati conflitti strutturali o sui nomi.



Regole aziendali

1. Solamente una risposta di tipo “privato” può generare un canale di comunicazione; quest’ultimo avrà come partecipanti mittente e destinatario, il progetto di riferimento sarà quello del canale di comunicazione dal quale ha avuto origine.
2. La data di inizio di un progetto deve precedere temporalmente quella di fine dello stesso.
3. Un amministratore non può accedere in scrittura in nessun canale.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Lavoratore	Impiegato dell’azienda	CF, NomeLavoratore, Cognome	CF
Capoprogetto	Impiegato dell’azienda incaricato di coordinare alcuni progetti e creare i canali di comunicazione afferenti a quest’ultimi	CF, NomeLavoratore, Cognome	CF
Dipendente	Impiegato dell’azienda che lavora su determinati progetti	CF, NomeLavoratore, Cognome	CF
Canale di comunicazione	Stanza virtuale legata ad un progetto in cui i lavoratori possono scambiare dei messaggi	NomeCanale	NomeCanale, IDProgetto
Messaggio	Comunicazioni di testo, pubbliche o private, relative ad un progetto	DataInvio, OrarioInvio, Testo	DataInvio, OrarioInvio, CF
Progetto	Complesso di attività affidato ad un gruppo di lavoratori	IDProgetto, NomeProgetto, DataInizio, DataFine	IDProgetto

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Lavoratore	E	1000
Capoprogetto	E	100
Dipendente	E	900
Canale di comunicazione	E	1250
Messaggio	E	1500000
Progetto	E	250
Appartenenza	R	5000
Assegnazione	R	1500
Coordinazione	R	300
Creazione	R	750
Destinazione	R	2000000
Riferimento	R	1250
Risposta	R	300000
Scrittura	R	2000000

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
OP1	Inserimento di un lavoratore	50/anno
OP2	Inserimento di un progetto	15/mese
OP3	Assegnazione di un lavoratore ad un progetto	100/mese
OP4	Assegnazione di un capoprogetto ad un progetto	20/mese

[1] Indicare con E le entità, con R le relazioni

OP5	Scrittura di un messaggio	4000/giorno
OP6	Risposta ad uno specifico messaggio	1000/giorno
OP7	Creazione di un canale di comunicazione	45/mese
OP8	Creazione di un canale di comunicazione privato	30/mese
OP9	Assegnazione di un lavoratore ad un canale di comunicazione	250/mese
OP10	Stampa dei progetti a cui un dipendente è stato assegnato con i relativi canali di comunicazione a cui può accedere	1000/giorno
OP11	Stampa delle conversazioni di un canale di comunicazione	1500/giorno

Costo delle operazioni

f_i = frequenza OP_i

OP1: L'operazione richiede un solo accesso in scrittura all'entità "lavoratore".
Totale: $2*1*f_1 = 100$ accessi / anno.

OP2: L'operazione richiede un solo accesso in scrittura all'entità "progetto".
Totale: $2*1*f_2 = 30$ accessi / mese.

OP3: L'operazione richiede: un accesso in lettura all'entità "lavoratore", un accesso in scrittura alla relazione "appartenenza".
Totale: $(1+2*1)*f_3 = 300$ accessi / mese.

OP4: L'operazione richiede: un accesso in lettura all'entità capoprogetto, un accesso in scrittura alla relazione "coordinazione".
Totale: $(1+2*1)*f_4 = 60$ accessi / mese.

OP5: L'operazione richiede: un accesso in scrittura all'entità "messaggio", un accesso in scrittura alla relazione "scrittura", un accesso in scrittura alla relazione "destinazione".
Totale: $(2*1+2*1+2*1)*f_5 = 24000$ accessi /giorno.

OP6: L'operazione richiede: un accesso in scrittura all'entità "messaggio", un accesso in scrittura alla relazione "scrittura", un accesso in scrittura alla relazione "destinazione", un accesso in scrittura alla relazione "risposta".
Totale: $(2*1+2*1+2*1+2*1)*f_6 = 8000$ accessi /giorno.

OP7: L'operazione richiede: un accesso in scrittura all'entità "canale di comunicazione", un accesso in scrittura alla relazione "creazione", un accesso in scrittura alla relazione "riferimento".
Totale: $(2*1+2*1+2*1)*f_7 = 210$ accessi / mese.

OP8: L'operazione richiede: un accesso in scrittura all'entità "canale di comunicazione",un accesso in scrittura alla relazione "riferimento", due accessi in scrittura alla relazione appartenenza.
Totale: $(1+2*1+2*2)*f_8 = 270$ accessi / mese.

OP9: L'operazione richiede: un accesso in lettura all'entità lavoratore, un accesso in scrittura alla relazione "appartenenza".
Totale: $(1+2*1)*f_9 = 750$ accessi / mese.

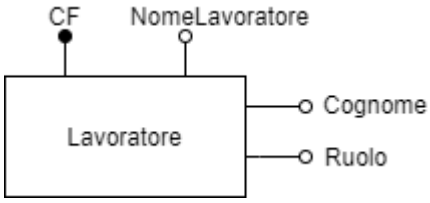
OP10: L'operazione richiede: un accesso in lettura all'entità "lavoratore", tre accessi in lettura alla relazione "assegnazione", tre accessi in lettura all'entità "progetto", quindici accessi in lettura alla relazione "riferimento", quindici accessi in lettura all'entità "canale di comunicazione", nove accessi in lettura alla relazione "appartenenza".
Si tenga in considerazione che:
- un dipendente lavora/ha lavorato su un massimo di tre progetti
- ogni progetto ha in media cinque canali di comunicazione
- un dipendente partecipa in media a tre canali di comunicazione per ogni progetto
Totale $(1+3*1+3*1+15*1+15*1+9*1)*f_{10} = 46000$ accessi / giorno.

OP11: L'operazione richiede: un accesso in lettura all'entità "canale di comunicazione", 1000 accessi in lettura alla relazione "destinazione", 1000 accessi in lettura all'entità "messaggio". "1000" è un valore ragionevole tra la media ponderata e la media aritmetica del numero di messaggi presente in un canale di conversazione.
Totale: $(1+1*1000+1*1000)*f_{11} = 3001500$ accessi / giorno.

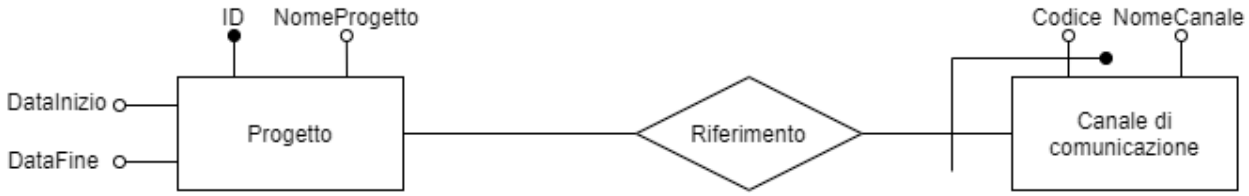
Ristrutturazione dello schema E-R

Nello schema è presente una sola generalizzazione che coinvolge le entità "lavoratore", "dipendente" e "capoprogetto". Tre relazioni ("assegnazione", "appartenenza", "scrittura") coinvolgono entrambe

le sottocategorie, mentre le relazioni “creazione” e “coordinazione” sono una peculiarità del “capoprogetto”. Poiché l’entità “dipendente” non ha alcuna relazione esclusiva associata ed entrambe le sottocategorie condividono gli stessi attributi (assenza di eventuali valori nulli), è stato scelto di accorpare le entità figlie nell’entità genitore, avendo così solamente l’entità “lavoratore” a cui va aggiunto un attributo “ruolo” per distinguere il tipo di occorrenza e l’aggiunta del vincolo sulle regole aziendali che permette solamente alle entità “lavoratore” con l’attributo “ruolo” corrispondente al valore “capoprogetto” di partecipare alle relazioni “creazione” e “coordinazione”.



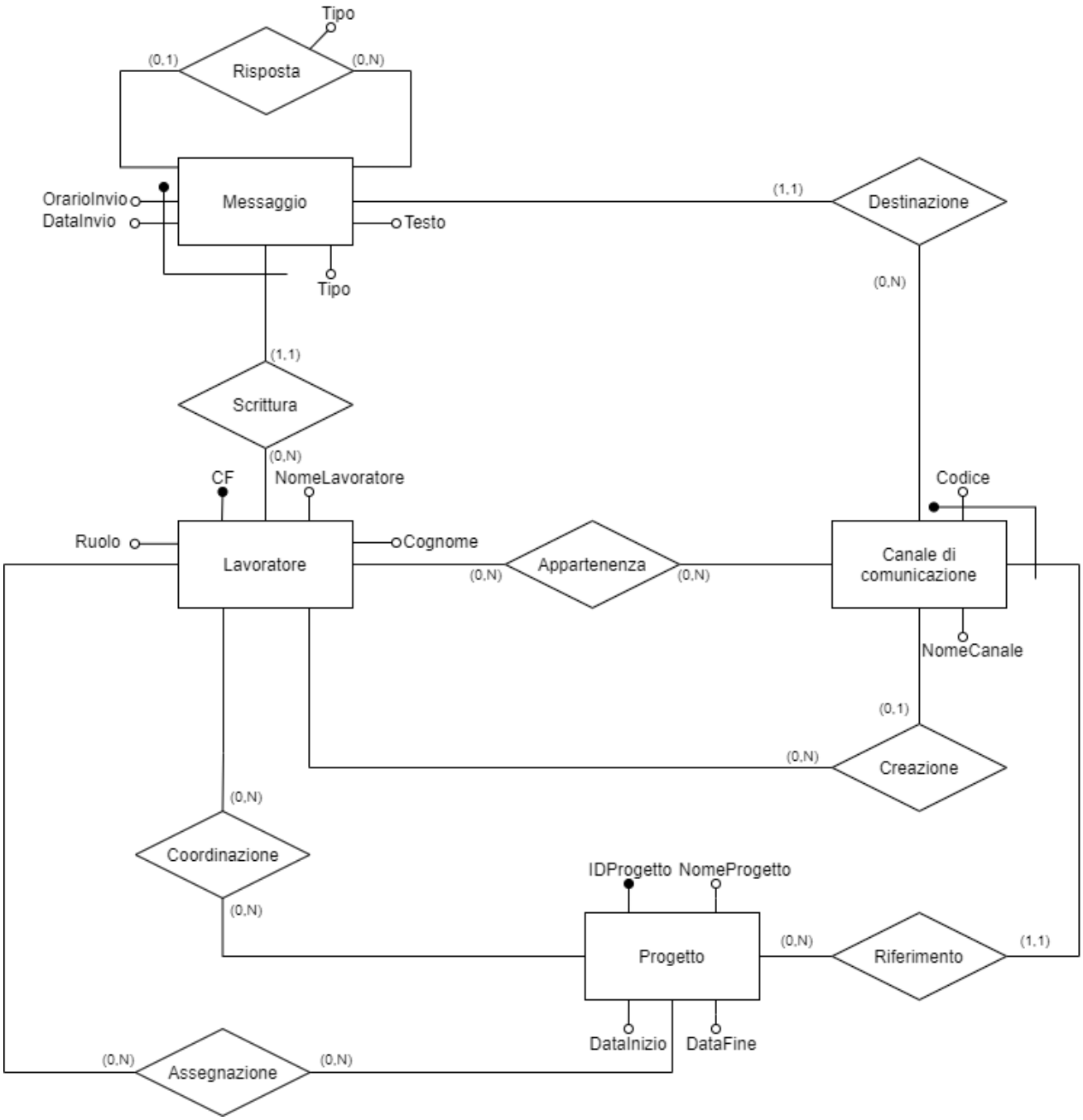
La scelta dell’identificare un canale di comunicazione tramite il nome dello stesso risulterebbe poco pratica nel caso in cui si voglia attribuire nome identico a due canali riferenti lo stesso progetto. Risulta invece più ragionevole scegliere un codice univoco legato sempre al progetto.



Dal momento che non sono presenti né attributi composti né informazioni ridondanti, non è stata ritenuta opportuna nessun’altra operazione di ristrutturazione.

Si propone di seguito lo schema ER con apportate le modifiche sopra elencate.

SCHEMA FINALE



Trasformazione di attributi e identificatori

Nella traduzione della relazione ricorsiva "risposta" è stato aggiunto il suffisso "mittente" e "destinatario" per distinguere gli attributi delle entità partecipanti.

Traduzione di entità e associazioni

Lavoratore (CF, NomeLavoratore, Cognome, Ruolo)

Progetto (IDProgetto, NomeProgetto, DataInizio, DataFine)

Canale di comunicazione (Codice, IDProgetto, NomeCanale)

Messaggio (OrarioInvio, DataInvio, CF, Testo, Codice, IDProgetto)

Appartenenza (CF, Codice, IDProgetto)

Assegnazione (CF, IDProgetto)

Coordinazione (CF, , IDProgetto)

Creazione (Codice, IDProgetto, CF)

Risposta (OrarioInvioMittente, DataInvioMittente, CFMittente, OrarioInvioDestinatario, DataInvioDestinatario, CFDestinatario, Tipo)

Canale di comunicazione(IDProgetto) ⊆ Progetto(IDProgetto)

Messaggio(CF) ⊆ Lavoratore(CF)

Messaggio(Codice, IDProgetto) ⊆ Canale di comunicazione(Codice, IDProgetto)

Appartenenza(Codice, IDProgetto) ⊆ Canale di comunicazione(Codice, IDProgetto)

Appartenenza(CF) ⊆ Lavoratore(CF)

Assegnazione(CF) ⊆ Lavoratore(CF)

Assegnazione(IDProgetto) ⊆ Progetto(IDProgetto)

Coordinazione(CF) ⊆ Lavoratore(CF)

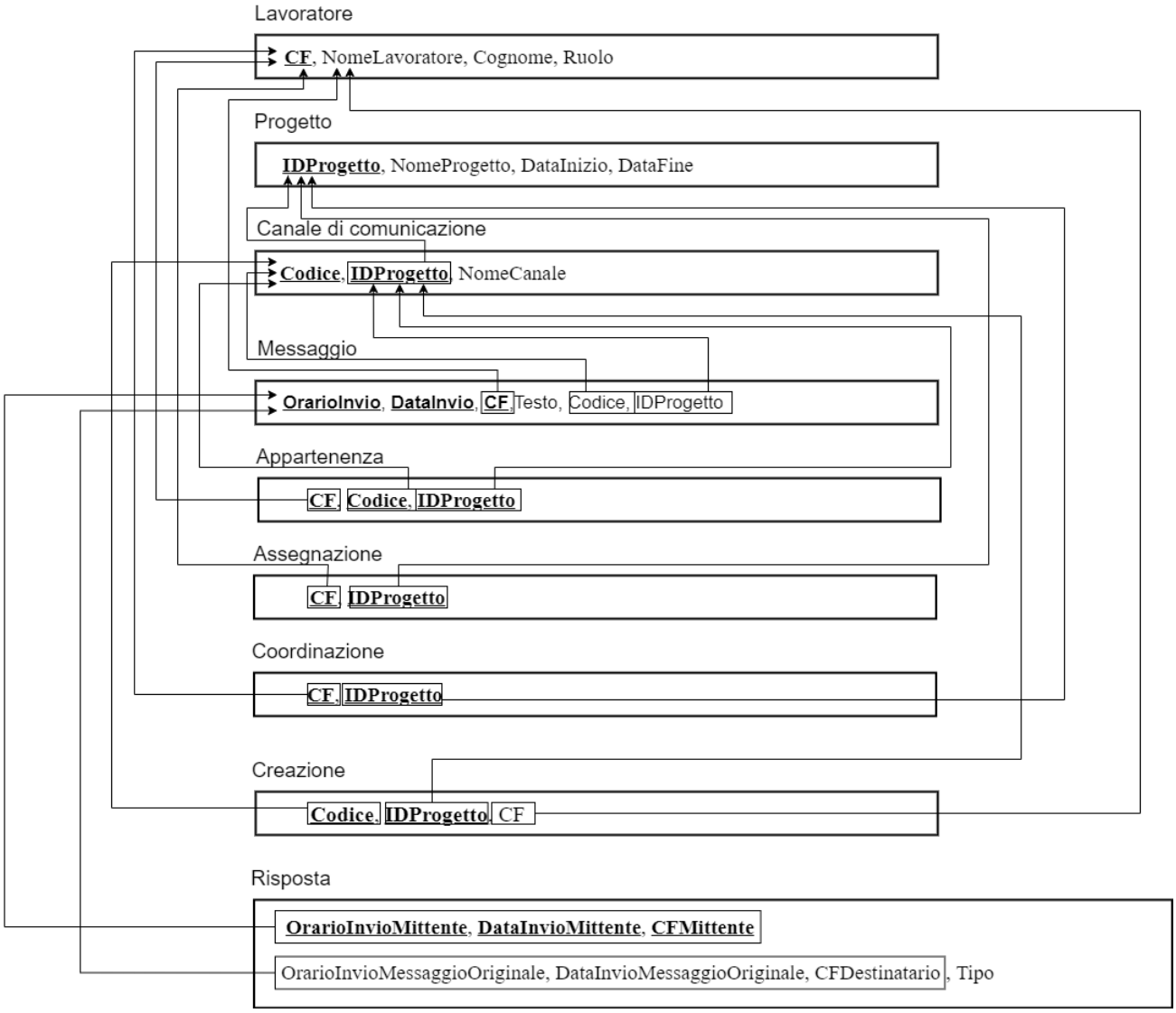
Coordinazione(IDProgetto) ⊆ Progetto(IDProgetto)

Creazione(Codice, IDProgetto) ⊆ Canale di comunicazione(Codice, IDProgetto)

Creazione(CF) ⊆ Lavoratore(CF)

Risposta(OrarioInvioMittente, DataInvioMittente, CFMittente) ⊆
Messaggio(OrarioInvio,DataInvio,CF)

Risposta(OrarioInvioDestinatario, DataInvioDestinatario, CFDestinatario) ⊆
Messaggio(OrarioInvio,DataInvio,CF)



Normalizzazione del modello relazionale

Tutte le tabelle rispettano la 3FN.

5. Progettazione fisica

Utenti e privilegi

Sono state previste tre tipologie di utente: amministratori, capoprogetto e dipendenti, necessitano di autenticazione tramite codice fiscale e password dato che non è possibile utilizzare l'applicazione al di fuori del contesto aziendale e/o "anonimamente". Gli utenti effettivamente inserti sono stati quattro, comprendendo un utente login in grado di eseguire unicamente la procedure login, per permettere la verifica del codice fiscale e della password di un utente in fase di login. I privilegi sono limitati all'esecuzione di procedure.

