Relazione Basi di Dati

2019-2020

Andrea Mansi - 137857
Daniele Passabì - 137758

Indice

Introduzione

Lo scopo del progetto è quello di implementare un database relazionale a partire da specifiche e requisiti verosimili, al fine di approfondire i temi e le problematiche che possono insorgere durante tale sviluppo.

La relazione copre le fasi dello sviluppo, riportando per ciascuna di esse riflessioni, scelte implementative, di progettazione ed eventuali problemi riscontrati.

Obiettivo

Si vuole automatizzare il sistema di gestione degli animali di uno zoo.

- Ogni esemplare di animale ospitato è identificato dal suo genere (ad esempio, zebra) e da un codice unico all'interno del genere di appartenenza. Per ogni esemplare, si memorizzano la data di arrivo nello zoo, il nome proprio, il sesso, il paese di provenienza e la data di nascita.
- Lo zoo è diviso in aree. In ogni area c'è un insieme di case, ognuna destinata ad un determinato genere di animali. Ogni casa contiene un insieme di gabbie, ognuna contenente un solo esemplare. Ogni casa ha un addetto che pulisce ciascuna gabbia in un determinato giorno della settimana.
- Gli animali sono sottoposti periodicamente a controllo veterinario. In un controllo, un veterinario rileva il peso degli esemplari, diagnostica eventuali malattie e prescrive il tipo di dieta da seguire.

Fasi del progetto

Segue la lista delle principali fasi del progetto:

- Raccolta e analisi dei requisiti
- Progettazione concettuale
- Progettazione logica
- Progettazione fisica e implementazione
- Popolazione e analisi dei dati in R

1. Raccolta e analisi dei requisiti

La raccolta e analisi dei requisiti comincia con la stesura di un glossario dei termini. Il suo scopo è quello di fornire, per ogni concetto rilevante: una breve descrizione, eventuali sinonimi, relazioni con altri concetti del glossario stesso.

Glossario dei termini

Termine	Descrizione	Sinonimi	Collegamenti	Tipologia
zoo	Dominio del database	---	---	dominio del DB
esemplare	È identificato dal suo genere e da un codice unico all'interno del genere di appartenenza.	animale	veterinario, gabbia, genere	entità
area	È formata da un insieme di abitazioni in essa collocate.	zona	abitazione	entità
abitazione	È destinata ad un determinato genere di animali; ogni abitazione contiene un insieme di gabbie; ogni abitazione ha un addetto che pulisce ciascuna gabbia.	casa	gabbia, addetto pulizie, area, genere	entità
collocata	Mette in relazione un'abitazione con una determinata area.	---	abitazione, area	relazione
in	Mette in relazione le gabbie con l'abitazione in cui si trovano.	---	gabbia, abitazione	relazione
contenuto	Mette in relazione un esemplare con la gabbia nel quale è contenuto.	---	esemplare, gabbia	relazione
gabbia	Contiene un solo animale. Viene pulita regolarmente da un addetto alle pulizie.	---	esemplare, abitazione	entità
addetto pulizie	Pulisce tutte le gabbie assegnate all'abitazione in base al proprio turno di pulizia.	dipendente	abitazione	entità
pulire	Mette in relazione un addetto delle pulizie con le abitazioni che deve pulire.	---	abitazione, addetto pulizie	relazione
veterinario	Controlla/visita periodicamente gli esemplari.	---	esemplare	entità
visitare	Controllo periodico degli animali, rilevamento del peso, diagnostica di eventuali malattie, prescrizione del tipo di dieta.	visita	veterinario, esemplare	relazione
genere	Identifica il genere di un esemplare con cui è in relazione. Identifica il genere di animali che possono essere contenuti nelle gabbie di una determinata abitazione.	tipologia, specie	abitazione, esemplare	entità
assegnato	Mette in relazione un'abitazione con un genere per identificare il tipo di animali che possono essere contenuti nelle sue gabbie.	---	abitazione, genere	relazione
appartiene	Mette in relazione un esemplare con un genere per identificarne l'appartenenza.	---	esemplare, genere	relazione

Analisi dei requisiti

Segue il risultato dell'analisi dei requisiti. I requisiti ottenuti sono stati organizzati in gruppi, ciascuno contenente le funzionalità che il database dovrà fornire per ogni singola entità chiave.

Dominio: zoo

Il sistema deve permettere la gestione di uno zoo, deve quindi consentire di mantenere le informazioni sugli esemplari ospitati, sulle aree, abitazioni e gabbie dello zoo, sugli addetti alle pulizie e sui veterinari.

Esemplare

- Ogni esemplare è caratterizzato da:
  - genere e codice univoco all'interno del genere di appartenenza
  - data di arrivo nello zoo
  - data di nascita
  - nome proprio
  - sesso
  - paese di provenienza
- Il sistema deve consentire:
  - l'inserimento (e la rimozione) di un esemplare in qualsiasi momento
  - l'assegnazione univoca (e lo spostamento) di ogni esemplare ad una singola gabbia
  - la gestione periodica dei controlli veterinari effettuati a ciascun esemplare

Area

- Le aree sono formate da un insieme di abitazioni
- Le aree sono contraddistinte da un nome (univoco)
- Il sistema deve consentire di:
  - gestire (aggiungere, modificare o rimuovere) le aree dello zoo
  - cambiare (per ciascuna area) le abitazioni che le appartengono
  - tenere traccia del numero di abitazioni assegnate a ciascun'area

Abitazione

- Le abitazioni sono contraddistinte da un ID univoco
- Ogni abitazione:
  - è riservata ad un determinato genere di animale
  - include un insieme di gabbie (se queste contengono un esemplare, egli dovrà essere del genere assegnato all'abitazione)
  - ha un addetto alla pulizia delle gabbie
- Il sistema deve permettere di:
  - gestire (aggiungere, modificare o rimuovere) le abitazioni dello zoo
  - cambiare (per ciascuna abitazione) il genere assegnato
  - tenere traccia del numero di gabbie incluse in ciascun'abitazione

Gabbia

- Ciascuna gabbia è identificata da un ID univoco
- Ogni gabbia contiene al massimo un animale (il cui genere dovrà coincidere con quello dell'abitazione in cui si trova)
- Il sistema deve permettere di:
  - gestire (aggiungere, modificare e rimuovere) le gabbie dello zoo
  - cambiare (per ciascuna gabbia) l'esemplare in essa contenuto

Genere

- Il sistema deve permettere di:
  - gestire (aggiungere, modificare o rimuovere) generi gestiti dallo zoo
  - assegnare a ciascuna abitazione il genere di animale che può ospitare
  - assegnare a ciascun esemplare il genere a cui appartiene