Login: login
Amministratore: chiusura_progetto, coordinazione_progetto (assegnazione della relazione di coordinazione), insert_lavoratore, insert_utente, insert_progetto, retrieve conversazioni, retrieve_dipendenti, retrieve_progetti_canali
Capoprogetto: assegnazione_canale, assegnazione_progetto, creazione_canale, insert_messaggio, retrieve_appartenenza, retrieve_appartenenza_coordinazione, retrieve_conversazioni, retrieve_coordinazione, retrieve_dipendenti, risposta_privata, risposta_pubblica
Dipendente: insert_messaggio, retrieve_appartenenza, retrieve_conversazioni, risposta_privata, risposta_pubblica

Strutture di memorizzazione

Tabella Utente		
Colonna	**Tipo di dato**	**Attributi**[2]
U_CF	Varchar(45)	PK, NN
Password	Varchar(45)	NN
Ruolo	ENUM('Amministratore', 'Capoprogetto', 'Dipendente')	NN
Tabella Lavoratore		
Colonna	**Tipo di dato**	**Attributi**
CF	Varchar(45)	PK, NN
NomeLavoratore	Varchar(45)	NN

[2] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

CognomeLavoratore	Varchar(45)	NN
Ruolo	TinyINT	NN
Tabella Progetto		
Colonna	**Tipo di dato**	**Attributi**
IDProgetto	Varchar(45)	PK, NN,AI
NomeProgetto	Varchar(45)	
DataInizio	Date	NN
DataFine	Date	
Tabella CanaleDiComunicazione		
Colonna	**Tipo di dato**	**Attributi**
Codice	INT	PK, NN,AI
Progetto_IDProgetto	INT	PK, NN
NomeCanale	Varchar(45)	
Tipo	ENUM(‘Pubblico’,’Privato’)	
Tabella Appartenenza		
Colonna	**Tipo di dato**	**Attributi**
CanaleDiComunicazione_Codice	INT	PK, NN
CanaleDiComunicazione_Progetto_IDProgetto	INT	PK, NN
Lavoratore_CF	Varchar(45)	PK, NN
Tabella Assegnazione		
Colonna	**Tipo di dato**	**Attributi**
Progetto_IDProgetto	INT	PK, NN
Lavoratore_CF	Varchar(45)	PK, NN
Tabella Coordinazione		
Colonna	**Tipo di dato**	**Attributi**
Progetto_IDProgetto	INT	PK, NN
Lavoratore_CF	Varchar(45)	PK, NN
Tabella Creazione		
Colonna	**Tipo di dato**	**Attributi**
CanaleDiComunicazione_Cod	INT	PK, NN, UQ

ice		
CanaleDiComunicazione_Progetto_IDProgetto	INT	PK, NN, UQ
Lavoratore_CF	Varchar(45)	NN
Tabella Messaggio		
Colonna	**Tipo di dato**	**Attributi**
Lavoratore_CF	Varchar(45)	PK, NN
DataInvio	Date	PK, NN
OrarioInvio	Time	PK, NN
Testo	Varchar(700)	NN
CanaleDiComunicazione_Codice	INT	NN
CanaleDiComunicazione_Progetto_IDProgetto	INT	NN
Tabella Risposta		
Colonna	**Tipo di dato**	**Attributi**
Messaggio_Lavoratore_CF	Varchar(45)	PK, NN
Messaggio_DataInvio	Date	PK, NN
Messaggio_OrarioInvio	Time	PK, NN
Messaggio_Lavoratore_CFD	Varchar(45)	NN
Messaggio_DataInvioD	Date	NN
Messaggio_OrarioInvioD	Time	NN
Tipo	ENUM('Pubblica','Privata')	NN

Indici

Non sono previsti indici oltre le chiavi primarie data l'assenza di query che ne avrebbero tratto un particolare beneficio.

Tabella Utente	
Indice <Primary>	**Tipo[3]:**
U_CF	<PR>

[3] IDX = index, UQ = unique, FT = full text, PR = primary.

Tabella Lavoratore	
Indice <Primary>	**Tipo:**
CF	<PR>
Tabella Progetto	
Indice <Primary>	**Tipo:**
IDProgetto	<PR>
Tabella CanaleDiComunicazione	
Indice <Primary>	**Tipo:**
Codice	<PR>
IDProgetto	<PR>
Tabella Messaggio	
Indice <Primary>	**Tipo:**
Lavoratore_CF	<PR>
DataInvio	<PR>
OrarioInvio	<PR>
Tabella Risposta	
Indice <Primary>	**Tipo:**
Messaggio_Lavoratore_CF	<PR>
Messaggio_DataInvio	<PR>
Messaggio_OrarioInvio	<PR>
Tabella Assegnazione	
Indice <Primary>	**Tipo:**
Lavoratore_CF	<PR>
Progetto_IDProgetto	<PR>
Tabella Appartenenza	
Indice <Primary>	**Tipo:**
CanaleDiComunicazione_Codice	<PR>
CanaleDiComunicazione_Progetto_IDProgetto	<PR>
Lavoratore_CF	<PR>
Tabella Coordinazione	
Indice <Primary>	**Tipo:**

Lavoratore_CF	<PR>
Progetto_IDProgetto	<PR>
Tabella Creazione	
Indice <Primary>	**Tipo:**
CanaleDiComunicazione_Codice	<PR>
CanaleDiComunicazione_Progetto_IDProgetto	<PR>
Lavoratore_CF	<PR>

Trigger

```
DELIMITER ;
USE `Azienda`;
DELIMITER $$
USE `Azienda`$$
DROP TRIGGER IF EXISTS `Azienda`.`Progetto_BEFORE_UPDATE` $$
USE `Azienda`$$
CREATE DEFINER = CURRENT_USER TRIGGER `Azienda`.`Progetto_BEFORE_UPDATE`
BEFORE UPDATE ON `Progetto` FOR EACH ROW
BEGIN
        declare var_datainizio date;
   -- selezione data inizio progetto
   select DataInizio from Progetto where Progetto.IDProgetto = new.IDProgetto into var_datainizio;
   -- controllo data
   if var_datainizio < new.DataFine then
                signal sqlstate '45002' set message_text = "Il termine non può precedere la data di
inizio di un progetto";
        end if;
END$$
```

Trigger che implementa la regola aziendale relativa alla data di chiusura di un progetto.

```
USE `Azienda`$$
DROP TRIGGER IF EXISTS `Azienda`.`CanaleDiComunicazione_BEFORE_INSERT` $$
```

```
USE `Azienda`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`Azienda`.`CanaleDiComunicazione_BEFORE_INSERT` BEFORE INSERT ON
`CanaleDiComunicazione` FOR EACH ROW
BEGIN
        declare var_datafine date;
   -- selezione data fine progetto
   select DataFine from Progetto where Progetto.IDProgetto = new.Progetto_IDProgetto into
var_datafine;
   -- controllo data
   if var_datafine is not null then
                signal sqlstate '45003' set message_text = "Impossibile creare un canale di
comunicazione afferente ad un progetto chiuso";
        end if;
END$$
```

Trigger che regolamenta la creazione di canali relativamente a progetti terminati. Anche se non previsto dalla richiesta, è stato implementato in modo tale da evitare la creazione di canali in contesti obsoleti.

```
USE `Azienda`$$
DROP TRIGGER IF EXISTS `Azienda`.`Messaggio_BEFORE_INSERT` $$
USE `Azienda`$$
CREATE DEFINER = CURRENT_USER TRIGGER `Azienda`.`Messaggio_BEFORE_INSERT`
BEFORE INSERT ON `Messaggio` FOR EACH ROW
BEGIN
        declare var_datafine date;
   -- selezione data fine progetto
   select DataFine from Progetto
   where Progetto.IDProgetto = new.CanaleDiComunicazione_Progetto_IDProgetto into
var_datafine;
   -- controllo data
   if var_datafine is not null then
                signal sqlstate '45004' set message_text = "Impossibile inviare un messaggio in un
```

```
canale di comunicazione afferente ad un progetto chiuso";
        end if;
END$$
```

Trigger che regolamenta l'invio di messaggi in canali afferenti a progetti terminati. Anche se non previsto dalla richiesta, è stato implementato in modo tale da evitare la crescita a dismisura del volume di dati causato da un uso improprio dell'applicativo.

```
USE `Azienda`$$
DROP TRIGGER IF EXISTS `Azienda`.`Messaggio_BEFORE_INSERT_Ruolo` $$
USE `Azienda`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`Azienda`.`Messaggio_BEFORE_INSERT_Ruolo` BEFORE INSERT ON `Messaggio` FOR
EACH ROW
BEGIN
        declare var_ruolo tinyint;
        select Ruolo from Lavoratore
   where Lavoratore.CF = new.Lavoratore_CF into var_ruolo;
   if var_ruolo = 1 then
                signal sqlstate '45006' set message_text = "Impossibile inviare un messaggio se il
ruolo è di Amministratore";
        end if;
END$$
```

Trigger che implementa la regola aziendale che impedisce ad un amministratore di scrivere messaggi.

```
USE `Azienda`$$
DROP TRIGGER IF EXISTS `Azienda`.`Appartenenza_BEFORE_INSERT` $$
USE `Azienda`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`Azienda`.`Appartenenza_BEFORE_INSERT` BEFORE INSERT ON `Appartenenza` FOR EACH
ROW
```

```
BEGIN
        declare var_tipocanale ENUM('Pubblico', 'Privato');
   declare var_numeropartecipanti INT;
   -- identificazione tipologia canale
   select tipo from CanaleDiComunicazione
   where CanaleDiComunicazione.Codice = new.CanaleDiComunicazione_Codice and
CanaleDiComunicazione.Progetto_IDProgetto =
new.CanaleDiComunicazione_Progetto_IDProgetto
   into var_tipocanale;
   -- identificazione numero partecipanti
   select count(*) from Appartenenza where Appartenenza.CanaleDiComunicazione_Codice =
new.CanaleDiComunicazione_Codice and
Appartenenza.CanaleDiComunicazione_Progetto_IDProgetto =
new.CanaleDiComunicazione_Progetto_IDProgetto
   into var_numeropartecipanti;
   -- verifica tipologia canale & possibilità di inserimento
   if var_tipocanale = 'Privato' and var_numeropartecipanti > 2 then
                signal sqlstate '45001' set message_text = "Impossibile aggiungere un'ulteriore
persona ad un canale privato";
   end if;
END$$
```

Trigger che regola parte della regola aziendale relativamente al numero di partecipanti di un canale di tipo “privato”.

```
USE `Azienda`$$
DROP TRIGGER IF EXISTS `Azienda`.`Appartenenza_BEFORE_INSERT_1` $$
USE `Azienda`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`Azienda`.`Appartenenza_BEFORE_INSERT_1` BEFORE INSERT ON `Appartenenza` FOR
EACH ROW
BEGIN
        declare var_datafine date;
   -- selezione data fine progetto
```

```
    select DataFine from Progetto
    where Progetto.IDProgetto = new.CanaleDiComunicazione_Progetto_IDProgetto into
var_datafine;
    -- controllo data
    if var_datafine is not null then
                signal sqlstate '45005' set message_text = "Impossibile aggiungere un partecipante in
un canale di comunicazione afferente ad un progetto chiuso";
        end if;
END$$
```

Trigger che regolamenta partecipazione a canali relativamente a progetti terminati. Anche se non previsto dalla richiesta, è stato implementato in modo tale da evitare un uso improprio.

```
USE `Azienda`$$
DROP TRIGGER IF EXISTS `Azienda`.`Creazione_BEFORE_INSERT` $$
USE `Azienda`$$
CREATE DEFINER = CURRENT_USER TRIGGER `Azienda`.`Creazione_BEFORE_INSERT`
BEFORE INSERT ON `Creazione` FOR EACH ROW
BEGIN
        declare var_datafine date;
    -- selezione data fine progetto
    select DataFine from Progetto join Messaggio on Progetto.IDProgetto =
Messaggio.CanaleDiComunicazione_Progetto_IDProgetto
    where Progetto.IDProgetto = new.CanaleDiComunicazione_Progetto_IDProgetto into
var_datafine;
    -- controllo data
    if var_datafine is not null then
                signal sqlstate '45004' set message_text = "Impossibile creare un canale di
comunicazione afferente ad un progetto chiuso";
        end if;
END$$
```

Trigger che regolamenta la creazione di canali relativamente a progetti terminati.

Anche se non previsto dalla richiesta, è stato implementato in modo tale da evitare la creazione di canali in contesti obsoleti.

Eventi

Non sono richiesti operazioni di manutenzione nel tempo, tuttavia sono stati impiegati alcuni accorgimenti nei trigger (diversi dagli eventi) per evitare che il numero dei dati crescesse a dismisura.

Viste

Non è previsto l'utilizzo di viste.

Stored Procedures e transazioni

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`insert_progetto`;
DELIMITER $$
USE `Azienda`$$
create procedure `insert_progetto`(in var_nomeprogetto varchar(45))
begin
declare exit handler for sqlexception
	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;
set transaction isolation level read uncommitted;
start transaction;
	insert into `Progetto` (`NomeProgetto`, `DataInizio`)
   values (var_nomeprogetto, date(now()));
commit;
end$$
```

insert_progetto: permette di inserire un nuovo progetto nel sistema.

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`insert_lavoratore`;
DELIMITER $$
USE `Azienda`$$
create procedure `insert_lavoratore`(in var_CF varchar(45), in var_nomelavoratore varchar(45), in
var_cognome varchar(45), in var_ruolo tinyint)
begin
declare exit handler for sqlexception
	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;
set transaction isolation level read uncommitted;
start transaction;
	insert into `Lavoratore` (`CF`, `NomeLavoratore`, `Cognome`, `Ruolo`)
   values (var_CF, var_nomelavoratore, var_cognome, var_ruolo);
commit;
end$$
```

insert_lavoratore: permette di inserire un nuovo lavoratore nel sistema.

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`insert_messaggio`;
DELIMITER $$
USE `Azienda`$$
create procedure `insert_messaggio`(in var_CF varchar(45), in var_testo varchar(700), in var_codice
INT, in var_IDProgetto INT)
begin
declare exit handler for sqlexception
	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;
```

```
set transaction isolation level serializable;
start transaction;
        insert into `Messaggio` (`Lavoratore_CF`, `DataInvio`, `OrarioInvio`, `Testo`,
`CanaleDiComunicazione_Codice`, `CanaleDiComunicazione_Progetto_IDProgetto`)
   values (var_CF, date(now()), time(now()), var_testo, var_codice, var_IDProgetto);
commit;
end$$
```

insert_messaggio: permette di inserire un nuovo messaggio in uno specifico canale di comunicazione.

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`creazione_canale`;

DELIMITER $$
USE `Azienda`$$
create procedure `creazione_canale`(in var_IDProgetto INT, in var_nomecanale varchar(45), in
var_CF varchar(45))
begin
declare exit handler for sqlexception
        begin
                rollback; -- rollback any changes made in the transaction
                resignal; -- raise again the sql exception to the caller
        end;
set transaction isolation level repeatable read;
start transaction;
   -- insert del canale di comunicazione non privato
        insert into `CanaleDiComunicazione` (`Progetto_IDProgetto`, `NomeCanale`, `Tipo`)
   values (var_IDProgetto, var_nomecanale, 'Pubblico');
   -- attribuzione della relazione di creazione al capoprogetto
   insert into `Creazione` (`CanaleDiComunicazione_Codice`,
`CanaleDiComunicazione_Progetto_IDProgetto`, `Lavoratore_CF`)
   values (last_insert_id(), var_IDProgetto, var_CF);
   -- inserimento capoprogetto nel canale
```

```
   insert into `Appartenenza` (`Lavoratore_CF`, `CanaleDiComunicazione_Codice`,
`CanaleDiComunicazione_Progetto_IDProgetto`)
   values(var_CF, last_insert_id(), var_IDProgetto);
commit;
end$$
```

creazione_canale: permette la creazione di un nuovo canale di comunicazione da parte di un capoprogetto, quest'ultimo verrà registrato anche come appartenente al canale appena creato.

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`assegnazione_progetto`;
DELIMITER $$
USE `Azienda`$$
create procedure `assegnazione_progetto`(in var_CF varchar(45), in var_IDProgetto INT)
begin
declare exit handler for sqlexception
        begin
                rollback; -- rollback any changes made in the transaction
                resignal; -- raise again the sql exception to the caller
        end;
set transaction isolation level repeatable read;
start transaction;
        insert into `Assegnazione` (`Lavoratore_CF`, `Progetto_IDProgetto`)
   values (var_CF, var_IDProgetto);
commit;
end$$
```

assegnazione_progetto: permette la registrazione della relazione di assegnazione progetto-lavoratore.

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`coordinazione_progetto`;
DELIMITER $$
USE `Azienda`$$
```

```
create procedure `coordinazione_progetto`(in var_CF varchar(45), in var_IDProgetto INT)
begin
declare exit handler for sqlexception
	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;
set transaction isolation level repeatable read;
start transaction;
	insert into `Coordinazione` (`Lavoratore_CF`, `Progetto_IDProgetto`)
   values (var_CF, var_IDProgetto);
commit;
end$$
```

coordinazione_progetto: permette la registrazione della relazione di coordinazione progetto-capoprogetto.

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`assegnazione_canale`;
DELIMITER $$
USE `Azienda`$$
create procedure `assegnazione_canale`(in var_CF varchar(45), in var_codice INT, in
var_IDProgetto INT)
begin
declare exit handler for sqlexception
	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;
set transaction isolation level repeatable read;
start transaction;
	insert into `Appartenenza` (`Lavoratore_CF`, `CanaleDiComunicazione_Codice`,
`CanaleDiComunicazione_Progetto_IDProgetto`)
 values (var_CF, var_codice, var_IDProgetto);
```

```
commit;
end$$
```

assegnazione_canale: permette al capoprogetto di far entrare un lavoratore in un canale di comunicazione.

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`risposta_pubblica`;
DELIMITER $$
USE `Azienda`$$
create procedure `risposta_pubblica`(in var_CFMittente varchar(45), in var_testo varchar(700), in var_CFDestinatario varchar(45), in var_datadestinatario date, in var_orariodestinatario time, in var_codice INT, in var_IDProgetto INT)
begin
declare var_datamittente date;
declare var_orariomittente time;
declare exit handler for sqlexception
	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;
set transaction isolation level serializable;
start transaction;
	set var_datamittente = date(now());
   set var_orariomittente = time(now());
	-- registrazione del messaggio
	insert into `Messaggio` (`Lavoratore_CF`, `DataInvio`, `OrarioInvio`, `Testo`, `CanaleDiComunicazione_Codice`, `CanaleDiComunicazione_Progetto_IDProgetto`)
   values (var_CFMittente, var_datamittente, var_orariomittente, var_testo, var_codice, var_IDProgetto);
   -- attribuzione della relazione di risposta
   insert into `Risposta` (`Messaggio_DataInvio`, `Messaggio_OrarioInvio`, `Messaggio_Lavoratore_CF`,`Messaggio_DataInvioD`, `Messaggio_OrarioInvioD`, `Messaggio_Lavoratore_CFD`, `Tipo`)
```

```
    values (var_datamittente, var_orariomittente, var_CFMittente, var_datadestinatario,
var_orariodestinatario, var_CFDestinatario, 'Pubblica');
commit;
end$$
```

risposta_pubblica: permette di inserire un messaggio in un canale (prima fase) e di riferirne un altro nello stesso canale (seconda fase); è stato scelto di differenziare la stored procedure da quella relativa ad una risposta privata in quanto presentano differenti operazioni.

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`risposta_privata`;
DELIMITER $$
USE `Azienda`$$
create procedure `risposta_privata`(in var_CFMittente varchar(45), in var_testo varchar(700), in
var_CFDestinatario varchar(45), in var_datadestinatario date, in var_orariodestinatario time, in
var_IDProgetto INT)
begin
declare var_datamittente date;
declare var_orariomittente time;
declare var_codice INT;
declare exit handler for sqlexception
        begin
                rollback; -- rollback any changes made in the transaction
                resignal; -- raise again the sql exception to the caller
        end;
set var_datamittente = date(now());
set var_orariomittente = time(now());
set transaction isolation level serializable;
start transaction;
                -- insert del canale di comunicazione privato
                insert into `CanaleDiComunicazione` (`Progetto_IDProgetto`, `NomeCanale`,
`Tipo`)
                values (var_IDProgetto, 'Canale Privato', 'Privato');
      -- assegnazione del canale ai due partecipanti
```

```
    insert into `Appartenenza` (`Lavoratore_CF`, `CanaleDiComunicazione_Codice`,
`CanaleDiComunicazione_Progetto_IDProgetto`)
            values (var_CFMittente, last_insert_id(), var_IDProgetto);
    insert into `Appartenenza` (`Lavoratore_CF`, `CanaleDiComunicazione_Codice`,
`CanaleDiComunicazione_Progetto_IDProgetto`)
            values (var_CFDestinatario, last_insert_id(), var_IDProgetto);
  -- procedura standard di inserimento di un messaggio
      -- registrazione del messaggio
      insert into `Messaggio` (`Lavoratore_CF`, `DataInvio`, `OrarioInvio`, `Testo`,
`CanaleDiComunicazione_Codice`, `CanaleDiComunicazione_Progetto_IDProgetto`)
  values (var_CFMittente, var_datamittente, var_orariomittente, var_testo, last_insert_id(),
var_IDProgetto);
  -- attribuzione della relazione di risposta
  insert into `Risposta` (`Tipo`,`Messaggio_Lavoratore_CF`,`Messaggio_DataInvio`,
`Messaggio_OrarioInvio`, `Messaggio_Lavoratore_CFD`,`Messaggio_DataInvioD`,
`Messaggio_OrarioInvioD`)
  values ('Privata',var_CFMittente, var_datamittente, var_orariomittente, var_CFDestinatario,
var_datadestinatario, var_orariodestinatario);
commit;
end$$
```

risposta_privata: permette di rispondere privatamente ad un messaggio; la procedure si articola in diverse fasi: (prima) creazione di un canale di comunicazione privato tra mittente e destinatario, (seconda) registrazione delle relazioni di appartenenza al canale, (terza) inserimento del messaggio nel canale di comunicazione appena creato, (quarta) attribuzione della relazione di risposta; è stato scelto di differenziare la stored procedure da quella relativa ad una risposta pubblica in quanto presentano differenti operazioni.

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`chiusura_progetto`;
DELIMITER $$
USE `Azienda`$$
create procedure `chiusura_progetto`(in var_IDProgetto INT, in var_datafine date)
begin
```

```
declare exit handler for sqlexception
	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;
set transaction isolation level repeatable read;
start transaction;
	update Progetto set DataFine = var_datafine where Progetto.IDProgetto = var_IDProgetto;
commit;
end$$
```

chiusura_progetto: permette di aggiornare il valore DataFine di un progetto esistente; anche se non previsto dalla richiesta, è stata implementata in modo tale da evitare una serie di operazioni obsolete che avrebbero potuto portare il volume dei dati a crescere a dismisura

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`login`;
DELIMITER $$
USE `Azienda`$$
create procedure `login`(in var_cf varchar(45), in var_pass varchar(45), out var_ruolo INT)
begin
	declare var_user_ruolo ENUM('Amministratore', 'Capoprogetto','Dipendente');
	select `Ruolo` from `Utente` where `U_CF` = var_cf AND `Password` = var_pass INTO
var_user_ruolo;
	if var_user_ruolo = 'Amministratore' then
		set var_ruolo = 1;
	elseif var_user_ruolo = 'Capoprogetto' then
		set var_ruolo = 2;
	elseif var_user_ruolo = 'Dipendente' then
		set var_ruolo = 3;
	else
		set var_ruolo = 4;
	end if;
END$$
```

login: permette di effettuare il login; restituisce il ruolo dell'utente come un valore di tipo INT

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`retrieve_appartenenza`;

DELIMITER $$
USE `Azienda`$$
create procedure `retrieve_appartenenza`(in var_CF varchar(45))
begin
declare var_ruolo tinyint;
declare exit handler for sqlexception
	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;
set transaction isolation level repeatable read;
start transaction;
   -- retrieve informazioni
	select Progetto.NomeProgetto as Progetto, Progetto.IDProgetto as ID_Progetto,
CanaleDiComunicazione.Codice as Codice_Canale, CanaleDiComunicazione.NomeCanale as
Nome_Canale
   from Progetto join Appartenenza on Progetto.IDProgetto =
Appartenenza.CanaleDiComunicazione_Progetto_IDProgetto join CanaleDiComunicazione on
Appartenenza.CanaleDiComunicazione_Progetto_IDProgetto =
CanaleDiComunicazione.Progetto_IDProgetto
   where Appartenenza.Lavoratore_CF = var_CF;
   -- secondo retireve per i progetti senza canale
   select Progetto.NomeProgetto as Progetto, Progetto.IDProgetto as ID_Progetto
   from Progetto join Appartenenza on Progetto.IDProgetto =
Appartenenza.CanaleDiComunicazione_Progetto_IDProgetto
   where Appartenenza.Lavoratore_CF = var_CF;
commit;
end$$
```

retrieve_appartenenza: procedura di supporto alle operazioni di stampa di progetti assegnati e canali di appartenenza relativamente ad un certo lavoratore.

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`retrieve_coordinazione`;
DELIMITER $$
USE `Azienda`$$
create procedure `retrieve_coordinazione`(in var_CF varchar(45))
begin
declare var_ruolo tinyint;
declare exit handler for sqlexception
	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;
set transaction isolation level repeatable read;
start transaction;
	select Ruolo from Lavoratore where Lavoratore.CF = var_CF into var_ruolo;
  -- controllo ruolo
  if var_ruolo <> 2 then
		signal sqlstate '45008' set message_text = "Operazione non concessa";
	end if;
  -- retrieve informazioni
	select Progetto.NomeProgetto as Progetto, Progetto.IDProgetto as ID_Progetto,
CanaleDiComunicazione.Codice as Codice_Canale, CanaleDiComunicazione.NomeCanale as
Nome_Canale
  from Progetto join Coordinazione on Progetto.IDProgetto = Coordinazione.Progetto_IDProgetto
join CanaleDiComunicazione on Coordinazione.Progetto_IDProgetto =
CanaleDiComunicazione.Progetto_IDProgetto
  where Coordinazione.Lavoratore_CF = var_CF;
  -- secondo retireve per i progetti senza canale
  select Progetto.NomeProgetto as Progetto, Progetto.IDProgetto as ID_Progetto
  from Progetto join Coordinazione on Progetto.IDProgetto = Coordinazione.Progetto_IDProgetto
```

```
   where Coordinazione.Lavoratore_CF = var_CF;
commit;
end$$
```

retrieve_coordinazione: procedura di supporto alle operazioni di stampa di progetti coordinati da un determinato capoprogetto e relativi canali di comunicazione.

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`retrieve_conversazioni`;
DELIMITER $$
USE `Azienda`$$
create procedure `retrieve_conversazioni`(in var_codice INT, in var_IDProgetto INT)
begin
declare exit handler for sqlexception
        begin
                rollback; -- rollback any changes made in the transaction
                resignal; -- raise again the sql exception to the caller
        end;
set transaction isolation level read committed;
start transaction;
   -- retrieve conversazioni
        select Lavoratore.NomeLavoratore as Nome_Lavoratore, Lavoratore.Cognome as
Cognome_Lavoratore, Messaggio.Lavoratore_CF as CF, Messaggio.DataInvio as Data_Invio,
Messaggio.OrarioInvio as Orario_Invio, Messaggio.Testo as Testo
   from CanaleDiComunicazione join Messaggio on (CanaleDiComunicazione.Codice =
Messaggio.CanaleDiComunicazione_Codice and CanaleDiComunicazione.Progetto_IDProgetto =
Messaggio.CanaleDiComunicazione_Progetto_IDProgetto) join Lavoratore on
Messaggio.Lavoratore_CF = Lavoratore.CF
   where CanaleDiComunicazione.Progetto_IDProgetto = var_IDProgetto and
CanaleDiComunicazione.Codice = var_codice
   order by Messaggio.DataInvio, Messaggio.OrarioInvio;
commit;
end$$
```

retrieve_conversazioni: procedura di supporto alle operazioni di stampa delle conversazioni

presenti in un determinato canale di comunicazione.

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`retrieve_progetti_canali`;
DELIMITER $$
USE `Azienda`$$
create procedure `retrieve_progetti_canali`()
begin
declare exit handler for sqlexception
	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;
set transaction isolation level repeatable read;
start transaction;
  -- retrieve informazioni
	select Progetto.NomeProgetto as Progetto, Progetto.IDProgetto as ID_Progetto,
CanaleDiComunicazione.Codice as Codice_Canale, CanaleDiComunicazione.NomeCanale as
Nome_Canale
  from Progetto join CanaleDiComunicazione on Progetto.IDProgetto =
CanaleDiComunicazione.Progetto_IDProgetto;
  select Progetto.NomeProgetto as Progetto, Progetto.IDProgetto as ID_Progetto from Progetto;
commit;
end$$
```

retrieve_progetti_canali: procedura di supporto alle operazioni di stampa di tutti i progetti e relativi canali di comunicazione presenti nel sistema.

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`insert_utente`;
DELIMITER $$
USE `Azienda`$$
create procedure `insert_utente`(in var_cf varchar(45), in var_password varchar(45), in var_ruolo
```

```
varchar(45))
begin
declare exit handler for sqlexception
	begin
		rollback; -- rollback any changes made in the transaction
		resignal; -- raise again the sql exception to the caller
	end;
set transaction isolation level read uncommitted;
start transaction;
	insert into `Utente` (`U_CF`, `Password`, `Ruolo`)
   values (var_CF, var_password, var_ruolo);
commit;
end$$
```

insert_utente: permette di inserire un nuovo utente nel sistema.

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`retrieve_appartenenza_coordinazione`;
DELIMITER $$
USE `Azienda`$$
create procedure `retrieve_appartenenza_coordinazione`(in var_CF varchar(45))
begin
declare var_ruolo tinyint;
declare exit handler for sqlexception
begin
rollback; -- rollback any changes made in the transaction
resignal; -- raise again the sql exception to the caller
end;
set transaction isolation level repeatable read;
start transaction;
select Ruolo from Lavoratore where Lavoratore.CF = var_CF into var_ruolo;
-- controllo ruolo
if var_ruolo <> 2 then
```

```
signal sqlstate '45009' set message_text = "Operazione non concessa";
end if;
-- retrieve informazioni
select Progetto.NomeProgetto as Progetto, Progetto.IDProgetto as ID_Progetto, Lavoratore.CF as
CF, Lavoratore.NomeLavoratore as Nome, Lavoratore.Cognome as Cognome
from Progetto join Assegnazione on Progetto.IDProgetto = Assegnazione.Progetto_IDProgetto join
Lavoratore on Assegnazione.Lavoratore_CF = Lavoratore.CF
where Progetto.IDProgetto in(
select Progetto.IDProgetto
from Progetto join Coordinazione on Progetto.IDProgetto = Coordinazione.Progetto_IDProgetto
where Coordinazione.Lavoratore_CF = var_CF
);
commit;
end$$
```

retrieve_appartenenza_coordinazione: procedura che permette ad un capoprogetto di stampare tutti i progetti da lui coordinati con relativi canali di comunicazione (annessi partecipanti).