Dipendente

- Ogni dipendente è caratterizzato da:
  - CF (codice fiscale)
  - nome
  - cognome
  - stipendio
  - uno o più numeri di telefono
- I dipendenti sono suddivisi in due categorie:
  - Addetto pulizie
    - pulisce ogni gabbia presente nell'abitazione a cui il suo turno di pulizia fa riferimento
  - Veterinario
    - visita periodicamente gli esemplari
- Il sistema deve permettere di:
  - gestire (aggiungere, modificare o rimuovere) i dipendenti dello zoo
  - tenere traccia dei controlli veterinari effettuati
  - modificare il turno di pulizia e le abitazioni assegnate agli addetti alle pulizie

Visita

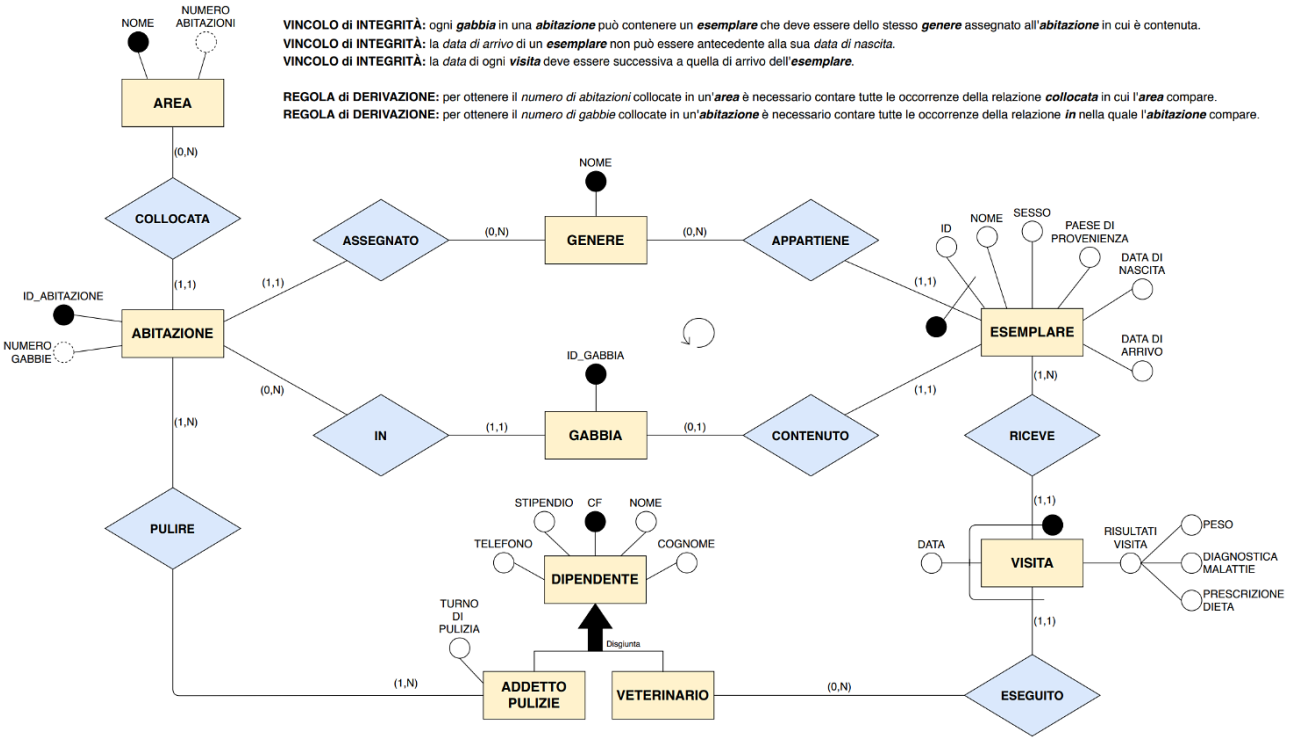
- Ogni visita è caratterizzata da:
  - data, esemplare visitato e veterinario che ha effettuato la visita
  - rilevamento del peso
  - diagnostica di eventuali malattie
  - prescrizione della tipologia di dieta
- Il sistema deve permettere di:
  - gestire (aggiungere, modificare o rimuovere) nuove visite
  - mantenere uno storico delle visite effettuate

Vincoli aggiuntivi

- Il sistema deve garantire che in ogni momento sia rispettato il vincolo di integrità sui generi, ovvero: un esemplare di genere x può essere contenuto solo in una gabbia appartenente ad un'abitazione di genere assegnato x (ovvero il genere di un esemplare deve essere lo stesso del genere assegnato all'abitazione in cui è contenuta la sua gabbia).
- Vincoli di integrità sulle date: il sistema deve controllare le seguenti restrizioni relative alle date:
  1. La data di arrivo di un esemplare non può essere antecedente alla sua data di nascita.
  2. Un esemplare non può aver ricevuto una visita prima che sia arrivato nello zoo.

2. Progettazione concettuale

Segue lo schema entità relazioni frutto della fase di progettazione concettuale.



Alcune considerazioni:

- Come si può notare dallo schema, la chiave di ESEMPLARE è caratterizzata dalla coppia ID, nome; dove nome rappresenta il GENERE a cui l'animale appartiene.
- Le due entità ADDETTO_PULIZIE e VETERINARIO sono state rappresentate attraverso una generalizzazione totale e disgiunta di DIPENDENTE.
- Al fine di modellare il concetto di "storico delle visite" è necessario far si che la chiave dell'entità VISITA sia la quadrupla data da: CF del VETERINARIO, data, ID e GENERE dell'ESEMPLARE.
- Alcune cardinalità delle relazioni sono frutto di ipotesi relative al dominio del database. Segue un esempio: si ipotizza che all'arrivo di un ESEMPLARE nello zoo esso sia sottoposto obbligatoriamente ad una VISITA, da cui (1,N).
- È importante notare la presenza di un ciclo, che può portare a possibili inconsistenze. Ad esempio, sul vincolo di GENERE citato precedentemente.

3. Progettazione logica

A partire dallo schema ER e dalla tabella dei requisiti si è passati alla fase di progettazione logica. Seguono le fasi che la compongono.

Analisi delle prestazioni

Come prima fase è stata effettuata un'analisi delle prestazioni basandosi su tabella dei volumi e delle operazioni. Il risultato è stato poi utilizzato come criterio di valutazione per la ristrutturazione del modello entità relazioni.

Tabella dei volumi e delle operazioni

Le seguenti tabelle sono frutto di una previsione d'uso del sistema ipotetica, in cui i volumi sono risultato di stime.