```
USE `Azienda`;
DROP procedure IF EXISTS `Azienda`.`retrieve_dipendenti`;
DELIMITER $$
USE `Azienda`$$
create procedure `retrieve_dipendenti`()
begin
declare exit handler for sqlexception
begin
rollback; -- rollback any changes made in the transaction
resignal; -- raise again the sql exception to the caller
end;
set transaction isolation level read committed;
start transaction;
-- retrieve informazioni
select Lavoratore.CF as CF, Lavoratore.NomeLavoratore as Nome, Lavoratore.Cognome as
```

Cognome, Lavoratore.Ruolo as Ruolo from Lavoratore; commit; end$$
retrieve_dipendenti: procedura che permetti di stampare una lista contenente tutti i lavoratori presenti nel sistema.

Appendice: Implementazione

Codice SQL per instanziare il database

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;

SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;

SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';


-- -----------------------------------------------------

-- Schema Azienda

-- -----------------------------------------------------

DROP SCHEMA IF EXISTS `Azienda` ;


-- -----------------------------------------------------

-- Schema Azienda

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `Azienda` DEFAULT CHARACTER SET utf8 ;

USE `Azienda` ;


-- -----------------------------------------------------

-- Table `Azienda`.`Lavoratore`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `Azienda`.`Lavoratore` ;
```

```
CREATE TABLE IF NOT EXISTS `Azienda`.`Lavoratore` (
  `CF` VARCHAR(45) NOT NULL,
  `NomeLavoratore` VARCHAR(45) NOT NULL,
  `Cognome` VARCHAR(45) NOT NULL,
  `Ruolo` TINYINT NOT NULL,
  PRIMARY KEY (`CF`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `Azienda`.`Progetto`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Azienda`.`Progetto` ;

CREATE TABLE IF NOT EXISTS `Azienda`.`Progetto` (
  `IDProgetto` INT NOT NULL AUTO_INCREMENT,
  `NomeProgetto` VARCHAR(45) NULL,
  `DataInizio` DATE NOT NULL,
  `DataFine` DATE NULL,
  PRIMARY KEY (`IDProgetto`))
ENGINE = InnoDB;



-- -----------------------------------------------------
```

```
-- Table `Azienda`.`CanaleDiComunicazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Azienda`.`CanaleDiComunicazione` ;

CREATE TABLE IF NOT EXISTS `Azienda`.`CanaleDiComunicazione` (
  `Codice` INT NOT NULL AUTO_INCREMENT,
  `Progetto_IDProgetto` INT NOT NULL,
  `NomeCanale` VARCHAR(45) NULL,
  `Tipo` ENUM('Pubblico', 'Privato') NULL,
  PRIMARY KEY (`Codice`, `Progetto_IDProgetto`),
  CONSTRAINT `fk_CanaleDiComunicazione_Progetto1`
    FOREIGN KEY (`Progetto_IDProgetto`)
    REFERENCES `Azienda`.`Progetto` (`IDProgetto`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;

CREATE UNIQUE INDEX `Codice_UNIQUE` ON `Azienda`.`CanaleDiComunicazione` (`Codice`
ASC) VISIBLE;

CREATE INDEX `fk_CanaleDiComunicazione_Progetto1_idx` ON
`Azienda`.`CanaleDiComunicazione` (`Progetto_IDProgetto` ASC) VISIBLE;

-- -----------------------------------------------------
```

```
-- Table `Azienda`.`Messaggio`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Azienda`.`Messaggio` ;


CREATE TABLE IF NOT EXISTS `Azienda`.`Messaggio` (
  `Lavoratore_CF` VARCHAR(45) NOT NULL,
  `DataInvio` DATE NOT NULL,
  `OrarioInvio` TIME NOT NULL,
  `Testo` VARCHAR(700) NOT NULL,
  `CanaleDiComunicazione_Codice` INT NOT NULL,
  `CanaleDiComunicazione_Progetto_IDProgetto` INT NOT NULL,
  PRIMARY KEY (`Lavoratore_CF`, `DataInvio`, `OrarioInvio`),
  CONSTRAINT `fk_Messaggio_Lavoratore1`
    FOREIGN KEY (`Lavoratore_CF`)
    REFERENCES `Azienda`.`Lavoratore` (`CF`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Messaggio_CanaleDiComunicazione1`
    FOREIGN KEY (`CanaleDiComunicazione_Codice` ,
`CanaleDiComunicazione_Progetto_IDProgetto`)
    REFERENCES `Azienda`.`CanaleDiComunicazione` (`Codice` , `Progetto_IDProgetto`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
CREATE INDEX `fk_Messaggio_Lavoratore1_idx` ON `Azienda`.`Messaggio` (`Lavoratore_CF` ASC) VISIBLE;


CREATE INDEX `fk_Messaggio_CanaleDiComunicazione1_idx` ON `Azienda`.`Messaggio` (`CanaleDiComunicazione_Codice` ASC, `CanaleDiComunicazione_Progetto_IDProgetto` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `Azienda`.`Appartenenza`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Azienda`.`Appartenenza` ;

CREATE TABLE IF NOT EXISTS `Azienda`.`Appartenenza` (
  `Lavoratore_CF` VARCHAR(45) NOT NULL,
  `CanaleDiComunicazione_Codice` INT NOT NULL,
  `CanaleDiComunicazione_Progetto_IDProgetto` INT NOT NULL,
  PRIMARY KEY (`Lavoratore_CF`, `CanaleDiComunicazione_Codice`, `CanaleDiComunicazione_Progetto_IDProgetto`),
  CONSTRAINT `fk_Appartenenza_Lavoratore1`
    FOREIGN KEY (`Lavoratore_CF`)
    REFERENCES `Azienda`.`Lavoratore` (`CF`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Appartenenza_CanaleDiComunicazione1`
```

```
  FOREIGN KEY (`CanaleDiComunicazione_Codice` ,
`CanaleDiComunicazione_Progetto_IDProgetto`)
  REFERENCES `Azienda`.`CanaleDiComunicazione` (`Codice` , `Progetto_IDProgetto`)
  ON DELETE CASCADE
  ON UPDATE CASCADE)
ENGINE = InnoDB;


CREATE INDEX `fk_Appartenenza_CanaleDiComunicazione1_idx` ON `Azienda`.`Appartenenza`
(`CanaleDiComunicazione_Codice` ASC, `CanaleDiComunicazione_Progetto_IDProgetto` ASC)
VISIBLE;



-- -----------------------------------------------------
-- Table `Azienda`.`Assegnazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Azienda`.`Assegnazione` ;


CREATE TABLE IF NOT EXISTS `Azienda`.`Assegnazione` (
  `Lavoratore_CF` VARCHAR(45) NOT NULL,
  `Progetto_IDProgetto` INT NOT NULL,
  PRIMARY KEY (`Lavoratore_CF`, `Progetto_IDProgetto`),
  CONSTRAINT `fk_Assegnazione_Lavoratore1`
    FOREIGN KEY (`Lavoratore_CF`)
    REFERENCES `Azienda`.`Lavoratore` (`CF`)
    ON DELETE CASCADE
```

```
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Assegnazione_Progetto1`
    FOREIGN KEY (`Progetto_IDProgetto`)
    REFERENCES `Azienda`.`Progetto` (`IDProgetto`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


CREATE INDEX `fk_Assegnazione_Progetto1_idx` ON `Azienda`.`Assegnazione`
(`Progetto_IDProgetto` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `Azienda`.`Coordinazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Azienda`.`Coordinazione` ;


CREATE TABLE IF NOT EXISTS `Azienda`.`Coordinazione` (
  `Lavoratore_CF` VARCHAR(45) NOT NULL,
  `Progetto_IDProgetto` INT NOT NULL,
  PRIMARY KEY (`Lavoratore_CF`, `Progetto_IDProgetto`),
  CONSTRAINT `fk_Coordinazione_Lavoratore1`
    FOREIGN KEY (`Lavoratore_CF`)
    REFERENCES `Azienda`.`Lavoratore` (`CF`)
    ON DELETE CASCADE
```

```
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Coordinazione_Progetto1`
    FOREIGN KEY (`Progetto_IDProgetto`)
    REFERENCES `Azienda`.`Progetto` (`IDProgetto`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


CREATE INDEX `fk_Coordinazione_Progetto1_idx` ON `Azienda`.`Coordinazione`
(`Progetto_IDProgetto` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `Azienda`.`Creazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Azienda`.`Creazione` ;


CREATE TABLE IF NOT EXISTS `Azienda`.`Creazione` (
  `CanaleDiComunicazione_Codice` INT NOT NULL,
  `CanaleDiComunicazione_Progetto_IDProgetto` INT NOT NULL,
  `Lavoratore_CF` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`CanaleDiComunicazione_Codice`,
`CanaleDiComunicazione_Progetto_IDProgetto`),
  CONSTRAINT `fk_Creazione_CanaleDiComunicazione1`
```

```
    FOREIGN KEY (`CanaleDiComunicazione_Codice` ,
`CanaleDiComunicazione_Progetto_IDProgetto`)
    REFERENCES `Azienda`.`CanaleDiComunicazione` (`Codice` , `Progetto_IDProgetto`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Creazione_Lavoratore1`
    FOREIGN KEY (`Lavoratore_CF`)
    REFERENCES `Azienda`.`Lavoratore` (`CF`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


CREATE INDEX `fk_Creazione_Lavoratore1_idx` ON `Azienda`.`Creazione` (`Lavoratore_CF`
ASC) VISIBLE;


CREATE UNIQUE INDEX `CanaleDiComunicazione_Codice_UNIQUE` ON
`Azienda`.`Creazione` (`CanaleDiComunicazione_Codice` ASC) VISIBLE;


CREATE UNIQUE INDEX `CanaleDiComunicazione_Progetto_IDProgetto_UNIQUE` ON
`Azienda`.`Creazione` (`CanaleDiComunicazione_Progetto_IDProgetto` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `Azienda`.`Risposta`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Azienda`.`Risposta` ;
```

```
CREATE TABLE IF NOT EXISTS `Azienda`.`Risposta` (
  `Tipo` ENUM('Pubblica', 'Privata') NOT NULL,
  `Messaggio_Lavoratore_CF` VARCHAR(45) NOT NULL,
  `Messaggio_DataInvio` DATE NOT NULL,
  `Messaggio_OrarioInvio` TIME NOT NULL,
  `Messaggio_Lavoratore_CFD` VARCHAR(45) NOT NULL,
  `Messaggio_DataInvioD` DATE NOT NULL,
  `Messaggio_OrarioInvioD` TIME NOT NULL,
  PRIMARY KEY (`Messaggio_Lavoratore_CF`, `Messaggio_DataInvio`,
`Messaggio_OrarioInvio`),
  CONSTRAINT `fk_Risposta_Messaggio1`
    FOREIGN KEY (`Messaggio_Lavoratore_CF` , `Messaggio_DataInvio` ,
`Messaggio_OrarioInvio`)
    REFERENCES `Azienda`.`Messaggio` (`Lavoratore_CF` , `DataInvio` , `OrarioInvio`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Risposta_Messaggio2`
    FOREIGN KEY (`Messaggio_Lavoratore_CFD` , `Messaggio_DataInvioD` ,
`Messaggio_OrarioInvioD`)
    REFERENCES `Azienda`.`Messaggio` (`Lavoratore_CF` , `DataInvio` , `OrarioInvio`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
CREATE INDEX `fk_Risposta_Messaggio1_idx` ON `Azienda`.`Risposta`
(`Messaggio_Lavoratore_CF` ASC, `Messaggio_DataInvio` ASC, `Messaggio_OrarioInvio` ASC)
VISIBLE;


CREATE INDEX `fk_Risposta_Messaggio2_idx` ON `Azienda`.`Risposta`
(`Messaggio_Lavoratore_CFD` ASC, `Messaggio_DataInvioD` ASC, `Messaggio_OrarioInvioD`
ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `Azienda`.`Utente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Azienda`.`Utente` ;

CREATE TABLE IF NOT EXISTS `Azienda`.`Utente` (
  `U_CF` VARCHAR(45) NOT NULL,
  `Password` VARCHAR(45) NOT NULL,
  `Ruolo` ENUM('Amministratore', 'Capoprogetto', 'Dipendente') NOT NULL,
  PRIMARY KEY (`U_CF`))
ENGINE = InnoDB;

USE `Azienda` ;

DELIMITER ;
SET SQL_MODE = '';
DROP USER IF EXISTS Amministratore;
```

```
SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'Amministratore' IDENTIFIED BY '4mm1n1str4t0r3';


GRANT EXECUTE ON procedure `Azienda`.`chiusura_progetto` TO 'Amministratore';

GRANT EXECUTE ON procedure `Azienda`.`coordinazione_progetto` TO 'Amministratore';

GRANT EXECUTE ON procedure `Azienda`.`insert_lavoratore` TO 'Amministratore';

GRANT EXECUTE ON procedure `Azienda`.`insert_progetto` TO 'Amministratore';

GRANT EXECUTE ON procedure `Azienda`.`retrieve_conversazioni` TO 'Amministratore';

GRANT EXECUTE ON procedure `Azienda`.`retrieve_progetti_canali` TO 'Amministratore';

GRANT EXECUTE ON procedure `Azienda`.`insert_utente` TO 'Amministratore';

GRANT EXECUTE ON procedure `Azienda`.`retrieve_dipendenti` TO 'Amministratore';

SET SQL_MODE = '';

DROP USER IF EXISTS Capoprogetto;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'Capoprogetto' IDENTIFIED BY 'C4p0pr0g3tt0';


GRANT EXECUTE ON procedure `Azienda`.`retrieve_conversazioni` TO 'Capoprogetto';

GRANT EXECUTE ON procedure `Azienda`.`assegnazione_canale` TO 'Capoprogetto';

GRANT EXECUTE ON procedure `Azienda`.`assegnazione_progetto` TO 'Capoprogetto';

GRANT EXECUTE ON procedure `Azienda`.`creazione_canale` TO 'Capoprogetto';

GRANT EXECUTE ON procedure `Azienda`.`insert_messaggio` TO 'Capoprogetto';

GRANT EXECUTE ON procedure `Azienda`.`retrieve_appartenenza` TO 'Capoprogetto';
```

```
GRANT EXECUTE ON procedure `Azienda`.`retrieve_coordinazione` TO 'Capoprogetto';

GRANT EXECUTE ON procedure `Azienda`.`risposta_privata` TO 'Capoprogetto';

GRANT EXECUTE ON procedure `Azienda`.`risposta_pubblica` TO 'Capoprogetto';

GRANT EXECUTE ON procedure `Azienda`.`retrieve_appartenenza_coordinazione` TO
'Capoprogetto';

GRANT EXECUTE ON procedure `Azienda`.`retrieve_dipendenti` TO 'Capoprogetto';

GRANT EXECUTE ON procedure `Azienda`.`insert_messaggio` TO 'Capoprogetto';

GRANT EXECUTE ON procedure `Azienda`.`retrieve_appartenenza` TO 'Capoprogetto';

GRANT EXECUTE ON procedure `Azienda`.`retrieve_conversazioni` TO 'Capoprogetto';

GRANT EXECUTE ON procedure `Azienda`.`risposta_privata` TO 'Capoprogetto';

GRANT EXECUTE ON procedure `Azienda`.`risposta_pubblica` TO 'Capoprogetto';

SET SQL_MODE = '';

DROP USER IF EXISTS Dipendente;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'Dipendente' IDENTIFIED BY 'D1p3nd3nt3';


GRANT EXECUTE ON procedure `Azienda`.`insert_messaggio` TO 'Dipendente';

GRANT EXECUTE ON procedure `Azienda`.`retrieve_appartenenza` TO 'Dipendente';

GRANT EXECUTE ON procedure `Azienda`.`retrieve_conversazioni` TO 'Dipendente';

GRANT EXECUTE ON procedure `Azienda`.`risposta_privata` TO 'Dipendente';

GRANT EXECUTE ON procedure `Azienda`.`risposta_pubblica` TO 'Dipendente';

SET SQL_MODE = '';

DROP USER IF EXISTS Login;
```

```
SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'Login' IDENTIFIED BY 'L0g1n';


GRANT EXECUTE ON procedure `Azienda`.`login` TO 'Login';


SET SQL_MODE=@OLD_SQL_MODE;

SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;

SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;


-- ----------------------------------------------------

-- Data for table `Azienda`.`Lavoratore`

-- ----------------------------------------------------

START TRANSACTION;

USE `Azienda`;

INSERT INTO `Azienda`.`Lavoratore` (`CF`, `NomeLavoratore`, `Cognome`, `Ruolo`) VALUES
('JSSMNT90A41D810W', 'Jessie', 'Magenta', 3);

INSERT INTO `Azienda`.`Lavoratore` (`CF`, `NomeLavoratore`, `Cognome`, `Ruolo`) VALUES
('JMSBLU90A01D810M', 'James', 'Blu', 3);

INSERT INTO `Azienda`.`Lavoratore` (`CF`, `NomeLavoratore`, `Cognome`, `Ruolo`) VALUES
('MWTPMN90A01D810Q', 'Meowth', 'Pokemon', 2);

INSERT INTO `Azienda`.`Lavoratore` (`CF`, `NomeLavoratore`, `Cognome`, `Ruolo`) VALUES
('HRSLTN90A01D810A', 'Hershel', 'Layton', 2);

INSERT INTO `Azienda`.`Lavoratore` (`CF`, `NomeLavoratore`, `Cognome`, `Ruolo`) VALUES
('KTRLTN90A41D810N', 'Katrielle', 'Layton', 3);
```

```
INSERT INTO `Azienda`.`Lavoratore` (`CF`, `NomeLavoratore`, `Cognome`, `Ruolo`) VALUES ('LKUTTN90A01D810S', 'Luke', 'Triton', 3);

INSERT INTO `Azienda`.`Lavoratore` (`CF`, `NomeLavoratore`, `Cognome`, `Ruolo`) VALUES ('FLRRHL90A41D810X', 'Flora', 'Reinhold', 3);

INSERT INTO `Azienda`.`Lavoratore` (`CF`, `NomeLavoratore`, `Cognome`, `Ruolo`) VALUES ('CLVDVO90A01D810V', 'Clive', 'Dove', 3);

INSERT INTO `Azienda`.`Lavoratore` (`CF`, `NomeLavoratore`, `Cognome`, `Ruolo`) VALUES ('MMYLTV90A41D810U', 'Emmy', 'Altava', 3);


COMMIT;



-- -----------------------------------------------------

-- Data for table `Azienda`.`Progetto`

-- -----------------------------------------------------

START TRANSACTION;

USE `Azienda`;

INSERT INTO `Azienda`.`Progetto` (`IDProgetto`, `NomeProgetto`, `DataInizio`, `DataFine`) VALUES (345, 'PokemonGO', '2022/09/13', NULL);

INSERT INTO `Azienda`.`Progetto` (`IDProgetto`, `NomeProgetto`, `DataInizio`, `DataFine`) VALUES (346, 'Enigma001', '2022/09/14', NULL);


COMMIT;



-- -----------------------------------------------------
```

```
-- Data for table `Azienda`.`CanaleDiComunicazione`

-- -----------------------------------------------------

START TRANSACTION;

USE `Azienda`;

INSERT INTO `Azienda`.`CanaleDiComunicazione` (`Codice`, `Progetto_IDProgetto`,
`NomeCanale`, `Tipo`) VALUES (1, 345, 'Canale Principale', 'Pubblico');


COMMIT;



-- -----------------------------------------------------

-- Data for table `Azienda`.`Messaggio`

-- -----------------------------------------------------

START TRANSACTION;

USE `Azienda`;

INSERT INTO `Azienda`.`Messaggio` (`Lavoratore_CF`, `DataInvio`, `OrarioInvio`, `Testo`,
`CanaleDiComunicazione_Codice`, `CanaleDiComunicazione_Progetto_IDProgetto`) VALUES
('MWTPMN90A01D810Q', '2022/09/13', '14:06:00', 'Benvenuti nel canale principale!', 1, 345);

INSERT INTO `Azienda`.`Messaggio` (`Lavoratore_CF`, `DataInvio`, `OrarioInvio`, `Testo`,
`CanaleDiComunicazione_Codice`, `CanaleDiComunicazione_Progetto_IDProgetto`) VALUES
('MWTPMN90A01D810Q', '2022/09/14', '00:01:32', 'Sono lunico gatto a parlare', 1, 345);


COMMIT;



-- -----------------------------------------------------
```

```
-- Data for table `Azienda`.`Appartenenza`

-- -----------------------------------------------------

START TRANSACTION;

USE `Azienda`;

INSERT INTO `Azienda`.`Appartenenza` (`Lavoratore_CF`, `CanaleDiComunicazione_Codice`, `CanaleDiComunicazione_Progetto_IDProgetto`) VALUES ('MWTPMN90A01D810Q', 1, 345);

INSERT INTO `Azienda`.`Appartenenza` (`Lavoratore_CF`, `CanaleDiComunicazione_Codice`, `CanaleDiComunicazione_Progetto_IDProgetto`) VALUES ('JMSBLU90A01D810M', 1, 345);

INSERT INTO `Azienda`.`Appartenenza` (`Lavoratore_CF`, `CanaleDiComunicazione_Codice`, `CanaleDiComunicazione_Progetto_IDProgetto`) VALUES ('JSSMNT90A41D810W', 1, 345);


COMMIT;



-- -----------------------------------------------------

-- Data for table `Azienda`.`Assegnazione`

-- -----------------------------------------------------

START TRANSACTION;

USE `Azienda`;

INSERT INTO `Azienda`.`Assegnazione` (`Lavoratore_CF`, `Progetto_IDProgetto`) VALUES ('MWTPMN90A01D810Q', 345);

INSERT INTO `Azienda`.`Assegnazione` (`Lavoratore_CF`, `Progetto_IDProgetto`) VALUES ('JMSBLU90A01D810M', 345);

INSERT INTO `Azienda`.`Assegnazione` (`Lavoratore_CF`, `Progetto_IDProgetto`) VALUES ('JSSMNT90A41D810W', 345);
```

```
COMMIT;


-- -----------------------------------------------------
-- Data for table `Azienda`.`Coordinazione`
-- -----------------------------------------------------
START TRANSACTION;
USE `Azienda`;
INSERT INTO `Azienda`.`Coordinazione` (`Lavoratore_CF`, `Progetto_IDProgetto`) VALUES ('MWTPMN90A01D810Q', 345);

COMMIT;


-- -----------------------------------------------------
-- Data for table `Azienda`.`Utente`
-- -----------------------------------------------------
START TRANSACTION;
USE `Azienda`;
INSERT INTO `Azienda`.`Utente` (`U_CF`, `Password`, `Ruolo`) VALUES ('FRLMRT00A41D810Z', 'teamrocket', 'Amministratore');
INSERT INTO `Azienda`.`Utente` (`U_CF`, `Password`, `Ruolo`) VALUES ('JSSMNT90A41D810W', 'pikachu', 'Dipendente');
INSERT INTO `Azienda`.`Utente` (`U_CF`, `Password`, `Ruolo`) VALUES ('JMSBLU90A01D810M', 'pikachu', 'Dipendente');
```

```
INSERT INTO `Azienda`.`Utente` (`U_CF`, `Password`, `Ruolo`) VALUES
('MWTPMN90A01D810Q', 'giovanni', 'Capoprogetto');

INSERT INTO `Azienda`.`Utente` (`U_CF`, `Password`, `Ruolo`) VALUES
('HRSLTN90A01D810A', 'enigmi', 'Capoprogetto');

INSERT INTO `Azienda`.`Utente` (`U_CF`, `Password`, `Ruolo`) VALUES
('LKUTTN90A01D810S', 'loosha', 'Dipendente');

INSERT INTO `Azienda`.`Utente` (`U_CF`, `Password`, `Ruolo`) VALUES
('FLRRHL90A41D810X', 'fioccorosa', 'Dipendente');

INSERT INTO `Azienda`.`Utente` (`U_CF`, `Password`, `Ruolo`) VALUES
('MMYLTV90A41D810U', 'giallo', 'Dipendente');

INSERT INTO `Azienda`.`Utente` (`U_CF`, `Password`, `Ruolo`) VALUES
('CLVDVO90A01D810V', 'giornalismo', 'Dipendente');

INSERT INTO `Azienda`.`Utente` (`U_CF`, `Password`, `Ruolo`) VALUES
('KTRLTN90A41D810N', 'dolcetti', 'Dipendente');


COMMIT;
```

Codice del Front-End

defines.h

```
#pragma once

#include <stdbool.h>

#include <mysql/mysql.h>

struct configuration

{

   char *host;

   char *db_username;
```

```
    char *db_password;
    unsigned int port;
    char *database;
    char cf[45];
    char password[45];
};
extern struct configuration conf;
extern int parse_config(char *path, struct configuration *conf);
extern char *getInput(unsigned int lung, char *stringa, bool hide);
extern bool yesOrNo(char *domanda, char yes, char no, bool predef, bool insensitive);
extern char multiChoice(char *domanda, char choices[], int num);
extern bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn);
extern void run_as_amministratore(MYSQL *conn);
extern void run_as_capoprogetto(MYSQL *conn);
extern void run_as_dipendente(MYSQL *conn);
extern int parse_date(char *date, MYSQL_TIME *parsed);
extern char *parse_time(char *time);
extern MYSQL_RES *rs_metadata;
```

inout.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <ctype.h>
#include <termios.h>
#include <sys/ioctl.h>
#include <pthread.h>
#include <signal.h>
#include <stdbool.h>

#include "defines.h"

// Per la gestione dei segnali
static volatile sig_atomic_t signo;
typedef struct sigaction sigaction_t;
static void handler(int s);

char *getInput(unsigned int lung, char *stringa, bool hide)
{
   char c;
   unsigned int i;
   // Dichiara le variabili necessarie ad un possibile mascheramento dell'input
   sigaction_t sa, savealrm, saveint, savehup, savequit, saveterm;
   sigaction_t savetstp, savettin, savettou;
   struct termios term, oterm;
   if(hide) {
      // Svuota il buffer
```

```
(void) fflush(stdout);
// Cattura i segnali che altrimenti potrebbero far terminare il programma, lasciando l'utente
// senza output sulla shell
sigemptyset(&sa.sa_mask);
sa.sa_flags = 0; // Per non resettare le system call
sa.sa_handler = handler;
(void) sigaction(SIGALRM, &sa, &savealrm);
(void) sigaction(SIGINT, &sa, &saveint);
(void) sigaction(SIGHUP, &sa, &savehup);
(void) sigaction(SIGQUIT, &sa, &savequit);
(void) sigaction(SIGTERM, &sa, &saveterm);
(void) sigaction(SIGTSTP, &sa, &savetstp);
(void) sigaction(SIGTTIN, &sa, &savettin);
(void) sigaction(SIGTTOU, &sa, &savettou);
// Disattiva l'output su schermo
if (tcgetattr(fileno(stdin), &oterm) == 0) {
   (void) memcpy(&term, &oterm, sizeof(struct termios));
   term.c_lflag &= ~(ECHO|ECHONL);
   (void) tcsetattr(fileno(stdin), TCSAFLUSH, &term);
}
else
{
   (void) memset(&term, 0, sizeof(struct termios));
   (void) memset(&oterm, 0, sizeof(struct termios));
```

```
    }
}
// Acquisisce da tastiera al pi lung - 1 caratteri
for(i = 0; i < lung; i++) {
    (void) fread(&c, sizeof(char), 1, stdin);
    if(c == '\n') {
        stringa[i] = '\0';
        break;
    }
    else
    {
      stringa[i] = c;
    }

        // Gestisce gli asterischi
    if(hide)
    {
        if(c == '\b') // Backspace
            (void) write(fileno(stdout), &c, sizeof(char));
        else
            (void) write(fileno(stdout), "*", sizeof(char));
    }
}
// Controlla che il terminatore di stringa sia stato inserito
```

```
if(i == lung - 1)
    stringa[i] = '\0';

// Se sono stati digitati più caratteri, svuota il buffer della tastiera
if(strlen(stringa) >= lung)
{
    // Svuota il buffer della tastiera
    do {
        c = getchar();
    } while (c != '\n');
}
if(hide)
{
    //L'a capo dopo l'input
    (void) write(fileno(stdout), "\n", 1);
    // Ripristina le impostazioni precedenti dello schermo
    (void) tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);
    // Ripristina la gestione dei segnali
    (void) sigaction(SIGALRM, &savealrm, NULL);
    (void) sigaction(SIGINT, &saveint, NULL);
    (void) sigaction(SIGHUP, &savehup, NULL);
    (void) sigaction(SIGQUIT, &savequit, NULL);
    (void) sigaction(SIGTERM, &saveterm, NULL);
    (void) sigaction(SIGTSTP, &savetstp, NULL);
```

```
        (void) sigaction(SIGTTIN, &savettin, NULL);
        (void) sigaction(SIGTTOU, &savettou, NULL);
        // Se era stato ricevuto un segnale viene rilanciato al processo stesso
        if(signo)
            (void) raise(signo);
    }
    return stringa;
}



    // Per la gestione dei segnali
static void handler(int s)
{
    signo = s;
}

bool yesOrNo(char *domanda, char yes, char no, bool predef, bool insensitive)
{
    // I caratteri 'yes' e 'no' devono essere minuscoli
    yes = tolower(yes);
    no = tolower(no);
    // Decide quale delle due lettere mostrare come predefinite
    char s, n;
    if(predef) {
```

```
    s = toupper(yes);
    n = no;
}
else
{
    s = yes;
    n = toupper(no);
}
// Richiesta della risposta
while(true) {
    // Mostra la domanda
    printf("%s [%c/%c]: ", domanda, s, n);
    char c;
    getInput(1, &c, false);
    // Controlla quale risposta è stata data
    if(c == '\0') { // getInput() non può restituire '\n'!
        return predef;
    }
    else if(c == yes)
    {
        return true;
    } else if(c == no)
    {
        return false;
```

```
        } else if(c == toupper(yes))
        {
            if(predef || insensitive) return true;
        }
        else if(c == toupper(yes))
        {
            if(!predef || insensitive) return false;
        }
    }
}

char multiChoice(char *domanda, char choices[], int num)
{
    // Genera la stringa delle possibilità
    char *possib = malloc(2 * num * sizeof(char));
    int i, j = 0;
    for(i = 0; i < num; i++) {
        possib[j++] = choices[i];
        possib[j++] = '/';
    }
    possib[j-1] = '\0'; // Per eliminare l'ultima '/'
    // Chiede la risposta
    while(true) {
        // Mostra la domanda
```

```
        printf("%s [%s]: ", domanda, possib);
        char c;
        getInput(1, &c, false);
        // Controlla se un carattere valido
        for(i = 0; i < num; i++) {
            if(c == choices[i])
            return c;
            }
    }
}
```

parse.c

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"

#define BUFF_SIZE 4096
MYSQL_RES *rs_metadata;
// The final config struct will point into this

static char config[BUFF_SIZE];
```

```
/**
* JSON type identifier. Basic types are:
* o Object
* o Array
* o String
* o Other primitive: number, boolean (true/false) or null
*/

typedef enum {
    JSMN_UNDEFINED = 0,
    JSMN_OBJECT = 1,
    JSMN_ARRAY = 2,
    JSMN_STRING = 3,
    JSMN_PRIMITIVE = 4
} jsmntype_t;

enum jsmnerr {
    /* Not enough tokens were provided */
    JSMN_ERROR_NOMEM = -1,
    /* Invalid character inside JSON string */
    JSMN_ERROR_INVAL = -2,
    /* The string is not a full JSON packet, more bytes expected */
    JSMN_ERROR_PART = -3
};
```

```
/**
* JSON token description.
* type type (object, array, string etc.)
* start start position in JSON data string
* end
end position in JSON data string
*/
typedef struct {
    jsmntype_t type;
    int start;
    int end;
    int size;
} jsmntok_t;



/**
* JSON parser. Contains an array of token blocks available. Also stores
* the string being parsed now and current position in that string
*/
typedef struct {
    unsigned int pos; /* offset in the JSON string */
    unsigned int toknext; /* next token to allocate */
    int toksuper; /* superior token node, e.g parent object or array */
```

```
} jsmn_parser;


/**
* Allocates a fresh unused token from the token pool.
*/
static jsmntok_t *jsmn_alloc_token(jsmn_parser *parser, jsmntok_t *tokens, size_t num_tokens)
{
    jsmntok_t *tok;
    if (parser->toknext >= num_tokens)
    {
        return NULL;
    }
    tok = &tokens[parser->toknext++];
    tok->start = tok->end = -1;
    tok->size = 0;
    return tok;
}


/**
* Fills token type and boundaries.
*/
static void jsmn_fill_token(jsmntok_t *token, jsmntype_t type,
int start, int end)
{
```

```
    token->type = type;
    token->start = start;
    token->end = end;
    token->size = 0;
}

/**
* Fills next available token with JSON primitive.
*/
static int jsmn_parse_primitive(jsmn_parser *parser, const char *js, size_t len, jsmntok_t *tokens, size_t num_tokens)
{
    jsmntok_t *token;
    int start;
    start = parser->pos;
    for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
        switch (js[parser->pos]) {
            /* In strict mode primitive must be followed by "," or "}" or "]" */
            case ':':
            case '\t' : case '\r' : case '\n' : case ' ' :
            case ',' : case ']' : case '}' :
            goto found;
            }
        if (js[parser->pos] < 32 || js[parser->pos] >= 127) {
            parser->pos = start;
```

```
            return JSMN_ERROR_INVAL;
        }
    }
    found:
        if (tokens == NULL) {
            parser->pos--;
            return 0;
        }
        token = jsmn_alloc_token(parser, tokens, num_tokens);
        if (token == NULL) {
            parser->pos = start;
            return JSMN_ERROR_NOMEM;
        }
        jsmn_fill_token(token, JSMN_PRIMITIVE, start, parser->pos);
        parser->pos--;
        return 0;
}

/**
* Fills next token with JSON string.
*/

static int jsmn_parse_string(jsmn_parser *parser, const char *js,size_t len, jsmntok_t *tokens, size_t
num_tokens)
{
```

```
jsmntok_t *token;
int start = parser->pos;
parser->pos++;
/* Skip starting quote */
for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++)
{
    char c = js[parser->pos];
    /* Quote: end of string */
    if (c == '\"') {
        if (tokens == NULL) {
            return 0;
        }
        token = jsmn_alloc_token(parser, tokens, num_tokens);
        if (token == NULL) {
            parser->pos = start;
            return JSMN_ERROR_NOMEM;
        }
        jsmn_fill_token(token, JSMN_STRING, start+1, parser->pos);
        return 0;
    }
    /* Backslash: Quoted symbol expected */
    if (c == '\\' && parser->pos + 1 < len) {
        int i;
        parser->pos++;
```

```
switch (js[parser->pos])
{
    /* Allowed escaped symbols */
    case '\"': case '/' : case '\\' : case 'b' :
    case 'f' : case 'r' : case 'n' : case 't' :
        break;
    /* Allows escaped symbol \uXXXX */
    case 'u':
        parser->pos++;
        for(i = 0; i < 4 && parser->pos < len && js[parser->pos] != '\0'; i++) {
            /* If it isn't a hex character we have an error */
            if(!((js[parser->pos] >= 48 && js[parser->pos] <= 57) || /* 0-9 */
            (js[parser->pos] >= 65 && js[parser->pos] <= 70) || /* A-F */
            (js[parser->pos] >= 97 && js[parser->pos] <= 102))) { /* a-f */
                parser->pos = start;
                return JSMN_ERROR_INVAL;
            }
            parser->pos++;
        }
        parser->pos--;
        break;
    /* Unexpected symbol */
    default:
        parser->pos = start;
```

```
                return JSMN_ERROR_INVAL;
            }
        }
    }
    parser->pos = start;
    return JSMN_ERROR_PART;
}
```

```
/**
* Parse JSON string and fill tokens.
*/
static int jsmn_parse(jsmn_parser *parser, const char *js, size_t len, jsmntok_t *tokens, unsigned int num_tokens)
{
    int r;
    int i;
    jsmntok_t *token;
    int count = parser->toknext;
    for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
        char c;
        jsmntype_t type;
        c = js[parser->pos];
        switch (c) {
```

```
case '{': case '[':
    count++;
    if (tokens == NULL)
    {
        break;
    }
    token = jsmn_alloc_token(parser, tokens, num_tokens);
    if (token == NULL)
    return JSMN_ERROR_NOMEM;
    if (parser->toksuper != -1)
    {
        tokens[parser->toksuper].size++;
    }
    token->type = (c == '{' ? JSMN_OBJECT : JSMN_ARRAY);
    token->start = parser->pos;
    parser->toksuper = parser->toknext - 1;
    break;
case '}': case ']':
    if (tokens == NULL)
        break;
    type = (c == '}' ? JSMN_OBJECT : JSMN_ARRAY);
    for (i = parser->toknext - 1; i >= 0; i--)
    {
        token = &tokens[i];
```

```
        if (token->start != -1 && token->end == -1)
        {
            if (token->type != type)
            {
                return JSMN_ERROR_INVAL;
            }
            parser->toksuper = -1;
            token->end = parser->pos + 1;
            break;
        }
    }
    /* Error if unmatched closing bracket */
    if (i == -1) return JSMN_ERROR_INVAL;
    for (; i >= 0; i--)
    {
        token = &tokens[i];
        if (token->start != -1 && token->end == -1)
        {
            parser->toksuper = i;
            break;
        }
    }
    break;
case '\"':
```

```
    r = jsmn_parse_string(parser, js, len, tokens, num_tokens);
    if (r < 0) return r;
        count++;
    if (parser->toksuper != -1 && tokens != NULL)
        tokens[parser->toksuper].size++;
    break;
case '\t' : case '\r' : case '\n' : case ' ':
    break;
case ':':
    parser->toksuper = parser->toknext - 1;
    break;
case ',':
    if (tokens != NULL && parser->toksuper != -1 &&
    tokens[parser->toksuper].type != JSMN_ARRAY &&
    tokens[parser->toksuper].type != JSMN_OBJECT)
    {
        for (i = parser->toknext - 1; i >= 0; i--)
        {
            if (tokens[i].type == JSMN_ARRAY || tokens[i].type == JSMN_OBJECT)
            {
                if (tokens[i].start != -1 && tokens[i].end == -1)
                {
                    parser->toksuper = i;
                    break;
```

```
                }
            }
        }
    }
    break;
/* In non-strict mode every unquoted value is a primitive */
default:
    r = jsmn_parse_primitive(parser, js, len, tokens, num_tokens);
    if (r < 0) return r;
        count++;
    if (parser->toksuper != -1 && tokens != NULL)
        tokens[parser->toksuper].size++;
    break;
  }
}
if (tokens != NULL)
{
  for (i = parser->toknext - 1; i >= 0; i--)
  {
    /* Unmatched opened object or array */
    if (tokens[i].start != -1 && tokens[i].end == -1)
    {
        return JSMN_ERROR_PART;
    }
```

```
        }
    }
    return count;
}



/**
* Creates a new parser based over a given buffer with an array of tokens
* available.
*/
static void jsmn_init(jsmn_parser *parser)
{
    parser->pos = 0;
    parser->toknext = 0;
    parser->toksuper = -1;
}


static int jsoneq(const char *json, jsmntok_t *tok, const char *s)
{
    if (tok->type == JSMN_STRING
    && (int) strlen(s) == tok->end - tok->start
    && strncmp(json + tok->start, s, tok->end - tok->start) == 0)
    {
        return 0;
```

```
    }
    return -1;
}

static size_t load_file(char *filename)
{
    FILE *f = fopen(filename, "rb");
    if(f == NULL)
    {
        fprintf(stderr, "Unable to open file %s\n", filename);
        exit(1);
    }
    fseek(f, 0, SEEK_END);
    size_t fsize = ftell(f);
    fseek(f, 0, SEEK_SET); //same as rewind(f);
    if(fsize >= BUFF_SIZE)
    {
        fprintf(stderr, "Configuration file too large\n");
        abort();
    }
    fread(config, fsize, 1, f);
    fclose(f);
    config[fsize] = 0;
    return fsize;
```

```
}


int parse_config(char *path, struct configuration *conf)
{
    int i;
    int r;
    jsmn_parser p;
    jsmntok_t t[128]; /* We expect no more than 128 tokens */
    load_file(path);
    jsmn_init(&p);
    r = jsmn_parse(&p, config, strlen(config), t, sizeof(t)/sizeof(t[0]));
    if (r < 0)
    {
        printf("Failed to parse JSON: %d\n", r);
        return 0;
    }
    /* Assume the top-level element is an object */
    if (r < 1 || t[0].type != JSMN_OBJECT)
    {
        printf("Object expected\n");
        return 0;
    }
    /* Loop over all keys of the root object */
    for (i = 1; i < r; i++) {
```

```
if (jsoneq(config, &t[i], "host") == 0)
{
    /* We may use strndup() to fetch string value */
    conf->host = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
    i++;
}
else if (jsoneq(config, &t[i], "username") == 0)
{
    conf->db_username = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
    i++;
}
else if (jsoneq(config, &t[i], "password") == 0)
{
    conf->db_password = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
    i++;
}
else if (jsoneq(config, &t[i], "port") == 0)
{
    conf->port = strtol(config + t[i+1].start, NULL, 10);
    i++;
}
else if (jsoneq(config, &t[i], "database") == 0)
{
    conf->database = strndup(config + t[i+1].start, t[i+1].end-t[i+1].start);
```

```
            i++;
        }
        else
        {
            printf("Unexpected key: %.*s\n", t[i].end-t[i].start, config + t[i].start);
        }
    }
    return 1;
}


int parse_date(char *date, MYSQL_TIME *parsed)
{
    char d[3];
    char m[3];
    char y[5];
    memcpy(y, date, 4);
    memcpy(m, date+5, 2);
    memcpy(d, date+8, 2);
    y[4]= '\0';
    m[2] = '\0';
    d[2] = '\0';
    parsed->year = atoi(y);
    parsed->month = atoi(m);
```

```
    parsed->day = atoi(d);
    return 0;
}

char *parse_time(char *time)
{
    char *parsed = malloc(10);
    memcpy(parsed, time, 5);
    memcpy(parsed+5, ":", 1);
    memcpy(parsed+6, "00\0", 3);
    return parsed;
}
```

utils.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "defines.h"
#include "utils.h"

void print_stmt_error (MYSQL_STMT *stmt, char *message)
{
    fprintf (stderr, "%s\n", message);
```

```
   if (stmt != NULL) {
      fprintf (stderr, "Error %u (%s): %s\n", mysql_stmt_errno (stmt), mysql_stmt_sqlstate(stmt), mysql_stmt_error (stmt));
   }
}

void print_error(MYSQL *conn, char *message)
{
   fprintf (stderr, "%s\n", message);
   if (conn != NULL) {
      #if MYSQL_VERSION_ID >= 40101
      fprintf (stderr, "Error %u (%s): %s\n",
      mysql_errno (conn), mysql_sqlstate(conn), mysql_error (conn));
      #else
      fprintf (stderr, "Error %u: %s\n",
      mysql_errno (conn), mysql_error (conn));
      #endif
   }
}

bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn)
{
   bool update_length = true;
   *stmt = mysql_stmt_init(conn);
   if (*stmt == NULL)
```

```
    {
        print_error(conn, "Could not initialize statement handler");
        return false;
    }
    if (mysql_stmt_prepare (*stmt, statement, strlen(statement)) != 0) {
        print_stmt_error(*stmt, "Could not prepare statement");
        return false;
    }
    mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH, &update_length);
    return true;
}

void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message, bool
close_stmt)
{
    print_stmt_error(stmt, message);
    if(close_stmt) mysql_stmt_close(stmt);
    mysql_close(conn);
    exit(EXIT_FAILURE);
}

static void print_dashes(MYSQL_RES *res_set)
{
    MYSQL_FIELD *field;
```

```
    unsigned int i, j;
    mysql_field_seek(res_set, 0);
    putchar('+');
    for (i = 0; i < mysql_num_fields(res_set); i++) {
        field = mysql_fetch_field(res_set);
        for (j = 0; j < field->max_length + 2; j++)
        putchar('-');
        putchar('+');
    }
    putchar('\n');
}

static void dump_result_set_header(MYSQL_RES *res_set)
{
    MYSQL_FIELD *field;
    unsigned long col_len;
    unsigned int i;
    /* determine column display widths -- requires result set to be */
    /* generated with mysql_store_result(), not mysql_use_result() */
    mysql_field_seek (res_set, 0);
    for (i = 0; i < mysql_num_fields (res_set); i++) {
        field = mysql_fetch_field (res_set);
        col_len = strlen(field->name);
        if (col_len < field->max_length)
```

```
            col_len = field->max_length;
        if (col_len < 4 && !IS_NOT_NULL(field->flags))
            col_len = 4; /* 4 = length of the word "NULL" */
        field->max_length = col_len; /* reset column info */
    }
    print_dashes(res_set);
    putchar('|');
    mysql_field_seek (res_set, 0);
    for (i = 0; i < mysql_num_fields(res_set); i++) {
        field = mysql_fetch_field(res_set);
        printf(" %-*s |", (int)field->max_length, field->name);
    }
    putchar('\n');
    print_dashes(res_set);
}
```

```
MYSQL_RES *dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title, int
*num_result)
{
    int i;
    int status;
    int num_fields;
    /* number of columns in result */
    MYSQL_FIELD *fields; /* for result set metadata */
    MYSQL_BIND *rs_bind; /* for output buffers */
```

```
MYSQL_RES *rs_metadata;
MYSQL_TIME *date;
size_t attr_size;
*num_result=-1;
/* Prefetch the whole result set. This in conjunction with
* STMT_ATTR_UPDATE_MAX_LENGTH set in `setup_prepared_stmt`
* updates the result set metadata which are fetched in this
* function, to allow to compute the actual max length of
* the columns.
*/
if (mysql_stmt_store_result(stmt)) {
    fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}
/* the column count is > 0 if there is a result set */
/* 0 if the result is only the final status packet */
num_fields = mysql_stmt_field_count(stmt);
if (num_fields > 0) {
    /* there is a result set to fetch */
    printf("%s\n", title);
    if((rs_metadata = mysql_stmt_result_metadata(stmt)) == NULL) {
        finish_with_stmt_error(conn, stmt, "Unable to retrieve result metadata\n", true);
    }
```

```
dump_result_set_header(rs_metadata);
fields = mysql_fetch_fields(rs_metadata);
rs_bind = (MYSQL_BIND *)malloc(sizeof (MYSQL_BIND) * num_fields);
if (!rs_bind) {
   finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n", true);
}
memset(rs_bind, 0, sizeof (MYSQL_BIND) * num_fields);
/* set up and bind result set output buffers */
for (i = 0; i < num_fields; ++i) {
   // Properly size the parameter buffer
   switch(fields[i].type) {
      case MYSQL_TYPE_DATE:
          attr_size = sizeof(MYSQL_TIME);
          break;
      case MYSQL_TYPE_TIMESTAMP:
      case MYSQL_TYPE_DATETIME:
      case MYSQL_TYPE_TIME:
         attr_size = sizeof(MYSQL_TIME);
         break;
      case MYSQL_TYPE_FLOAT:
         attr_size = sizeof(float);
         break;
      case MYSQL_TYPE_DOUBLE:
         attr_size = sizeof(double);
```

```
            break;
        case MYSQL_TYPE_TINY:
            attr_size = sizeof(signed char);
            break;
        case MYSQL_TYPE_SHORT:
        case MYSQL_TYPE_YEAR:
            attr_size = sizeof(short int);
            break;
        case MYSQL_TYPE_LONG:
        case MYSQL_TYPE_INT24:
            attr_size = sizeof(int);
            break;
        case MYSQL_TYPE_LONGLONG:
            attr_size = sizeof(int);
            break;
        default:
            attr_size = fields[i].max_length;
            break;
    }
    // Setup the binding for the current parameter
    rs_bind[i].buffer_type = fields[i].type;
    rs_bind[i].buffer = malloc(attr_size + 1);
    rs_bind[i].buffer_length = attr_size + 1;
    if(rs_bind[i].buffer == NULL)
```

```
    {
        finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n", true);
    }
}
if(mysql_stmt_bind_result(stmt, rs_bind)) {
    finish_with_stmt_error(conn, stmt, "Unable to bind output parameters\n", true);
}
/* fetch and display result set rows */
while (true) {
    *num_result = *num_result +1;
    status = mysql_stmt_fetch(stmt);

    if (status == 1 || status == MYSQL_NO_DATA)
    break;
    putchar('|');
    for (i = 0; i < num_fields; i++) {
        if (rs_bind[i].is_null_value) {
            printf (" %-*s |", (int)fields[i].max_length, "NULL");
            continue;
        }
        switch (rs_bind[i].buffer_type) {
            case MYSQL_TYPE_VAR_STRING:
            case MYSQL_TYPE_DATETIME:
                printf(" %-*s |", (int)fields[i].max_length, (char*)rs_bind[i].buffer);
```

```
    break;
case MYSQL_TYPE_DATE:
case MYSQL_TYPE_TIMESTAMP:
    date = (MYSQL_TIME *)rs_bind[i].buffer;
    printf(" %d-%02d-%02d |", date->year, date->month, date->day);
    break;
case MYSQL_TYPE_TIME:
    date = (MYSQL_TIME *)rs_bind[i].buffer;
    char *tempo = malloc(sizeof(date));
    sprintf(tempo, " %02d:%02d:%02d ", date->hour, date->minute, date->second);
    printf(" %-*s |",(int)fields[i].max_length, tempo);
    free(tempo);
   break;
case MYSQL_TYPE_STRING:
    printf(" %-*s |", (int)fields[i].max_length, (char *)rs_bind[i].buffer);
    break;
case MYSQL_TYPE_FLOAT:
case MYSQL_TYPE_DOUBLE:
    printf(" %.02f |", *(float *)rs_bind[i].buffer);
    break;
case MYSQL_TYPE_LONG:
case MYSQL_TYPE_SHORT:
case MYSQL_TYPE_TINY:
    printf(" %-*d |", (int)fields[i].max_length, *(int *)rs_bind[i].buffer);
```

```
                break;
            case MYSQL_TYPE_NEWDECIMAL:
                printf(" %-*.02lf |", (int)fields[i].max_length, *(float*) rs_bind[i].buffer);
                break;
            default:
                printf("ERROR: Unhandled type (%d)\n", rs_bind[i].buffer_type);
                abort();
          }
        }
        putchar('\n');
        print_dashes(rs_metadata);
      }