Tabella dei volumi

	Concetto	Volume
E	Area	10
E	Abitazione	100
E	Gabbia	5000
E	Genere	80
E	Esemplare	4500
E	Addetto pulizie	100
R	Collocata	1000
R	In	5000
R	Contenuto	4500
R	Assegnato	95
R	Appartiene	4500
R	Pulire	300
R	Riceve	270000
R	Eseguito	270000
E	Visita	270000
E	Veterinario	20

Note	
n° medio di abitazioni per area	10
n° medio di gabbie per abitazione	50
percentuale gabbie libere	10%
media esemplari per genere	56,25
media esemplari per abitazione (escluse quelle vuote)	47,368
media gabbie assegnate a ciascun addetto	50
---	---
si suppone che una gabbia vuota sia comunque assegnata	---
n° gabbie vuote	500
percentuale abitazioni senza genere assegnato (vuote)	5%
---	---
n° medio abitazioni assegnate per ciascun addetto alle pulizie	3
---	---
---	---
n° medio visite per esemplare mensili	0,5
n° medio visite effettuate da veterinario dopo 10 anni	13500

durata (in anni) prevista utilizzo database	20

Nota: si ipotizza che il numero di animali nello zoo rimanga pressoché costante, inoltre, il numero di visite (presenti nello storico) è relativo a un periodo di 10 anni (metà vita del database).

Tabella delle operazioni

	Operazione	Frequenza
I	Aggiunta nuovo esemplare	5 a sett.
I	Rimozione esemplare	5 a sett.
I	Ricerca collocazione di un esemplare	4500 al giorno
I	Lettura informazioni relative ad un esemplare	900 al giorno
I	Spostamento di un esemplare	10 a sett.
I	Aggiunta di una nuova visita	563 a sett.
I	Lettura informazioni di visite già effettuate	1125 a sett.
B	Lettura stipendio di ciascun dipendente dello zoo	120 al mese
I	Aggiunta, modifica e rimozione di gabbie	30 al mese
I	Aggiunta, modifica e rimozione di abitazioni	5 al mese
I	Aggiunta, modifica e rimozione dipendenti	5 al mese

Esemplare * n° medio visite mensili * 0.25

Si suppone che in ogni visita vengano lette mediamente le informazioni relative alle due visite precedenti.

Nota: non sono, ovviamente, elencate tutte le possibili operazioni; ma si è ritenuto importante aggiungere quelle più significative (relativamente alle fasi successive).

Frequenza operazioni

Basandosi sulle tabelle dei volumi e delle operazioni sono state individuate le operazioni più e meno frequenti del database.

Operazioni di scrittura

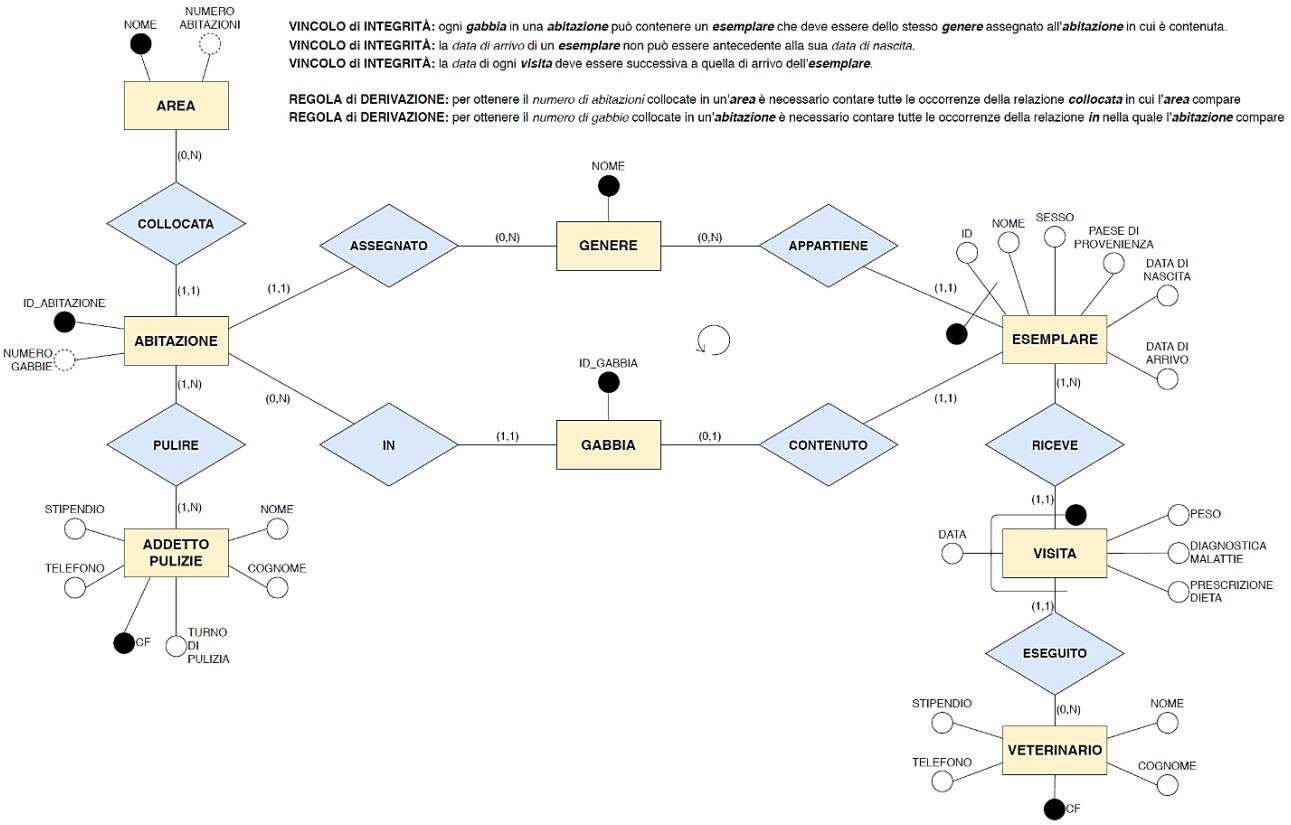
- **Operazioni più frequenti (a cadenza giornaliera)**
  - Aggiunta/rimozione ESEMPLARE
  - Aggiunta/rimozione di un ESEMPLARE ad una GABBIA
  - Aggiunta nuova VISITA veterinaria (scrittura nello "storico")
- **Operazioni meno frequenti**
  - Modifiche assegnazioni ADDETTO_PULIZIE
  - Aggiunta/rimozione ADDETTO_PULIZIE e VETERINARIO
  - Modifica della struttura gerarchica AREA/ABITAZIONE/GABBIA

Operazioni di lettura

- **Operazioni più frequenti (a cadenza giornaliera)**
  - Informazioni su ESEMPLARE
    - attributi
    - collocazione: le entità GABBIA, ABITAZIONE, AREA verranno coinvolte nelle operazioni di lettura
  - Informazioni sulla VISITA (si ipotizza che, prima di effettuare una VISITA, un VETERINARIO consulti quelle precedenti)
- **Operazioni meno frequenti**
  - Informazioni sugli attributi dell'ADDETTO_PULIZIE
  - Informazioni sugli attributi del VETERINARIO