      /* free output buffers */
      for (i = 0; i < num_fields; i++) {
        free(rs_bind[i].buffer);
      }
      free(rs_bind);
   }
        return rs_metadata;
}

int date_compare(char *s)
{
```

```
    if(strlen(s)!=10) return 0;
    for(int i=0; i<10; i++)
    {
        if(i!= 4 && i!= 7 && (s[i]<48 || s[i]>57)) return 0;
    }
    if(s[4]!= '-' || s[7]!= '-') return 0;
    return 1;
}

int time_compare(char *s)
{
    if (strlen(s)!=8) return 0;
    for(int i=0; i<7; i++)
    {
        if(i!=2 && i!=5 && (s[i]<48 || s[i]>57)) return 0;
    }
    if(s[2]!= ':' && s[5] != ':') return 0;
    return 1;
}

int int_compare(char *s)
{
    for(unsigned long i=0;i<strlen(s);i++)
    {
```

```
        if(s[i]<48||s[i]>57) return 0;
    }
    return 1;
}
```

utils.h

```
#include <stdbool.h>
#include <mysql/mysql.h>


extern void print_error (MYSQL *conn, char *message);
extern void print_stmt_error (MYSQL_STMT *stmt, char *message);
extern void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message, bool close_stmt);
extern MYSQL_RES *dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title, int *num_result);
extern int date_compare(char *s);
extern int time_compare(char *s);
extern int int_compare(char *s);
```

amministratore.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "utils.h"
#include "defines.h"
```

```
// funzione da chiamare in caso di inserimento di un progetto
static void inserisci_progetto(MYSQL *conn)
{
   MYSQL_STMT *prepared_stmt;
   char nomeprogetto[45];
   char options[2] = {'1','2'};
   char op1;

   // retrieve del parametro necessario (titolo)
   printf("\nDigitare il nome da associare al progetto: ");
   getInput(45, nomeprogetto, false);


   MYSQL_BIND param[1];
   if(!setup_prepared_stmt(&prepared_stmt, "call insert_progetto( ? )", conn))
   {
      finish_with_stmt_error(conn, prepared_stmt, "Impossibile inserire il progetto\n", false);
   }
   // Preparazione parametri
   memset(param, 0, sizeof(param));
   param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
   param[0].buffer = nomeprogetto;
   param[0].buffer_length = strlen(nomeprogetto);
```

```
if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
{
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri per l'inserimento del progetto\n", true);
}
// Esecuzione procedura
if (mysql_stmt_execute(prepared_stmt) != 0)
{
    print_stmt_error (prepared_stmt, "Errore nell'inserimento del progetto.");
    mysql_stmt_close(prepared_stmt);
}
else
{
    printf("Progetto inserito correttamente...\n");
    mysql_stmt_close(prepared_stmt);



    printf("\nVuoi inserire un altro progetto? \n1) Sì \n2) No\n");
    op1 = multiChoice("Select: ", options, 2);
    switch(op1)
    {
        case '1':
            inserisci_progetto(conn);
            break;
        case '2':
```

```
            printf("\nArrivederci!");
            return;

        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
      }
   }
}
```

```
//funzione da chiamare in caso in cui si voglia affidare un progetto ad un capoprogetto
static void inserisci_coordinazione_progetto(MYSQL *conn)
{
   MYSQL_STMT *prepared_stmt;
   MYSQL_BIND param[2];
   char option[2] = {'1','2'};
   char op1;
   // Parametri necessari alla relazione di coordinazione
   char cf[45];
   char id_c[45];
   int id;
   int tratta;
```

```
    // Retrieve informazioni
    printf("\nInserire codice fiscale capoprogetto: ");
        getInput(45, cf, false);



    printf("Inserire ID del progetto: ");
    while(true)
    {
        getInput(45, id_c, false);
        if(int_compare(id_c))
        break;
        else printf("Formato errato, riprova: ");
    }
    id = atoi(id_c);
    // stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call coordinazione_progetto(?, ?)", conn))
    {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile assegnare coordinazione progetto\n",
false);
    }
    // sistemazione parametri
    memset(param, 0, sizeof(param));
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer = cf;
    param[0].buffer_length = strlen(cf);
```

```
param[1].buffer_type = MYSQL_TYPE_LONG;
param[1].buffer = &id;
param[1].buffer_length = sizeof(id);
if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
{
   finish_with_stmt_error(conn, prepared_stmt, "Non è stato possibile effettuare il bind dei parametri\n" , true);
}
// Esecuzione procedura
if (mysql_stmt_execute(prepared_stmt) != 0)
{
   print_stmt_error(prepared_stmt, "Errore nell'assegnazione del progetto al manager.");
   goto out;
}
printf("Coordinazione progetto assegnata correttamente...\n");
out:
{mysql_stmt_close(prepared_stmt);}

printf("\nVuoi gestire la coordinazione di un altro progetto? \n1) Sì \n2) No\n");
   op1 = multiChoice("Select: ", option, 2);
   switch(op1)
   {
      case '1':
         inserisci_coordinazione_progetto(conn);
         break;
```

```
            case '2':
                printf("\nArrivederci!");
                return;

            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
        }
}

static void inserisci_chiusura_progetto(MYSQL *conn)
{
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[2];
    // parametri necessari all'operazione di chiusura del progetto
    char id_c[45];
    int id;
    char data_c[11];
    char options[2] = {'1','2'};
    char op;
    MYSQL_TIME data;
    // retrieve informazioni
    printf("\nInserire ID del progetto che si vuole chiudere: ");
    getInput(45, id_c, false);
```

```
id = atoi(id_c);
printf("Data chiusura(yyyy-mm-dd): ");
while(true){
    getInput(11, data_c, false);
    if(date_compare(data_c)) break;
    else printf("Formato errato, riprova: ");
}

parse_date(data_c, &data);
// Prepare stored procedure call
if(!setup_prepared_stmt(&prepared_stmt, "call chiusura_progetto(?, ?)", conn)) {
    finish_with_stmt_error(conn, prepared_stmt, "Non è stato possibile effettuare la chiusura del progetto\n", false);
}
// sistemazione parametri
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG;
param[0].buffer = &id;
param[0].buffer_length = sizeof(id);
param[1].buffer_type = MYSQL_TYPE_DATE;
param[1].buffer = &data;
param[1].buffer_length = sizeof(MYSQL_TIME);
if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri\n", true);
}
```

```
    // Esecuzione procedura
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nella chiusura del progetto.");
    } else {
        printf("Progetto chiuso correttamente...\n");
    }
    mysql_stmt_close(prepared_stmt);
    printf("\nVuoi gestire la coordinazione di un altro progetto? \n1) Sì \n2) No\n");
        op = multiChoice("Select: ", options, 2);
        switch(op)
        {
            case '1':
                inserisci_chiusura_progetto(conn);
                break;
            case '2':
                printf("\nArrivederci!");
                return;

            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
        }
}
```

```
static void consultazione_canale(MYSQL *conn){
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[2];
    char header[512];
    int results;
    //parametri necessari
    char id_c[45];
    char cod_c[45];
    int id;
    int cod;
    int status;
    // Retrieve informazioni
    printf("Inserire ID del progetto: ");
    while(true)
    {
        getInput(45, id_c, false);
        if(int_compare(id_c))
        break;
        else printf("Formato errato, riprova: ");
    }
    id = atoi(id_c);
    printf("Inserire codice del canale: ");
    while(true)
    {
```

```
        getInput(45, cod_c, false);
        if(int_compare(cod_c))
        break;
        else printf("Formato errato, riprova: ");
    }
    cod = atoi(cod_c);
    // sistemazione parametri
    memset(param, 0, sizeof(param));
    param[0].buffer_type = MYSQL_TYPE_LONG;
    param[0].buffer = &cod;
    param[0].buffer_length = sizeof(cod);
    param[1].buffer_type = MYSQL_TYPE_LONG;
    param[1].buffer = &id;
    param[1].buffer_length = sizeof(id);
    // chiamata procedura
    if(!setup_prepared_stmt(&prepared_stmt, "call retrieve_conversazioni( ?, ? )", conn))
    {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile stampare le conversazioni\n", false);
    }

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri\n",
true);
    }
```

```
if (mysql_stmt_execute(prepared_stmt) != 0)
{
    print_stmt_error (prepared_stmt, "Errore nella stampa delle conversazioni.");
    mysql_stmt_close(prepared_stmt);
        return;
}


        // stampa conversazioni
    do {
    // Skip OUT variables (although they are not present in the procedure...)
    if(conn->server_status & SERVER_PS_OUT_PARAMS) {
        goto next;
    }
    dump_result_set(conn, prepared_stmt, header, &results);
    mysql_free_result(rs_metadata);
    next:
    status = mysql_stmt_next_result(prepared_stmt);
    if (status > 0){
        finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
    }
    if(results == 0) printf("Nessuna disponibilità\n");
} while (status == 0);
out:
```

```
    mysql_stmt_close(prepared_stmt);
        return;

}

// funzione in grado di stampare lista progetti + canali di comunicazione
static void stampa_progetti(MYSQL *conn)
{
    MYSQL_STMT *prepared_stmt;
    char op1;
    char header[512];
    char options[4] = {'1','2','3','4'};
    int status;
    int results;

    if(!setup_prepared_stmt(&prepared_stmt, "call retrieve_progetti_canali( )", conn))
    {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile stampare la lista dei progetti\n", false);
    }

    if (mysql_stmt_execute(prepared_stmt) != 0)
```

```
{
    print_stmt_error (prepared_stmt, "Errore nella stampa della lista dei progetti.");
    goto out;
}



    do {
    // Skip OUT variables (although they are not present in the procedure...)
    if(conn->server_status & SERVER_PS_OUT_PARAMS) {
        goto next;
    }
    dump_result_set(conn, prepared_stmt, header, &results);
    mysql_free_result(rs_metadata);
    next:
    status = mysql_stmt_next_result(prepared_stmt);
    if (status > 0){
        finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
    }
    if(results == 0) printf("Nessuna disponibilità\n");
} while (status == 0);
out:
mysql_stmt_close(prepared_stmt);
```

```
printf("*** Azioni disponibili: ***\n\n");
printf("1) Assegnare coordinazione progetto ad un capoprogetto\n");
printf("2) Chiusura di un progetto\n");
printf("3) Consultare un canale di comunicazione\n");
printf("4) Chiudere l'applicazione\n");
op1 = multiChoice("Select: ", options, 4);
switch(op1)
{
   case '1':
      inserisci_coordinazione_progetto(conn);
      break;
   case '2':
      inserisci_chiusura_progetto(conn);
      break;
   case '3':
      consultazione_canale(conn);
      break;
   case '4':
      printf("\nArrivederci!");
      return;

   default:
      fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
      abort();
```

```
        }

}


static void inserisci_lavoratore(MYSQL *conn)
{
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[4];
    char option[2] = {'1', '2'};
    char op;
    // Parametri necessari all'inserimento
    char cf[45];
    char nome[45];
    char cognome[45];
    signed char r;
    char ruolo;
    // Retrieve informazioni da input utente
    printf("\nCodice fiscale lavoratore: ");
    getInput(45, cf, false);
    printf("Nome lavoratore: ");
    getInput(45, nome, false);
    printf("Cognome lavoratore: ");
    getInput(45, cognome, false);
```

```
printf("Inserire un ruolo:\n1)Capoprogetto\n2)Dipendente\n");
ruolo=multiChoice("Select:", option, 2);
r = atoi(&ruolo);
// Preparazione stored procedure
if(!setup_prepared_stmt(&prepared_stmt, "call insert_lavoratore(?, ?, ?, ?)", conn))
{
   finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare l'inserimento di un Lavoratore\n", false);
}
// sistemazione parametri
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param[0].buffer = cf;
param[0].buffer_length = strlen(cf);
param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
param[1].buffer = nome;
param[1].buffer_length = strlen(nome);
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
param[2].buffer = cognome;
param[2].buffer_length = strlen(cognome);
param[3].buffer_type = MYSQL_TYPE_TINY;
param[3].buffer = &r;
param[3].buffer_length= sizeof(r);

if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
```

```
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri perl'inserimento di un lavoratore\n", true);
  }

  if (mysql_stmt_execute(prepared_stmt) != 0) {
    print_stmt_error(prepared_stmt, "Errore durante l'inserimento del lavoratore.");
    goto out;
  }
  printf("Lavoratore inserito correttamente...\n");
  out:
    {mysql_stmt_close(prepared_stmt);}
  printf("\nVuoi inserire un altro lavoratore? \n1) Sì \n2) No\n");
    op = multiChoice("Select: ", option, 2);
    switch(op)
    {
       case '1':
          inserisci_lavoratore(conn);
          break;
       case '2':
          printf("\nArrivederci!");
          return;

       default:
          fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
          abort();
```

```
        }
}



static void inserisci_utente(MYSQL *conn)
{
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[3];
    char options[3] = {'1','2', '3'};
    char r;
    char op1;


    char cf[45];
    char password[45];
    char ruolo[45];
    // Get the required information
    printf("\nCodice fiscale: ");
    getInput(45, cf, false);
    printf("Password: ");
    getInput(45, password, true);
    printf("Assegnare uno tra i ruoli disponibili:\n");
    printf("\t1) Amministratore\n");
    printf("\t2) Capoprogetto\n");
    printf("\t3) Dipendente\n");
```

```
r = multiChoice("Select role", options, 3);
// Converte ruolo nel valore corrispondente
switch(r) {
    case '1':
        strcpy(ruolo, "Amministratore");
        break;
    case '2':
        strcpy(ruolo, "Capoprogetto");
        break;
    case '3':
        strcpy(ruolo, "Dipendente");
        break;
    default:
        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
        abort();
}
// Preparazione chiamata store procedure
if(!setup_prepared_stmt(&prepared_stmt, "call insert_utente(?, ?, ?)", conn)) {
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare l'inserimento dell'utente\n", false);
}
// Prepare parameters
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[0].buffer = cf;
param[0].buffer_length = strlen(cf);
param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
param[1].buffer = password;
param[1].buffer_length = strlen(password);
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
param[2].buffer = ruolo;
param[2].buffer_length = strlen(ruolo);
if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
   finish_with_stmt_error(conn, prepared_stmt, "Non è stato possibile effettuare il bind dei parametri per l'inserimento dell'utente\n", true);
}
// Esecuzione procedura
if (mysql_stmt_execute(prepared_stmt) != 0)
{
   print_stmt_error (prepared_stmt, "Errore nell'inserimento dell'utente.");
}
else
{
   printf("Utente inserito correttamente...\n");
}
mysql_stmt_close(prepared_stmt);

   printf("\nVuoi inserire un altro utente? \n1) Sì \n2) No\n");
   op1 = multiChoice("Select: ", options, 2);
```

```
        switch(op1)
        {
            case '1':
                inserisci_utente(conn);
                break;
            case '2':
                printf("\nArrivederci!");
                return;

            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
        }



}



void run_as_amministratore(MYSQL *conn)
{
    char options[5] = {'1','2', '3', '4', '5'};
    char op;
    printf("Benvenuto amministratore.\n");
    if(!parse_config("users/amministratore.json", &conf)) {
```

```
    fprintf(stderr, "Non è stato possibile attivare la modalità d'uso prevista!\n");
    exit(EXIT_FAILURE);
}
if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
    fprintf(stderr, "mysql_change_user() failed\n");
    exit(EXIT_FAILURE);
}
while(true) {
    printf("\033[2J\033[H");
    printf("*** Azioni disponibili: ***\n\n");
    printf("1) Inserire un nuovo progetto\n");
    printf("2) Inserire un nuovo lavoratore\n");
    printf("3) Inserire un nuovo utente\n");
    printf("4) Gestire informazioni riguardanti progetti esistenti\n");
    printf("5) Chiudere l'applicazione\n");
    op = multiChoice("Select an option", options, 5);
    switch(op) {
        case '1':
            inserisci_progetto(conn);
            break;
        case '2':
            inserisci_lavoratore(conn);
            break;
        case '3':
```

```
            inserisci_utente(conn);
            break;
        case '4':
                stampa_progetti(conn);
                break;
        case '5':
            return;
        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
      }
   }
}
```

capoprogetto.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "utils.h"
#include "defines.h"



static void inserisci_canale_di_comunicazione(MYSQL *conn)
{
```

```
MYSQL_STMT *prepared_stmt;
char options[2] = {'1','2'};
char op1;
// parametri necessari all'inserimento del canale
char nomecanale[45];
char id_c[45];
int id;
// retrieve dei parametri necessari
printf("Inserire ID del progetto associato: ");
while(true)
{
   getInput(45, id_c, false);
   if(int_compare(id_c))
   break;
   else printf("Formato errato, riprova: ");
}
id = atoi(id_c);
printf("\nInserire il nome da associare al canale: ");
getInput(45, nomecanale, false);



MYSQL_BIND param[3];
if(!setup_prepared_stmt(&prepared_stmt, "call creazione_canale( ?, ?, ? )", conn))
{
```

```
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile inserire il canale di comunicazione\n", false);
  }
  // Preparazione parametri
  memset(param, 0, sizeof(param));
  param[0].buffer_type = MYSQL_TYPE_LONG;
  param[0].buffer = &id;
  param[0].buffer_length = sizeof(id);
  param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
  param[1].buffer = nomecanale;
  param[1].buffer_length = strlen(nomecanale);
  param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
  param[2].buffer = conf.cf;
  param[2].buffer_length = strlen(conf.cf);

  if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
  {
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri\n", true);
  }
  // Esecuzione procedura
  if (mysql_stmt_execute(prepared_stmt) != 0)
  {
    print_stmt_error (prepared_stmt, "Errore nell'inserimento del canale di comunicazione.");
    mysql_stmt_close(prepared_stmt);
```

```
    }
    else
    {
        printf("Canale di comunicazione inserito correttamente...\n");
        mysql_stmt_close(prepared_stmt);



        printf("\nVuoi inserire un altro canale di comunicazione? \n1) Sì \n2) No\n");
        op1 = multiChoice("Select: ", options, 2);
        switch(op1)
        {
            case '1':
                inserisci_canale_di_comunicazione(conn);
                break;
            case '2':
                return;


            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
        }
    }
}
```

```
static void inserisci_appartenenza_canale(MYSQL *conn)
{
   MYSQL_STMT *prepared_stmt0;
   char options[2] = {'1','2'};
   char op1;
   int results0;
   char header0[512];
   int status;
   // fase di lettura
   if(!setup_prepared_stmt(&prepared_stmt0, "call retrieve_appartenenza_coordinazione(?)", conn))
   {
      finish_with_stmt_error(conn, prepared_stmt0, "Impossibile stampare la lista dei progetti\n", false);
   }
   // sistemazione parametri
   MYSQL_BIND param0[1];
   memset(param0, 0, sizeof(param0));
   param0[0].buffer_type = MYSQL_TYPE_VAR_STRING;
   param0[0].buffer = conf.cf;
   param0[0].buffer_length = strlen(conf.cf);
   if (mysql_stmt_bind_param(prepared_stmt0, param0) != 0)
   {
```

```
    finish_with_stmt_error(conn, prepared_stmt0, "Non è stato possibile effettuare il bind dei parametri\n" , true);
  }

  if (mysql_stmt_execute(prepared_stmt0) != 0)
  {
    print_stmt_error (prepared_stmt0, "Errore nella stampa della lista dei progetti.");
    goto out;
  }

    do {
    // Skip OUT variables (although they are not present in the procedure...)
    if(conn->server_status & SERVER_PS_OUT_PARAMS) {
      goto next;
    }
    dump_result_set(conn, prepared_stmt0, header0, &results0);
    mysql_free_result(rs_metadata);
    next:
    status = mysql_stmt_next_result(prepared_stmt0);
    if (status > 0){
      finish_with_stmt_error(conn, prepared_stmt0, "Unexpected condition", true);
    }
    if(results0 == 0) printf("Nessuna disponibilità\n");
  } while (status == 0);
```

```
out:
mysql_stmt_close(prepared_stmt0);


//fase di inserimento
MYSQL_STMT *prepared_stmt;
// parametri necessari
char cf[45];
char id_c[45];
int id;
char cod_c[45];
int cod;
// retrieve dei parametri necessari
printf("Inserire codice del canale di interesse: ");
while(true)
{
   getInput(45, cod_c, false);
   if(int_compare(cod_c))
   break;
   else printf("Formato errato, riprova: ");
}
cod = atoi(cod_c);
printf("Inserire ID del progetto associato: ");
while(true)
{
```

```
    getInput(45, id_c, false);
    if(int_compare(id_c))
    break;
    else printf("Formato errato, riprova: ");
  }
  id = atoi(id_c);
  printf("\nInserire il codice fiscale del dipendente: ");
  getInput(45, cf, false);