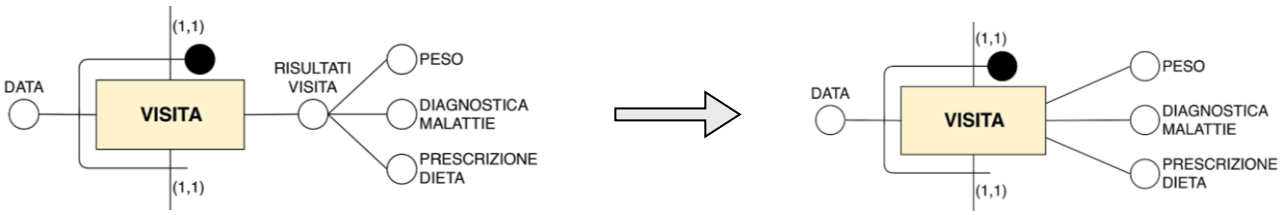
Ristrutturazione dello schema ER

Segue lo schema ER ristrutturato e le relative modifiche effettuate.



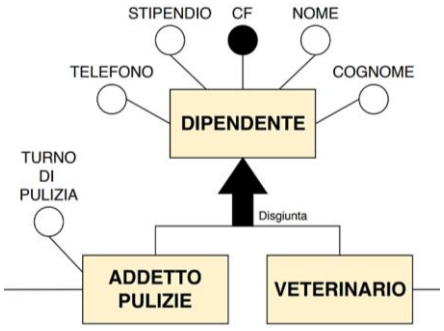
Rimozione attributi multipli

Attributo multiplo risultati visita (entità visita)



Rimozione generalizzazioni

Generalizzazione addetto pulizie – veterinario, totale disgiunta



In fase di analisi delle generalizzazioni è stato valutato di mantenere entrambi i figli e rimuovere l'entità genitore.

I motivi che hanno portato a questa scelta sono:

- La generalizzazione è totale.
- Non risulta mai necessario rappresentare un DIPENDENTE che non siano un ADDETTO_PULIZIE o VETERINARIO.
- Si suppone siano più frequenti le operazioni di lettura riguardanti gli attributi specifici (di ADDETTO_PULIZIE e VETERINARIO) che quelli generici.
- Mantenere l'entità DIPENDENTE avrebbe comportato la presenza di un attributo il cui scopo sarebbe stato quello di indicare se un determinato DIPENDENTE fosse un ADDETTO_PULIZIE o VETERINARIO. Inoltre, l'attributo turno_di_pulizia sarebbe stato sempre presente per ogni DIPENDENTE, anche per VETERINARIO (a cui non serve).
- Mantenere l'entità DIPENDENTE avrebbe reso più difficoltoso il controllo e la gestione delle inconsistenze (ad es. un ADDETTO_PULIZIE che visita un ESEMPLARE).

Analisi delle ridondanze

Entità GENERE (ridondanza su ESEMPLARE-ABITAZIONE)

Possibile inconsistenza: un'abitazione con genere assegnato x ha gabbie con esemplari di genere y.

Ovvero: "*se si ha un'**abitazione** x in relazione **assegnato** con il **genere** y mentre esiste un **esemplare** di **genere** diverso da y **contenuto** in una **gabbia** che è in relazione **in** con x, si ha un'inconsistenza".*

Considerazioni:

- L'attributo genere non può essere rimosso dall'entità ESEMPLARE in quanto chiave (chiave multipla genere-ID).
- La ricerca di quale GENERE sia assegnato a una determinata ABITAZIONE risulta più veloce nel caso in cui l'ABITAZIONE si trovi in relazione assegnato con l'entità GENERE (sulla base della tabella degli accessi riportata successivamente).
- Non c'è necessità di modificare l'attributo genere nell'entità ESEMPLARE (se non in seguito a un inserimento errato, quindi, operazione molto rara).
- In riferimento a un futuro schema relazionale, in cui la relazione assegnato verrà rappresentata con la presenza dell'attributo genere in ABITAZIONE:
    - alla sua modifica sarà necessario gestire correttamente la riassegnazione degli esemplari (tutti) che non soddisfano più il vincolo di eguaglianza di genere.
    - si suppone che il numero di istanze presenti nel database dell'entità ESEMPLARE siano molto superiori al numero di istanze di ABITAZIONE; di conseguenza, la ridondanza dell'attributo non causa un eccessivo spreco di memoria secondaria.

Conclusioni: in seguito alle considerazioni sopra elencate è stato deciso di mantenere la ridondanza.

Si consideri ora l'operazione di recuperare l'informazione relativa al genere assegnato ad una determinata abitazione, valutiamo i costi nel caso di ridondanza rimossa o mantenuta.

Caso relazione ASSEGNATO mantenuta

Per recuperare l'informazione relativa al genere assegnato ad una determinata ABITAZIONE sarà necessario leggere l'attributo "nome" dell'entità GENERE con cui essa si trova in relazione ASSEGNATO.

Costrutto	Concetto	Tipo	n° accessi	Note
E	Abitazione	read	1	trovo il record
R	Assegnato	read	1	trovo il riferimento al genere
E	Genere	read	1	trovo il record

Caso relazione ASSEGNATO rimossa
Per recuperare l'informazione relativa al genere assegnato ad una determinata abitazione devo leggere l'attributo "genere" dell'entità ESEMPLARE in relazione contenuto con una (indifferente quale) GABBIA in relazione IN con l'abitazione di riferimento.

Costrutto	Concetto	Tipo	n° accessi	Note
E	Abitazione	read	1	trovo il record
R	In	read	1	trovo una gabbia contenuta
E	Gabbia	read	1	trovo il record
R	Contenuto	read	1	trovo il riferimento all'esemplare
E	Esemplare	read	1	trovo il record

Risulta evidente che mantenendo la relazione ASSEGNATO tra le entità ABITAZIONE e GENERE il numero di operazioni è inferiore (3 al posto di 5).

Attributo numero_abitazioni
L'attributo numero_abitazioni (attributo derivato) è ridondante in quanto è possibile calcolarlo contando quante volte una determinata AREA è in relazione collocata con delle abitazioni.
Si considera comunque opportuno mantenere l'attributo derivato (che dovrà essere aggiornato ad ogni aggiunta/rimozione di un'abitazione ad un'AREA) per abbattere il tempo di recupero di quest'informazione.

Consideriamo quindi l'operazione di recuperare l'informazione relativa al numero di abitazioni contenute in una determinata AREA.

Caso attributo numero_abitazioni mantenuto
Per recuperare l'informazione voluta, basterà semplicemente leggere l'attributo numero_abitazioni.

Costrutto	Concetto	Tipo	n° accessi	Note
E	AREA	read	1	trovo il record e leggo l'attributo

Caso attributo numero_abitazioni rimosso
Nel caso in cui non si mantenga l'attributo derivato sarà necessario eseguirne il calcolo ogni qual volta che risulterà necessario recuperare l'informazione.

Costrutto	Concetto	Tipo	n° accessi	Note
E	AREA	read	1	trovo il record
R	COLLOCATA	read	10	trovo le abitazioni in relazione
E	ABITAZIONE	read	10	trovo le abitazioni

(NB: il valore 10 fa riferimento alla tabella dei volumi)

Operazione di aggiornamento dell'attributo derivato
L'operazione dell'attributo derivato consiste nel recupero dell'informazione più una semplice operazione di scrittura.

Costrutto	Concetto	Tipo	n° accessi	Note
E	AREA	read	1	trovo il record
R	COLLOCATA	read	10	trovo le abitazioni in relazione
E	ABITAZIONE	read	10	trovo le abitazioni
E	AREA	write	1	aggiorno l'attributo

(NB: il valore 10 fa riferimento alla tabella dei volumi)

Vengono fatte alcune considerazioni:

1. nel caso in cui si eseguano molte più letture dell'attributo derivato che modifiche della struttura delle aree e abitazioni, risulta molto più vantaggioso mantenere l'attributo derivato. Se ad esempio in media si hanno 1000 letture dell'attributo derivato ogni aggiornamento dell'attributo, avremmo un costo pari a:
   - 1000*1+22 = 1022 con l'attributo derivato
   - 1000*21 = 21000 senza l'attributo derivato

   con l'attributo derivato il vantaggio è: 1 – (1022/21000) = 95% degli accessi in meno
2. nel caso in cui, invece, si eseguono relativamente poche letture dell'attributo derivato tra una modifica e l'altra dell'organizzazione aree-abitazioni risulta comunque vantaggioso mantenere l'attributo derivato, ma in percentuale minore. Se si hanno 5 letture dell'attributo derivato ogni aggiornamento dell'attributo, avremmo un costo pari a:
   - 5*1+22 = 27 con l'attributo derivato
   - 5*21 = 105 senza l'attributo derivato

   con l'attributo derivato il vantaggio è di: 1 – (27/105) = 74% degli accessi in meno
3. il caso finale (opposto del primo) è la situazione in cui il numero di variazioni della struttura di aree e abitazioni supera di molto il numero di volte in cui si necessita di conoscere il numero di abitazioni presenti in un'area. Se, ad esempio, si ha 1 lettura dell'attributo ogni 250 modifiche, avremmo un costo pari a:
   - 1*1+250*22 = 5501 con l'attributo derivato
   - 250*21 = 5250 senza l'attributo derivato

   con l'attributo derivato il numero di accessi è maggiore del 5% (= 5501/5250). Il valore aggiornato dell'attributo, nella maggior parte dei casi, non viene letto.

Nel nostro caso è conveniente mantenere l'attributo derivato. Questo perché, nel caso reale, è molto più probabile che vengano lette informazioni riguardo le abitazioni, piuttosto che siano effettuate modifiche e spostamenti (punto 1).

Attributo numero_gabbie

Le considerazioni riguardo l'attributo derivato numero_gabbie sono molto simili a quelle relative all'attributo numero_abitazioni. Questo perché, nel nostro caso, è più probabile che vengano spostati animali dentro le gabbie, piuttosto che le gabbie stesse. Dunque vi saranno più letture che update e, come nel caso precedente, è vantaggioso mantenere l'attributo derivato.

Schema relazionale

Segue il modello relazionale.

AREA (nome, numero_abitazioni)
NOT NULL: numero_abitazioni

ABITAZIONE (ID, genere, numero_gabbie, area)
ABITAZIONE (genere) → GENERE (nome)
ABITAZIONE (area) → AREA (nome)
NOT NULL: genere, numero_gabbie, area

GABBIA (ID, abitazione)
GABBIA (abitazione) → ABITAZIONE (ID)
NOT NULL: abitazione

ESEMPLARE (ID, genere, nome, sesso, paese_provenienza, data_nascita, data_arrivo, gabbia)
ESEMPLARE (genere) → GENERE (nome)
ESEMPLARE (gabbia) → GABBIA (ID)
NOT NULL: nome, sesso, data_arrivo, gabbia
UNIQUE: gabbia

GENERE (nome)

ADDETTO PULIZIE (CF, nome, cognome, stipendio, telefono, turno_pulizia)
NOT NULL: nome, cognome, stipendio, turno_pulizia

PULIRE (addetto_pulizie, abitazione)
PULIRE (addetto_pulizie) → ADDETTO_PULIZIE(CF)
PULIRE (abitazione) → ABITAZIONE (ID)

VETERINARIO (CF, nome, cognome, stipendio, telefono)
NOT NULL: nome, cognome, stipendio

VISITA (veterinario, esemplare_id, esemplare_gen, data, peso, diagnostica, dieta)
VISITA (veterinario) → VETERINARIO (CF)
VISITA (esemplare_id) → ESEMPLARE (ID)
VISITA (esemplare_genere) → ESEMPLARE (genere)
NOT NULL: peso, diagnostica, dieta

4. Progettazione fisica e implementazione

Conclusa la progettazione logica vi è la fase di progettazione del database. Questa ha portato a una serie di decisioni relative alla fase di implementazione, come: tipi di dati da utilizzare per i vari attributi, individuazione dei trigger necessari per eseguire i controlli sui vincoli di integrità e sul calcolo degli attributi derivati, scelte relative agli indici, etc.

Per evitare di descrivere ogni riga di codice e ogni scelta, anche banale, seguirà una lista di scelte meno ovvie e qualche esempio rappresentativo per ogni fase.

DBMS

Il database è stato implementato utilizzando il DBMS ad oggetti PostgreSQL. La macchina su cui è stato implementato e su cui sono stati eseguiti i test è composta da:

- **CPU:** i7 9700K 5Ghz
- **RAM:** 2x8Gb Corsair Vengeance 3400Mhz
- **Memoria secondaria:** Samsung 970 EVO Plus
- **Sistema operativo:** Windows 10 x64

Il codice SQL è fornito in allegato alla relazione insieme a una copia importabile del DB già popolato.