  MYSQL_BIND param[3];
  if(!setup_prepared_stmt(&prepared_stmt, "call assegnazione_canale( ?, ?, ? )", conn))
  {
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile invitare il dipendente nel canale\n", false);
  }
  // Preparazione parametri
  memset(param, 0, sizeof(param));
  param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
  param[0].buffer = cf;
  param[0].buffer_length = strlen(cf);
  param[1].buffer_type = MYSQL_TYPE_LONG;
  param[1].buffer = &cod;
  param[1].buffer_length = sizeof(cod);
  param[2].buffer_type = MYSQL_TYPE_LONG;
```

```
    param[2].buffer = &id;
    param[2].buffer_length = sizeof(id);

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
    {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri\n", true);
    }
    // Esecuzione procedura
    if (mysql_stmt_execute(prepared_stmt) != 0)
    {
        print_stmt_error (prepared_stmt, "Errore nell'inserimento del dipendente nel canale.");
        mysql_stmt_close(prepared_stmt);
    }
    else
    {
        printf("Dipendente inserito correttamente...\n");
        mysql_stmt_close(prepared_stmt);


        printf("\nVuoi inserire un altro dipendente in un canale di comunicazione? \n1) Sì \n2) No\n");
        op1 = multiChoice("Select: ", options, 2);
        switch(op1)
        {
            case '1':
```

```
            inserisci_appartenenza_canale(conn);
            break;
        case '2':
            return;

        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
      }
   }
}

static void inserisci_assegnazione_progetto(MYSQL *conn)
{
   MYSQL_STMT *prepared_stmt0;
   char options[2] = {'1','2'};
   char op1;
   int results0;
   char header0[512];
   int status0;
   // prima fase di lettura
   if(!setup_prepared_stmt(&prepared_stmt0, "call retrieve_appartenenza_coordinazione(?)", conn))
   {
      finish_with_stmt_error(conn, prepared_stmt0, "Impossibile stampare la lista dei progetti\n", false);
```

```
}
// sistemazione parametri
MYSQL_BIND param0[1];
memset(param0, 0, sizeof(param0));
param0[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param0[0].buffer = conf.cf;
param0[0].buffer_length = strlen(conf.cf);
if (mysql_stmt_bind_param(prepared_stmt0, param0) != 0)
{
   finish_with_stmt_error(conn, prepared_stmt0, "Non è stato possibile effettuare il bind dei parametri\n" , true);
}

if (mysql_stmt_execute(prepared_stmt0) != 0)
{
   print_stmt_error (prepared_stmt0, "Errore nella stampa della lista dei progetti.");
   goto out;
}



   do {
   // Skip OUT variables (although they are not present in the procedure...)
   if(conn->server_status & SERVER_PS_OUT_PARAMS) {
      goto next;
   }
```

```
        dump_result_set(conn, prepared_stmt0, header0, &results0);
        mysql_free_result(rs_metadata);
        next:
        status0 = mysql_stmt_next_result(prepared_stmt0);
        if (status0 > 0){
            finish_with_stmt_error(conn, prepared_stmt0, "Unexpected condition", true);
        }
        if(results0 == 0) printf("Nessuna disponibilità\n");
    } while (status0 == 0);
    out:
    mysql_stmt_close(prepared_stmt0);
    // seconda fase di lettura
    int results1;
    char header1[512];
    int status1;
    MYSQL_STMT *prepared_stmt1;
    if(!setup_prepared_stmt(&prepared_stmt1, "call retrieve_dipendenti()", conn))
    {
        finish_with_stmt_error(conn, prepared_stmt1, "Impossibile stampare la lista dei dipendenti\n",
false);
    }
    /* sistemazione parametri
    MYSQL_BIND param1[1];
    memset(param, 0, sizeof(param));
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[0].buffer = conf.cf;
param[0].buffer_length = strlen(conf.cf);
if (mysql_stmt_bind_param(prepared_stmt1, param1) != 0)
{
   finish_with_stmt_error(conn, prepared_stmt1, "Non è stato possibile effettuare il bind dei parametri\n" , true);
}*/

if (mysql_stmt_execute(prepared_stmt1) != 0)
{
   print_stmt_error (prepared_stmt1, "Errore nella stampa della lista dei dipendenti.");
   goto out1;
}


   do {
   // Skip OUT variables (although they are not present in the procedure...)
   if(conn->server_status & SERVER_PS_OUT_PARAMS) {
      goto next1;
   }
   dump_result_set(conn, prepared_stmt1, header1, &results1);
   mysql_free_result(rs_metadata);
   next1:
   status1 = mysql_stmt_next_result(prepared_stmt1);
   if (status1 > 0){
```

```
        finish_with_stmt_error(conn, prepared_stmt1, "Unexpected condition", true);
    }
    if(results1 == 0) printf("Nessuna disponibilità\n");
} while (status1 == 0);
out1:
mysql_stmt_close(prepared_stmt1);

//fase di inserimento
MYSQL_STMT *prepared_stmt;
// parametri necessari
char cf[45];
char id_c[45];
int id;

// retrieve dei parametri necessari
printf("Inserire ID del progetto associato: ");
while(true)
{
    getInput(45, id_c, false);
    if(int_compare(id_c))
    break;
    else printf("Formato errato, riprova: ");
}
id = atoi(id_c);
```

```
printf("\nInserire il codice fiscale del dipendente: ");
getInput(45, cf, false);

MYSQL_BIND param[2];
if(!setup_prepared_stmt(&prepared_stmt, "call assegnazione_progetto( ?, ? )", conn))
{
  finish_with_stmt_error(conn, prepared_stmt, "Impossibile assegnare il progetto al dipendente\n", false);
}
// Preparazione parametri
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param[0].buffer = cf;
param[0].buffer_length = strlen(cf);
param[1].buffer_type = MYSQL_TYPE_LONG;
param[1].buffer = &id;
param[1].buffer_length = sizeof(id);

if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
{
  finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri\n", true);
}
// Esecuzione procedura
if (mysql_stmt_execute(prepared_stmt) != 0)
```

```
{
    print_stmt_error (prepared_stmt, "Errore nell'assegnazione del progetto al dipendente.");
    mysql_stmt_close(prepared_stmt);
}
else
{
    printf("Dipendente inserito correttamente...\n");
    mysql_stmt_close(prepared_stmt);



    printf("\nVuoi assegnare un progetto ad un dipendente? \n1) Sì \n2) No\n");
    op1 = multiChoice("Select: ", options, 2);
    switch(op1)
    {
        case '1':
            inserisci_assegnazione_progetto(conn);
            break;
        case '2':
            return;


        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
    }
```

```
    }
}



static void consultazione_canale(MYSQL *conn){
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[2];
    char header[512];
    int results;
    //parametri necessari
    char id_c[45];
    char cod_c[45];
    int id;
    int cod;
    int status;
    // Retrieve informazioni
    printf("Inserire ID del progetto: ");
    while(true)
    {
        getInput(45, id_c, false);
        if(int_compare(id_c))
        break;
        else printf("Formato errato, riprova: ");
    }
```

```
id = atoi(id_c);
printf("Inserire codice del canale: ");
while(true)
{
   getInput(45, cod_c, false);
   if(int_compare(cod_c))
   break;
   else printf("Formato errato, riprova: ");
}
cod = atoi(cod_c);
// sistemazione parametri
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG;
param[0].buffer = &cod;
param[0].buffer_length = sizeof(cod);
param[1].buffer_type = MYSQL_TYPE_LONG;
param[1].buffer = &id;
param[1].buffer_length = sizeof(id);
// chiamata procedura
if(!setup_prepared_stmt(&prepared_stmt, "call retrieve_conversazioni( ?, ? )", conn))
{
   finish_with_stmt_error(conn, prepared_stmt, "Impossibile stampare le conversazioni\n", false);
}
```

```
if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri\n", true);
}



if (mysql_stmt_execute(prepared_stmt) != 0)
{
    print_stmt_error (prepared_stmt, "Errore nella stampa delle conversazioni.");
    mysql_stmt_close(prepared_stmt);
        return;
}


        // stampa conversazioni
    do {
    // Skip OUT variables (although they are not present in the procedure...)
    if(conn->server_status & SERVER_PS_OUT_PARAMS) {
        goto next;
    }
    dump_result_set(conn, prepared_stmt, header, &results);
    mysql_free_result(rs_metadata);
    next:
    status = mysql_stmt_next_result(prepared_stmt);
    if (status > 0){
        finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
```

```
        }
        if(results == 0) printf("Nessuna disponibilità\n");
    } while (status == 0);
    out:
    mysql_stmt_close(prepared_stmt);


}



static void stampa_progetti_coordinazione(MYSQL *conn){
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[1];
    char op1;
    int status;
    char header[512];
    char options[5] = {'1','2','3','4','5'};
    int results;


    if(!setup_prepared_stmt(&prepared_stmt, "call retrieve_coordinazione(?)", conn))
    {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile stampare la lista dei progetti\n", false);
    }
```

```
   // sistemazione parametri
   memset(param, 0, sizeof(param));
   param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
   param[0].buffer = conf.cf;
   param[0].buffer_length = strlen(conf.cf);
   if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
   {
      finish_with_stmt_error(conn, prepared_stmt, "Non è stato possibile effettuare il bind dei parametri\n" , true);
   }

   if (mysql_stmt_execute(prepared_stmt) != 0)
   {
      print_stmt_error (prepared_stmt, "Errore nella stampa della lista dei progetti.");
      goto out;
   }

do {
      // Skip OUT variables (although they are not present in the procedure...)
      if(conn->server_status & SERVER_PS_OUT_PARAMS) {
         goto next;
      }
      dump_result_set(conn, prepared_stmt, header, &results);
      mysql_free_result(rs_metadata);
```

```
    next:
    status = mysql_stmt_next_result(prepared_stmt);
    if (status > 0){
        finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
    }
    if(results == 0) printf("Nessuna disponibilità\n");
} while (status == 0);
out:
mysql_stmt_close(prepared_stmt);


    printf("*** Azioni disponibili: ***\n\n");
    printf("1) Creare un canale di comunicazione afferente ad un progetto\n");
    printf("2) Aggiungere un dipendente ad un canale di comunicazione\n");
    printf("3) Consultare un canale di comunicazione\n");
        printf("4) Assegnare un progetto ad un dipendente\n");
    printf("5) Chiudere l'applicazione\n");
    op1 = multiChoice("Select: ", options, 4);
    switch(op1)
    {
        case '1':
            inserisci_canale_di_comunicazione(conn);
            break;
        case '2':
            inserisci_appartenenza_canale(conn);
```

```
            break;
        case '3':
            consultazione_canale(conn);
            break;
        case '4':
            inserisci_assegnazione_progetto(conn);
            break;
        case '5':
            return;

        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
    }

}

// attività di messaggistica legate ai progetti
static void inserisci_risposta_privata(MYSQL *conn, int id)
{
    MYSQL_STMT *prepared_stmt;
    char options[2] = {'1','2'};
    char op1;
    int idprog = id;
```

```
// parametri necessari all'inserimento del messaggio
char testo[700];
MYSQL_TIME parsed_data;
char data[11];
char orario[9];
char cfdest[45];

// retrieve dei parametri necessari
printf("Inserire il codice fiscale del destinatario: ");
getInput(45, cfdest, false);
printf("Inserisci l'orario di invio del messaggio a cui vuoi rispondere(hh:mm:ss): ");
        while(true){
            getInput(9, orario, false);
            if(time_compare(orario)) break;
            else printf("Formato sbagliato, riprova: ");
        }
printf("Inserire la data di invio del messaggio a cui vuoi rispondere: ");
        while(true){
            getInput(11, data, false);
            if(date_compare(data)) break;
            else printf("Formato sbagliato, riprova: ");
        }
parse_date(data, &parsed_data);
printf("Inserire il testo del messaggio: ");
```

```
getInput(700, testo, false);



MYSQL_BIND param[6];
if(!setup_prepared_stmt(&prepared_stmt, "call risposta_privata(?, ?, ?, ?, ?, ?)", conn))
{
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile inserire il messaggio\n", false);
}
// Preparazione parametri
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param[0].buffer = conf.cf;
param[0].buffer_length = strlen(conf.cf);
param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
param[1].buffer = testo;
param[1].buffer_length = strlen(testo);
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
param[2].buffer = cfdest;
param[2].buffer_length = strlen(cfdest);
param[3].buffer_type = MYSQL_TYPE_DATE;
param[3].buffer = &parsed_data;
param[3].buffer_length = sizeof(MYSQL_TIME);
param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
param[4].buffer = orario;
```

```
param[4].buffer_length = strlen(orario);
param[5].buffer_type = MYSQL_TYPE_LONG;
param[5].buffer = &idprog;
param[5].buffer_length = sizeof(idprog);

if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
{
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri\n", true);
}
// Esecuzione procedura
if (mysql_stmt_execute(prepared_stmt) != 0)
{
    print_stmt_error (prepared_stmt, "Errore nell'inserimento della risposta.");
    mysql_stmt_close(prepared_stmt);
}
else
{
    printf("Risposta inserita correttamente...\n");
    mysql_free_result(rs_metadata);
    mysql_stmt_close(prepared_stmt);


    printf("\nVuoi rispondere ad un altro messaggio? \n1) Sì \n2) No\n");
    op1 = multiChoice("Select: ", options, 2);
```

```
        switch(op1)
        {
            case '1':
                inserisci_risposta_privata(conn, id);
                break;
            case '2':
                return;

            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
        }
    }
}



static void inserisci_risposta_pubblica(MYSQL *conn, int id, int cod)
{
    MYSQL_STMT *prepared_stmt;
    char options[2] = {'1','2'};
    char op1;
    int idprog = id;
    int codcanale = cod;
    // parametri necessari all'inserimento del messaggio
```

```
char testo[700];
MYSQL_TIME parsed_data;
char data[11];
char orario[9];
char cfdest[45];

// retrieve dei parametri necessari
printf("Inserire il codice fiscale del destinatario: ");
getInput(45, cfdest, false);
printf("Inserisci l'orario di invio del messaggio a cui vuoi rispondere(hh:mm:ss): ");
        while(true){
            getInput(9, orario, false);
            if(time_compare(orario)) break;
            else printf("Formato sbagliato, riprova: ");
        }
printf("Inserire la data di invio del messaggio a cui vuoi rispondere: ");
        while(true){
            getInput(11, data, false);
            if(date_compare(data)) break;
            else printf("Formato sbagliato, riprova: ");
        }
parse_date(data, &parsed_data);
printf("Inserire il testo del messaggio: ");
getInput(700, testo, false);
```

```
MYSQL_BIND param[7];
if(!setup_prepared_stmt(&prepared_stmt, "call risposta_pubblica(?, ?, ?, ?, ?, ?, ?)", conn))
{
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile inserire il messaggio\n", false);
}
// Preparazione parametri
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param[0].buffer = conf.cf;
param[0].buffer_length = strlen(conf.cf);
param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
param[1].buffer = testo;
param[1].buffer_length = strlen(testo);
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
param[2].buffer = cfdest;
param[2].buffer_length = strlen(cfdest);
param[3].buffer_type = MYSQL_TYPE_DATE;
param[3].buffer = &parsed_data;
param[3].buffer_length = sizeof(MYSQL_TIME);
param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
param[4].buffer = orario;
param[4].buffer_length = strlen(orario);
```

```
    param[5].buffer_type = MYSQL_TYPE_LONG;
    param[5].buffer = &codcanale;
    param[5].buffer_length = sizeof(codcanale);
    param[6].buffer_type = MYSQL_TYPE_LONG;
    param[6].buffer = &idprog;
    param[6].buffer_length = sizeof(idprog);

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
    {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri\n", true);
    }
    // Esecuzione procedura
    if (mysql_stmt_execute(prepared_stmt) != 0)
    {
        print_stmt_error (prepared_stmt, "Errore nell'inserimento della risposta.");
        mysql_stmt_close(prepared_stmt);
    }
    else
    {
        printf("Risposta inserita correttamente...\n");
        mysql_free_result(rs_metadata);
        mysql_stmt_close(prepared_stmt);
```

```
        printf("\nVuoi rispondere ad un altro messaggio? \n1) Sì \n2) No\n");
        op1 = multiChoice("Select: ", options, 2);
        switch(op1)
        {
           case '1':
              inserisci_risposta_pubblica(conn, idprog, codcanale);
              break;
           case '2':
              return;

           default:
              fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
              abort();
        }
     }
}




static void inserisci_messaggio(MYSQL *conn, int id, int cod)
{

    MYSQL_STMT *prepared_stmt;
```

```
char options[2] = {'1','2'};
char op1;
int idprog = id;
int codcanale = cod;
// parametri necessari all'inserimento del messaggio
char testo[700];

// retrieve dei parametri necessari
printf("Inserire il testo del messaggio: ");
getInput(700, testo, false);



MYSQL_BIND param[4];
if(!setup_prepared_stmt(&prepared_stmt, "call insert_messaggio(?, ?, ?, ? )", conn))
{
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile inserire il messaggio\n", false);
}
// Preparazione parametri
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param[0].buffer = conf.cf;
param[0].buffer_length = strlen(conf.cf);
param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
param[1].buffer = testo;
```

```
    param[1].buffer_length = strlen(testo);
    param[2].buffer_type = MYSQL_TYPE_LONG;
    param[2].buffer = &codcanale;
    param[2].buffer_length = sizeof(codcanale);
    param[3].buffer_type = MYSQL_TYPE_LONG;
    param[3].buffer = &idprog;
    param[3].buffer_length = sizeof(idprog);

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
    {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri\n", true);
    }
    // Esecuzione procedura
    if (mysql_stmt_execute(prepared_stmt) != 0)
    {
        print_stmt_error (prepared_stmt, "Errore nell'inserimento del messaggio.");
        mysql_stmt_close(prepared_stmt);
    }
    else
    {
        printf("Messaggio inserito correttamente...\n");
        mysql_free_result(rs_metadata);
        mysql_stmt_close(prepared_stmt);
```

```
        printf("\nVuoi inserire un altro messaggio? \n1) Sì \n2) No\n");
        op1 = multiChoice("Select: ", options, 2);
        switch(op1)
        {
           case '1':
              inserisci_messaggio(conn, idprog, codcanale);
              break;
           case '2':
              return;

           default:
              fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
              abort();
        }
     }
}
```

```
static void consultazione_canale_scrittura(MYSQL *conn){
    MYSQL_STMT *prepared_stmt;
    int status;
    MYSQL_BIND param[2];
    char header[512];
    int results;
    char options[4] = {'1','2','3','4'};
    char op1;
    //parametri necessari
    char id_c[45];
    char cod_c[45];
    int id;
    int cod;
    // Retrieve informazioni
    printf("Inserire ID del progetto: ");
    while(true)
    {
        getInput(45, id_c, false);
        if(int_compare(id_c))
        break;
        else printf("Formato errato, riprova: ");
    }
    id = atoi(id_c);
    printf("Inserire codice del canale: ");
```

```
while(true)
{
    getInput(45, cod_c, false);
    if(int_compare(cod_c))
    break;
    else printf("Formato errato, riprova: ");
}
cod = atoi(cod_c);
// sistemazione parametri
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG;
param[0].buffer = &cod;
param[0].buffer_length = sizeof(cod);
param[1].buffer_type = MYSQL_TYPE_LONG;
param[1].buffer = &id;
param[1].buffer_length = sizeof(id);
// chiamata procedura
if(!setup_prepared_stmt(&prepared_stmt, "call retrieve_conversazioni( ?, ? )", conn))
{
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile stampare le conversazioni\n", false);
}

if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri\n",
true);
```

```
}



if (mysql_stmt_execute(prepared_stmt) != 0)
{
   print_stmt_error (prepared_stmt, "Errore nella stampa delle conversazioni.");
   mysql_stmt_close(prepared_stmt);
      return;
}
      // stampa conversazioni
      do {
   // Skip OUT variables (although they are not present in the procedure...)
   if(conn->server_status & SERVER_PS_OUT_PARAMS) {
      goto next;
   }
   dump_result_set(conn, prepared_stmt, header, &results);
   mysql_free_result(rs_metadata);
   next:
   status = mysql_stmt_next_result(prepared_stmt);
   if (status > 0){
      finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
   }
   if(results == 0) printf("Nessuna disponibilità\n");
} while (status == 0);
```