Definizione delle tabelle

Durante la definizione delle tabelle sono state fatte alcune scelte relative ai tipi dei dati da utilizzare per rappresentare le informazioni relative agli attributi. Più nello specifico è stato scelto di rappresentare:

- i nomi e cognomi con variabili di tipo ***varchar(32)***
- gli id con il tipo di dato ***oid*** fornito da PostgreSQL
- il sesso degli esemplari con un ***char*** che può assumere solo valore 'M' o 'F'
  - con relativo check per controllarne la correttezza: `check(sesso IN ( 'F' , 'M' ))`
- gli attributi turno di pulizia, diagnostica e dieta con ***varchar(1024)***.
- le date utilizzando il tipo ***date*** già fornito da PostgreSQL
- i CF dei dipendenti con ***char(16)*** con relativo check sulla lunghezza (per semplicità non viene effettuato un controllo sulla correttezza del CF)
- attributi come stipendio, numero_gabbie e numero_abitazioni con interi con relativo check per controllare che siano positivi

Segue un esempio di creazione di tabella per l'entità ESEMPLARE.

```
create table Esemplare(
    id                  oid,
    genere              varchar(32),
    nome                varchar(32) not null,
    sesso               varchar(1) check(sesso IN ( 'F' , 'M' )) not null,
    paese_provenienza   varchar(32) not null,
    data_nascita        date,
    data_arrivo         date not null,
    gabbia              oid unique not null,

    constraint pk_Esemplare primary key(id,genere),
    constraint fk_genere_Esemplare_Genere foreign key (genere) references Genere(nome)
        on delete restrict
        on update cascade,
    constraint fk_gabbia_Esemplare_Gabbia foreign key (gabbia) references Gabbia(id)
        on delete restrict
        on update cascade
);
```

Note: è stato scelto di vietare la cancellazione di un genere (tramite **on delete restrict** sul vincolo di chiave esterna) se nel database sono presenti esemplari che ne fanno parte; analogamente non è possibile eliminare una gabbia se contiene un animale. Scelte simili sono state fatte per aree, abitazioni, visite, etc.

Notazione nomi dei vincoli
Per chiarezza si illustra brevemente il criterio di nomina dei vincoli:

- **pk** e **fk** stanno a indicare rispettivamente PrimaryKey e ForeignKey

Nel caso di vincolo di chiave primaria, dopo l'identificatore pk viene indicato a quale tabella fa riferimento.

Nel caso di vincolo di chiave esterna la sintassi di nomina è la seguente:
fk_<attributo_coinvolto>_<tabella_del_vincolo>_<tabella_dell'attributo>

Esempio di *on delete cascade* nella tabella visita
Nella tabella VISITA è stato scelto di utilizzare ***on delete cascade*** negli attributi di esemplare genere-ID (chiave esterna) in modo tale che in seguito alla rimozione dell'esemplare x, tutte le visite effettuate su di esso vengano rimosse dallo storico.

```
constraint fk_esemplare_gen_Visita_Genere
    foreign key (esemplare_gen,esemplare_id)
    references Esemplare(genere,id)

    on delete cascade -- se un esemplare viene rimosso, cancello tutte le visite eseguite su di esso
    on update cascade
```

Vincoli di integrità

Si riportano i vincoli di integrità da controllare:

1. Ogni gabbia in un'abitazione può contenere un esemplare che deve essere dello stesso genere assegnato all'abitazione in cui è contenuta.
2. La data di arrivo di un esemplare non può essere antecedente alla sua data di nascita.
3. La data di ogni visita deve essere successiva a quella di arrivo dell'esemplare

Segue a pagina successiva la fase relativa alla definizione e implementazione dei trigger necessari per il controllo dei vincoli di integrità in tabella.

Definizione dei trigger e funzioni SQL relativi ai vincoli di integrità

La tabella illustra i controlli da effettuare in relazione ai vincoli di integrità sopra illustrati.

ID	Controllo da eseguire
1	All'aggiunta **(INSERT/UPDATE)** di un esemplare ad una gabbia bisogna controllare che l'abitazione in cui essa sia contenuta abbia il genere assegnato corretto (ovvero uguale). *("vincolo di genere")*
2	Alla modifica (spostamento) **(UPDATE)** di una gabbia in una abitazione, bisogna controllare che il genere dell'animale in essa contenuto combaci con quello assegnato alla nuova abitazione di destinazione; similmente al controllo precedente. Nota: non serve eseguire il check sull'inserimento perché è necessario prima aggiungere una gabbia e poi assegnargli un animale; di conseguenza, non è possibile assegnare una gabbia errata alla sua aggiunta in quanto sono sempre vuote durante la creazione.
3	All'aggiunta **(INSERT/UPDATE)** di un esemplare bisogna controllare che la data di arrivo sia successiva alla sua data nascita.
4	All'aggiunta **(INSERT/UPDATE)** di una visita bisogna controllare che la data della visita sia successiva della data di arrivo dell'esemplare.
5	Alla modifica di un genere assegnato **(UPDATE)** ad un'abitazione, bisogna controllare che non vengano violati i vincoli di genere (in riferimento all'1). Nota: non serve il check sull'insert perché non è possibile inserire un'abitazione già con delle gabbie (non c'è il rischio che queste violino il vincolo di genere perché vengono aggiunte e controllate successivamente).
6	Alla modifica della data di arrivo di un esemplare bisogna controllare che questa sia coerente con le date delle visite: una visita non può essere stata effettuata prima che un esemplare sia arrivato nello zoo.

Segue il codice della creazione dei trigger necessari per eseguire i controlli appena elencati.

```
create trigger aggiunta_modifica_esemplare -- per il controllo n°1,3,6
before insert or update of data_arrivo,data_nascita,genere,gabbia on Esemplare
for each row
execute procedure aggiunta_modifica_esemplare();

create trigger modifica_gabbia -- per il controllo n°2
before update of abitazione on Gabbia
for each row
execute procedure modifica_gabbia();

create trigger aggiunta_modifica_visita -- per il controllo n°4
before insert or update of data on Visita
for each row
execute procedure aggiunta_modifica_visita();

create trigger modifica_genere_abitazione -- per il controllo n°5
before update of genere on Abitazione
for each row
execute procedure modifica_genere_abitazione();
```

Verrà ora illustrato solo il funzionamento del primo trigger (con relativa funzione) in quanto è il più complesso.