```
out:
mysql_stmt_close(prepared_stmt);


    printf("*** Azioni disponibili: ***\n\n");
  printf("1) Scrivere un messaggio\n");
  printf("2) Rispondere pubblicamente ad un messaggio\n");
  printf("3) Rispondere privatamente ad un messaggio\n");
  op1 = multiChoice("Select:", options, 4);
  switch(op1)
  {
    case '1':
      inserisci_messaggio(conn, id, cod);
      break;
    case '2':
      inserisci_risposta_pubblica(conn, id, cod);
      break;
    case '3':
      inserisci_risposta_privata(conn, id);
      break;
    case '4':
      return;

    default:
      fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
```

```
            abort();
        }

}
```

```
static void stampa_progetti_appartenenza(MYSQL *conn){
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[1];
    int status;
    char op1;
    char header[512];
    char options[2] = {'1','2'};
    int results;

    if(!setup_prepared_stmt(&prepared_stmt, "call retrieve_appartenenza(?)", conn))
    {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile stampare la lista dei progetti\n", false);
    }
```

```
// sistemazione parametri
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param[0].buffer = conf.cf;
param[0].buffer_length = strlen(conf.cf);
if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
{
   finish_with_stmt_error(conn, prepared_stmt, "Non è stato possibile effettuare il bind dei parametri\n" , true);
}

if (mysql_stmt_execute(prepared_stmt) != 0)
{
   print_stmt_error (prepared_stmt, "Errore nella stampa della lista dei progetti.");
   goto out;
}


   do {
   // Skip OUT variables (although they are not present in the procedure...)
   if(conn->server_status & SERVER_PS_OUT_PARAMS) {
      goto next;
   }
   dump_result_set(conn, prepared_stmt, header, &results);
   mysql_free_result(rs_metadata);
```

```
    next:
    status = mysql_stmt_next_result(prepared_stmt);
    if (status > 0){
        finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
    }
    if(results == 0) printf("Nessuna disponibilità\n");
} while (status == 0);
out:
mysql_stmt_close(prepared_stmt);



    printf("*** Azioni disponibili: ***\n\n");
    printf("1) Consultare un canale di comunicazione\n");
    printf("2) Chiudere l'applicazione\n");
    op1 = multiChoice("Select: ", options, 2);
    switch(op1)
    {
        case '1':
            consultazione_canale_scrittura(conn);
            break;
        case '2':
            return;

        default:
```

```
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
        }

}




void run_as_capoprogetto(MYSQL *conn)
{
    char options[3] = {'1','2','3'};
    char op;
    printf("Benvenuto capoprogetto!\n");
    if(!parse_config("users/capoprogetto.json", &conf)) {
        fprintf(stderr, "Non è stato possibile attivare la modalità d'uso prevista!\n");
        exit(EXIT_FAILURE);
    }
    if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
        fprintf(stderr, "mysql_change_user() failed\n");
        exit(EXIT_FAILURE);
    }
    while(true) {
        printf("\033[2J\033[H");
        printf("*** Azioni disponibili: ***\n\n");
```

```
        printf("1) Attività di coordinazione sui progetti\n");
            printf("2) Attività di messaggistica legate ai progetti\n");
        printf("3) Chiudere l'applicazione\n");
        op = multiChoice("Select an option", options, 3);
        switch(op) {
            case '1':
                stampa_progetti_coordinazione(conn);
                break;
            case '2':
                        stampa_progetti_appartenenza(conn);
            case '3':
                return;
            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
        }
        getchar();
    }
}
```

dipendente.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include "utils.h"
#include "defines.h"



// attività di messaggistica legate ai progetti
static void inserisci_risposta_privata(MYSQL *conn, int id)
{
    MYSQL_STMT *prepared_stmt;
    char options[2] = {'1','2'};
    char op1;
    int idprog = id;
    // parametri necessari all'inserimento del messaggio
    char testo[700];
    MYSQL_TIME parsed_data;
    char data[11];
    char orario[9];
    //char *parsed_time; elimina
    char cfdest[45];

    // retrieve dei parametri necessari
    printf("Inserire il codice fiscale del destinatario: ");
    getInput(45, cfdest, false);
    printf("Inserisci l'orario di invio del messaggio a cui vuoi rispondere(hh:mm:ss): ");
```

```
        while(true){
            getInput(9, orario, false);
            if(time_compare(orario)) break;
            else printf("Formato sbagliato, riprova: ");
        }
        // parsed_time = parse_time(orario); elimina
printf("Inserire la data di invio del messaggio a cui vuoi rispondere: ");
        while(true){
            getInput(11, data, false);
            if(date_compare(data)) break;
            else printf("Formato sbagliato, riprova: ");
        }
parse_date(data, &parsed_data);
printf("Inserire il testo del messaggio: ");
getInput(700, testo, false);



MYSQL_BIND param[6];
if(!setup_prepared_stmt(&prepared_stmt, "call risposta_privata(?, ?, ?, ?, ?, ?)", conn))
{
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile inserire il messaggio\n", false);
}
// Preparazione parametri
memset(param, 0, sizeof(param));
```

```
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer = conf.cf;
    param[0].buffer_length = strlen(conf.cf);
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = testo;
    param[1].buffer_length = strlen(testo);
    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = cfdest;
    param[2].buffer_length = strlen(cfdest);
    param[3].buffer_type = MYSQL_TYPE_DATE;
    param[3].buffer = &parsed_data;
    param[3].buffer_length = sizeof(MYSQL_TIME);
    param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[4].buffer = orario;
    param[4].buffer_length = strlen(orario);
    param[5].buffer_type = MYSQL_TYPE_LONG;
    param[5].buffer = &idprog;
    param[5].buffer_length = sizeof(idprog);

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
    {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri\n", true);
    }
    // Esecuzione procedura
```

```
if (mysql_stmt_execute(prepared_stmt) != 0)
{
    print_stmt_error (prepared_stmt, "Errore nell'inserimento della risposta.");
    mysql_stmt_close(prepared_stmt);
}
else
{
    printf("Risposta inserita correttamente...\n");
    mysql_free_result(rs_metadata);
    mysql_stmt_close(prepared_stmt);



    printf("\nVuoi rispondere ad un altro messaggio? \n1) Sì \n2) No\n");
    op1 = multiChoice("Select: ", options, 2);
    switch(op1)
    {
        case '1':
            inserisci_risposta_privata(conn, id);
            break;
        case '2':
            printf("\nArrivederci!");
            return;

        default:
```

```
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
        }
    }
}


static void inserisci_risposta_pubblica(MYSQL *conn, int id, int cod)
{
    MYSQL_STMT *prepared_stmt;
    char options[2] = {'1','2'};
    char op1;
    int idprog = id;
    int codcanale = cod;
    // parametri necessari all'inserimento del messaggio
    char testo[700];
    MYSQL_TIME parsed_data;
    char data[11];
    char orario[9];
    //char *parsed_time; elimina
    char cfdest[45];

    // retrieve dei parametri necessari
    printf("Inserire il codice fiscale del destinatario: ");
```

```
getInput(45, cfdest, false);
printf("Inserisci l'orario di invio del messaggio a cui vuoi rispondere(hh:mm:ss): ");
        while(true){
            getInput(9, orario, false);
            if(time_compare(orario)) break;
            else printf("Formato sbagliato, riprova: ");
        }
        // parsed_time = parse_time(orario); elimina
printf("Inserire la data di invio del messaggio a cui vuoi rispondere: ");
        while(true){
            getInput(11, data, false);
            if(date_compare(data)) break;
            else printf("Formato sbagliato, riprova: ");
        }
parse_date(data, &parsed_data);
printf("Inserire il testo del messaggio: ");
getInput(700, testo, false);



MYSQL_BIND param[7];
if(!setup_prepared_stmt(&prepared_stmt, "call risposta_pubblica(?, ?, ?, ?, ?, ?, ?)", conn))
{
   finish_with_stmt_error(conn, prepared_stmt, "Impossibile inserire il messaggio\n", false);
}
```

```
// Preparazione parametri
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param[0].buffer = conf.cf;
param[0].buffer_length = strlen(conf.cf);
param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
param[1].buffer = testo;
param[1].buffer_length = strlen(testo);
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
param[2].buffer = cfdest;
param[2].buffer_length = strlen(cfdest);
param[3].buffer_type = MYSQL_TYPE_DATE;
param[3].buffer = &parsed_data;
param[3].buffer_length = sizeof(MYSQL_TIME);
param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
param[4].buffer = orario;
param[4].buffer_length = strlen(orario);
param[5].buffer_type = MYSQL_TYPE_LONG;
param[5].buffer = &codcanale;
param[5].buffer_length = sizeof(codcanale);
param[6].buffer_type = MYSQL_TYPE_LONG;
param[6].buffer = &idprog;
param[6].buffer_length = sizeof(idprog);
```

```
   if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
   {
      finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri\n", true);
   }
   // Esecuzione procedura
   if (mysql_stmt_execute(prepared_stmt) != 0)
   {
      print_stmt_error (prepared_stmt, "Errore nell'inserimento della risposta.");
      mysql_stmt_close(prepared_stmt);
   }
   else
   {
      printf("Risposta inserita correttamente...\n");
      mysql_free_result(rs_metadata);
      mysql_stmt_close(prepared_stmt);



      printf("\nVuoi visualizzare la lista dei canali? \n1) Sì \n2) No\n");
      op1 = multiChoice("Select: ", options, 2);
      switch(op1)
      {
         case '1':
            inserisci_risposta_pubblica(conn, idprog, codcanale);
            break;
```

```
            case '2':
                printf("\nArrivederci!");
                return;

            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
        }
    }
}
```

```
static void inserisci_messaggio(MYSQL *conn, int id, int cod)
{
    MYSQL_STMT *prepared_stmt;
    char options[2] = {'1','2'};
    char op1;
    int idprog = id;
    int codcanale = cod;
    // parametri necessari all'inserimento del messaggio
    char testo[700];
```

```
// retrieve dei parametri necessari
printf("Inserire il testo del messaggio: ");
getInput(700, testo, false);



MYSQL_BIND param[4];
if(!setup_prepared_stmt(&prepared_stmt, "call insert_messaggio(?, ?, ?, ? )", conn))
{
   finish_with_stmt_error(conn, prepared_stmt, "Impossibile inserire il messaggio\n", false);
}
// Preparazione parametri
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param[0].buffer = conf.cf;
param[0].buffer_length = strlen(conf.cf);
param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
param[1].buffer = testo;
param[1].buffer_length = strlen(testo);
param[2].buffer_type = MYSQL_TYPE_LONG;
param[2].buffer = &codcanale;
param[2].buffer_length = sizeof(codcanale);
param[3].buffer_type = MYSQL_TYPE_LONG;
param[3].buffer = &idprog;
param[3].buffer_length = sizeof(idprog);
```

```
if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
{
    finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri\n", true);
}
// Esecuzione procedura
if (mysql_stmt_execute(prepared_stmt) != 0)
{
    print_stmt_error (prepared_stmt, "Errore nell'inserimento del messaggio.");
    mysql_stmt_close(prepared_stmt);
}
else
{
    printf("Messaggio inserito correttamente...\n");
    mysql_free_result(rs_metadata);
    mysql_stmt_close(prepared_stmt);



    printf("\nVuoi inserire un altro messaggio? \n1) Sì \n2) No\n");
    op1 = multiChoice("Select: ", options, 2);
    switch(op1)
    {
        case '1':
            inserisci_messaggio(conn, idprog, codcanale);
```

```
            break;
        case '2':
            printf("\nArrivederci!");
            return;

        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
      }
   }
}
```

```
static void consultazione_canale_scrittura(MYSQL *conn){
   MYSQL_STMT *prepared_stmt;
   int status;
   MYSQL_BIND param[2];
   char header[512];
   int results;
```

```
char options[4] = {'1','2','3','4'};
char op1;
//parametri necessari
char id_c[45];
char cod_c[45];
int id;
int cod;
// Retrieve informazioni
printf("Inserire ID del progetto: ");
while(true)
{
    getInput(45, id_c, false);
    if(int_compare(id_c))
    break;
    else printf("Formato errato, riprova: ");
}
id = atoi(id_c);
printf("Inserire codice del canale: ");
while(true)
{
    getInput(45, cod_c, false);
    if(int_compare(cod_c))
    break;
    else printf("Formato errato, riprova: ");
```

```
    }
    cod = atoi(cod_c);
    // sistemazione parametri
    memset(param, 0, sizeof(param));
    param[0].buffer_type = MYSQL_TYPE_LONG;
    param[0].buffer = &cod;
    param[0].buffer_length = sizeof(cod);
    param[1].buffer_type = MYSQL_TYPE_LONG;
    param[1].buffer = &id;
    param[1].buffer_length = sizeof(id);
    // chiamata procedura
    if(!setup_prepared_stmt(&prepared_stmt, "call retrieve_conversazioni( ?, ? )", conn))
    {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile stampare le conversazioni\n", false);
    }

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile effettuare il bind dei parametri\n",
true);
    }

    if (mysql_stmt_execute(prepared_stmt) != 0)
    {
        print_stmt_error (prepared_stmt, "Errore nella stampa delle conversazioni.");
```

```
    mysql_stmt_close(prepared_stmt);
        return;
}
        // stampa conversazioni
        do {
    // Skip OUT variables (although they are not present in the procedure...)
    if(conn->server_status & SERVER_PS_OUT_PARAMS) {
        goto next;
    }
    dump_result_set(conn, prepared_stmt, header, &results);
    mysql_free_result(rs_metadata);
    next:
    status = mysql_stmt_next_result(prepared_stmt);
    if (status > 0){
        finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
    }
    if(results == 0) printf("Nessuna disponibilità\n");
} while (status == 0);
out:
mysql_stmt_close(prepared_stmt);

        printf("*** Azioni disponibili: ***\n\n");
    printf("1) Scrivere un messaggio\n");
    printf("2) Rispondere pubblicamente ad un messaggio\n");
```

```
        printf("3) Rispondere privatamente ad un messaggio\n");
        op1 = multiChoice("Select:", options, 4);
        switch(op1)
        {
           case '1':
              inserisci_messaggio(conn, id, cod);
              break;
           case '2':
              inserisci_risposta_pubblica(conn, id, cod);
              break;
           case '3':
              inserisci_risposta_privata(conn, id);
              break;
           case '4':
              return;

           default:
              fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
              abort();
        }

}
```

```
static void stampa_progetti_appartenenza(MYSQL *conn){
   MYSQL_STMT *prepared_stmt;
   MYSQL_BIND param[1];
   int status;
   char op1;
   char header[512];
   char options[2] = {'1','2'};
   int results;

   if(!setup_prepared_stmt(&prepared_stmt, "call retrieve_appartenenza(?)", conn))
   {
      finish_with_stmt_error(conn, prepared_stmt, "Impossibile stampare la lista dei progetti\n", false);
   }

   // sistemazione parametri
   memset(param, 0, sizeof(param));
   param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
   param[0].buffer = conf.cf;
   param[0].buffer_length = strlen(conf.cf);
   if (mysql_stmt_bind_param(prepared_stmt, param) != 0)
   {
```

```
    finish_with_stmt_error(conn, prepared_stmt, "Non è stato possibile effettuare il bind dei parametri\n" , true);
  }

  if (mysql_stmt_execute(prepared_stmt) != 0)
  {
    print_stmt_error (prepared_stmt, "Errore nella stampa della lista dei progetti.");
    goto out;
  }

    do {
    // Skip OUT variables (although they are not present in the procedure...)
    if(conn->server_status & SERVER_PS_OUT_PARAMS) {
      goto next;
    }
    dump_result_set(conn, prepared_stmt, header, &results);
    mysql_free_result(rs_metadata);
    next:
    status = mysql_stmt_next_result(prepared_stmt);
    if (status > 0){
      finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
    }
    if(results == 0) printf("Nessuna disponibilità\n");
  } while (status == 0);
```

```
    out:
    mysql_stmt_close(prepared_stmt);



        printf("*** Azioni disponibili: ***\n\n");
        printf("1) Consultare un canale di comunicazione\n");
        printf("2) Chiudere l'applicazione\n");
        op1 = multiChoice("Select: ", options, 2);
        switch(op1)
        {
            case '1':
                consultazione_canale_scrittura(conn);
                break;
            case '2':
                return;


            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
        }

}
```

```
void run_as_dipendente(MYSQL *conn)
   {
   char options[2] = {'1','2'};
   char op;
   printf("Benvenuto dipendente!\n");
   if(!parse_config("users/dipendente.json", &conf)) {
      fprintf(stderr, "Non è stato possibile far girare il sistema nella modalità prevista\n");
      exit(EXIT_FAILURE);
   }
   if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
      fprintf(stderr, "mysql_change_user() failed\n");
      exit(EXIT_FAILURE);
   }
   while(true) {
      printf("\033[2J\033[H");
      printf("*** Azioni disponibili: ***\n\n");
      printf("1) Attività di messaggistica legate ai progetti\n");
      printf("2) Chiudere l'applicazione\n");
      op = multiChoice("Select an option", options, 2);
      switch(op) {
         case '1':
            stampa_progetti_appartenenza(conn);
            break;
         case '2':
```

```
            return;
        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
      }
   }
}
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "utils.h"
#include "defines.h"



typedef enum
{
   Amministratore = 1,
   Capoprogetto = 2,
   Dipendente = 3,
   FAILED_LOGIN,
} role_t;
```

```
struct configuration conf;
static MYSQL *conn;

static role_t attempt_login(MYSQL *conn, char *cf, char *password)
{
   MYSQL_STMT *login;
   MYSQL_BIND param[3]; // Used both for input and output
   int role = 0;
   if(!setup_prepared_stmt(&login, "call login(?, ?, ?)", conn)) {
      print_stmt_error(login, "Unable to initialize login statement\n");
      goto err2;
   }
   // Preparazione parametri di input
   memset(param, 0, sizeof(param));
   param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
   param[0].buffer = cf;
   param[0].buffer_length = strlen(cf);

   param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
   param[1].buffer = password;
   param[1].buffer_length = strlen(password);

   param[2].buffer_type = MYSQL_TYPE_LONG; // OUT
   param[2].buffer = &role;
```

```
param[2].buffer_length = sizeof(role);
if (mysql_stmt_bind_param(login, param) != 0) { // Note _param
    print_stmt_error(login, "Could not bind parameters for login");
    goto err;
}
// Esecuzione procedura
if (mysql_stmt_execute(login) != 0) {
    print_stmt_error(login, "Could not execute login procedure");
    goto err;
}
// Preparazione parametri di output
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
param[0].buffer = &role;
param[0].buffer_length = sizeof(role);
if(mysql_stmt_bind_result(login, param)) {
    print_stmt_error(login, "Could not retrieve output parameter");
    goto err;
}
// Recupero parametri di output
if(mysql_stmt_fetch(login)) {
    print_stmt_error(login, "Could not buffer results");
    goto err;
}
```

```
    mysql_stmt_close(login);
    printf("%d\n", role);
    return role;
    err:
        mysql_stmt_close(login);
    err2:
        return FAILED_LOGIN;
}



int main(void) {
    role_t role;
    if(!parse_config("users/login.json", &conf))
    {
        fprintf(stderr, "Unable to load login configuration\n");
        exit(EXIT_FAILURE);
    }
    conn = mysql_init (NULL);
    if (conn == NULL)
    {
        fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
        exit(EXIT_FAILURE);
    }
```

```
if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL)
{
    fprintf (stderr, "mysql_real_connect() failed\n");
    mysql_close (conn);
    exit(EXIT_FAILURE);
}

else
{
    printf("Codice fiscale: ");
    getInput(45, conf.cf, false);
    printf("Password: ");
    getInput(45, conf.password, true);
    role = attempt_login(conn, conf.cf, conf.password);
}
switch(role) {
    case Amministratore:
        run_as_amministratore(conn);
        break;
    case Capoprogetto:
        run_as_capoprogetto(conn);
        break;
    case Dipendente:
        run_as_dipendente(conn);
```

```
            break;
        case FAILED_LOGIN:
            fprintf(stderr, "Le credenziali inserite non corrispondono a nessun utente presente nel sistema\n");
            exit(EXIT_FAILURE);
            break;
        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
    }
    printf("Arrivederci!\n");
    mysql_close (conn);
    return 0;
}
```