Trigger aggiunta_modifica_esemplare

Il trigger aggiunta_modifica_esemplare causa l'esecuzione di una procedura che si assicura che i controlli n°1, 3 e 6 elencati nella tabella vengano passati con successo. Più nello specifico la procedura è chiamata ogni qual volta che il vincolo di integrità rischia di non essere rispettato, ovvero quando:

- Viene inserito un nuovo esemplare
- Vengono aggiornati i campi ***data_arrivo, data_nascita, genere*** e ***gabbia***
  (gli altri non rischiano di violare alcun vincolo)

La funzione che viene chiamata e che esegue i tre controlli è riportata nella pagina seguente.

```
create or replace function aggiunta_modifica_esemplare()
returns trigger as $$ begin
-- questa prima fase esegue il primo controllo della tab. --
In questa fase controllo che il genere dell'esemplare che sto aggiungendo/spostando sia lo stesso
dell'abitazione di destinazione (che ottengo andando a guardare dov'è collocata la gabbia)

    perform *
    from(select    A.genere -- ottengo il genere assegnato all'abitazione contenente la gabbia.
         from      Abitazione A
         where     A.id IN(select    G.abitazione -- ottengo l'id dell'abit. della gabbia in cui
                             from    Gabbia G        sto inserendo l'esemplare
                            where    G.id = new.gabbia)) genere_ok
    where new.genere = genere_ok.genere;

-- questa seconda fase esegue il terzo controllo della tab. (nel caso il n°1 abbia avuto successo) --
Un semplice controllo sulle date, nel caso la condizione risulti VERA (e quindi le date non sono congrue)
viene lanciata un'eccezione (diversa in base al tipo di operazione).

    if found then
        if(new.data_arrivo <= new.data_nascita) then -- dopo aver controllato il vincolo 1, controlliamo il
                                                        vincolo 3 (la coerenza delle date)
            if(TG_OP = 'UPDATE') then
               raise exception 'Operazione di UPDATE non consentita! La modifica delle date ha portato a
                                delle incongruenze! Vincolo da rispettare: la data di nascita deve essere
                                antecedente o uguale alla data di arrivo';
            elseif(TG_OP = 'INSERT') then
               raise exception 'Operazione di INSERT non consentita! L'esemplare possiede delle incongruenze
                                sulle date! Vincolo da rispettare: la data di nascita deve essere antecedente
                                o uguale alla data di arrivo';
            end if;
        end if; -- end if del controllo sulle date

        -- questa terza fase esegue il sesto controllo della tab. (nel caso il n°3 abbia avuto successo)
        Il perform va alla ricerca di visite la cui data di esecuzione risulterebbe antecedente alla nuova
        data d'arrivo dell'esemplare, se ne vengono trovate viene lanciata un'eccezione

        perform *
        from    Visita V
        where   V.esemplare_id = new.id and V.esemplare_gen = new.genere and V.data < new.data_arrivo;

        if found then
            raise exception 'Operazione di UPDATE non consentita! La modifica della data di arrivo ha causato
                             un'incongruenza: ci sono visite effettuate prima della nuova data di arrivo
                             dell'esemplare ma non si può aver visitato un esemplare prima che questo sia
                             arrivato nello zoo.';
        end if; -- end if del controllo sulle visite
        return new; -- tutti i controlli sono andati a buon fine, ritorna NEW
    end if; -- se il primo controllo non è andato a buon fine: eccezione sul genere

    if(TG_OP = 'UPDATE') then
        if(new.genere = old.genere) then
            raise exception 'Operazione di UPDATE non consentita! La gabbia in cui si vuole spostare
                             l'esemplare è contenuta in un abitazione il cui genere assegnato differisce
                             da quello dell'esemplare';
        elseif(new.gabbia = old.gabbia) then
            raise exception 'Operazione di UPDATE non consentita! Il nuovo genere assegnato all'esemplare
                             non concide con quello assegnato all'abitazione in cui è contenuta la sua
                             gabbia';
        end if;
        raise exception 'Operazione di UPDATE non consentita!';
    elseif(TG_OP = 'INSERT') then
       raise exception 'Operazione di INSERT non consentita! La gabbia in cui si vuole inserire l'esemplare è
                    contenuta in un abitazione il cui genere assegnato differisce da quello dell'esemplare';
    end if;
end; $$ language plpgsql;
```

Attributi derivati

Oltre a vincoli di integrità, il database presenta alcuni attributi derivati che devono essere aggiornati al verificarsi di determinati eventi. L'approccio è stato quello di definire dei trigger che eseguono una chiamata alle funzioni per il calcolo degli attributi derivati quando necessario. Si elencano gli attributi derivati.

Attributo derivato	Tabella di appartenenza	Significato
numero_gabbie	Abitazione	Indica il n° di gabbie con cui è in relazione IN
numero_abitazioni	Area	Indica in n° di abitazioni con cui è in relazione COLLOCATA

Definizione dei trigger e funzioni SQL relative agli attributi derivati

Si illustrano le regole di derivazione: data una abitazione x, la regola di derivazione dell'attributo derivato numero_gabbie consiste nel semplice conteggio del numero di gabbie in relazione IN (collocate al suo interno) con x. La regola per il secondo attributo derivato è analoga.

Risulta quindi necessario aggiornare l'attributo **numero_gabbie** ogni qual volta venga eseguita una delle seguenti operazioni:

- **INSERIMENTO** di un record nella tabella GABBIA → *numero_gabbie*++
- **ELIMINAZIONE** di un record dalla tabella GABBIA → *numero_gabbie*--
- **UPDATE** del campo abitazione nella tabella GABBIA: consiste nello spostamento di una gabbia, quindi bisogna ricalcolare l'attributo per l'abitazione di partenza (*numero_gabbie*--) e per l'abitazione di destinazione (*numero_gabbie*++) solo dopo aver controllato che lo spostamento sia consentito, ovvero che non violi il vincolo di genere.

Per l'attributo numero_abitazioni si è seguito lo stesso approccio, segue l'illustrazione del primo trigger e della relativa funzione SQL:

Definizione del trigger per l'aggiornamento di numero_gabbie

```
create trigger aggiorna_numero_gabbie
after insert or delete or update of abitazione on Gabbia
for each row
execute procedure aggiorna_numero_gabbie();
```

Funzione SQL per l'aggiornamento di numero_gabbie

```
create or replace function aggiorna_numero_gabbie()
returns trigger as $$ begin
-- disattivazione temporanea del trigger che vieta le modifiche dell'attributo derivato --
è stato implementato un trigger che vieta all'utente di eseguire degli update manuali sull'attributo derivato
per evitare che inserisca valori inconsistenti. Questo trigger va disabilitato temporaneamente durante
l'aggiornamento (eseguito da codice, quindi, non manuale -> CONSENTITO) del relativo attributo derivato.

    ALTER TABLE Abitazione DISABLE TRIGGER deny_modifica_manuale_numero_gabbie;

    update Abitazione set numero_gabbie = (select  count(*)
                                           from   Gabbia G
                                           where  G.abitazione = new.abitazione)
    where  id = new.abitazione; -- update dell'attributo derivato della nuova tupla/tupla aggiornata

    update Abitazione set numero_gabbie = (select  count(*)
                                           from   Gabbia G
                                           where  G.abitazione = old.abitazione)
    where  id = old.abitazione; -- update dell'attributo derivato della vecchia tupla (nel caso di UPDATE)

-- riattivazione del trigger dopo aver aggiornato l'attributo derivato --

    ALTER TABLE Abitazione ENABLE TRIGGER deny_modifica_manuale_numero_gabbie;
    return new;

end; $$ language plpgsql;
```

Il trigger della funzione **aggiorna_numero_gabbie** è **deny_modifica_manuale_numero_gabbie**; si occupa di risolvere il problema di negare la modifica manuale dell'attributo numero_gabbie in quanto attributo derivato. Una modifica manuale causerebbe inconsistenze. Segue il codice del trigger e della relativa funzione.

Definizione del trigger per evitare modifiche manuali dell'attributo derivato

```
create trigger deny_modifica_manuale_numero_gabbie
before update of numero_gabbie on Abitazione
for each row
execute procedure deny_modifica_manuale_numero_gabbie();
```

Relativa funzione SQL:

```
create or replace function deny_modifica_manuale_numero_gabbie()
returns trigger as $$
begin

-- l'utente viene notificato che l'update violerebbe un vincolo di integrità, l'op. viene abortita --
    if(new.numero_gabbie != old.numero_gabbie) then
        raise exception 'MODIFICA DI UN ATTRIBUTO DERIVATO: Il numero di gabbie contenute in un'abitazione è
                         un attributo derivato e quindi non può essere modificato manualmente! Verrà
                         reimpostato al valore corretto!';
    end if;
        return new;

end;
$$ language plpgsql;
```

Un ultimo problema relativo all'attributo derivato numero_gabbie è il seguente: alla creazione di un'abitazione, l'attributo deve avere valore uguale a 0, in quanto l'abitazione non contiene gabbie. Bisogna quindi far sì che l'utente non possa inserire valori diversi da zero durante la creazione, o permetterglielo ma correggendo il record e notificarglielo (via log ad esempio). Nel nostro caso è stato seguito il secondo approccio. Seguono trigger e relativa funzione.

Definizione del trigger:

```
create trigger set_default_numero_gabbie
before insert on Abitazione
for each row
execute procedure set_default_numero_gabbie();
```

Definizione della funzione SQL:

```
create or replace function set_default_numero_gabbie()
returns trigger as $$
begin

if(new.numero_abitazioni != 0) then
    new.numero_gabbie := 0;
    raise warning 'Il numero di gabbie è stato impostato a 0 in quanto il valore presente nella query di
                   INSERT non era valido';
    return new;
end if;
return new;

end;
$$ language plpgsql;
```

Esempio

Inserendo la seguente tupla: `insert into Abitazione values (123456, 'Leone', 5, 'Area 1');`
il database restituisce il messaggio presente nel codice (di tipo warning, quindi non interrompe l'operazione di INSERT) avvisando l'utente che l'inserimento è andato a buon fine, ma che il valore dell'attributo derivato è stato corretto.
La tupla inserita realmente nel database risulta quindi: `(123456, 'Leone', 0, 'Area 1')`

Analogamente per l'attributo derivato numero_abitazioni, sono stati implementati gli stessi trigger e funzioni SQL appena illustrate.

Implementazioni degli indici

Si è ritenuto utile implementare alcuni indici al fine di ottimizzare le operazioni effettuate con maggiore frequenza. Segue una lista degli indici, con relative considerazioni e test sulle performance.

È bene ricordare che PostgreSQL implementa in automatico gli indici su ogni attributo con il parametro *unique* (quindi anche tutte le chiavi primarie).

Indice su ESEMPLARE.genere

In relazione alle considerazioni effettuate in precedenza (tabelle dei volumi, delle operazioni etc.) si ipotizza che le operazioni di lettura relative a informazioni sugli esemplari siano più frequenti di quelle di modifica. Si è scelto quindi di implementare un indice sull'attributo genere di ESEMPLARE, al fine di velocizzarne la ricerca.

Codice SQL:

```
create index esemplare_genere_index on Esemplare(genere)
```

Si consideri ora la seguente query:

```
select *
from   Esemplare E
where  E.genere = 'Elefante' and E.nome = 'Jazzy'
```

Mediante l'utilizzo dei comandi *explain analyze* forniti da PostgreSQL è possibile analizzare il tempo di esecuzione della query. Risulta evidente come, in presenza di un indice (B-tree) sull'attributo genere, la ricerca sia più veloce.

	Planning Time	Execution Time
SENZA	0.051 ms	0.259 ms
CON	0.055 ms	0.056 ms

(i risultati sono la media di 10 esecuzioni della query)

Indici su ESEMPLARE.nome ed ESEMPLARE.id

Si è ritenuto opportuno creare degli indici analoghi anche sui due attributi nome ed id, poiché si ipotizza che operazioni di ricerca per nome e id siano molto frequenti nel quotidiano utilizzo del database.

Indici su nome e cognome di ADDETTO_PULIZIE e VETERINARIO

Si è deciso di implementare degli indici sugli attributi nome e cognome di ADDETTO_PULIZIE e VETERINARIO, in quanto si ipotizza che le ricerche relative al personale possano avvenire utilizzando questi attributi (oltre alle ricerche per CF, di cui è già presente un indice, in quanto chiave primaria). Tuttavia, in fase di analisi delle performance, si è notato che il DBMS non utilizza gli indici, ma esegue comunque la ricerca tramite scansione lineare. A seguito di consultazioni del manuale è emerso che il planner di PostgreSQL, prima di eseguire una query, valuta i possibili approcci alla ricerca, scegliendo quello migliore. In questo caso, quindi, risulta più veloce la scansione lineare (evidentemente per la presenza di poche tuple nella tabella).

Indici relativi alle operazioni sul controllo del vincolo di genere

Si è ritenuto opportuno valutare l'ipotetica implementazione di indici che possano velocizzare le operazioni di controllo del vincolo di genere. Si consideri ora la query facente parte della funzione **aggiunta_modifica_esemplare** relativa al trigger che si occupa del controllo del vincolo di genere.

```
perform *
  from(select   A.genere -- ottengo il genere assegnato all'abitazione contenente la gabbia.
       from     Abitazione A
       where    A.id IN(select    G.abitazione -- ottengo l'id dell'abititazione della gabbia
                          from    Gabbia G        in cui sto inserendo l'esemplare
                         where    G.id = new.gabbia)) genere_ok
  where new.genere = genere_ok.genere;
```

Come si può notare leggendo il codice, le operazioni effettuate risultano le seguenti:

- ricerche sulla tabella GABBIA sull'attributo id
- ricerche sulla tabella ABITAZIONE sull'attributo id
- un'ultima operazione di ricerca sulla tabella ABITAZIONE sull'attributo genere (ininfluente)

Gli indici che migliorerebbero le prestazioni di questa operazione (indice su GABBIA.id e ABITAZIONE.id) sono già presenti in quanto, come già specificato, PostgreSQL crea in automatico indici per attributi di tipo *unique*.

Indice su esemplare_id di VISITA

Infine, si è ritenuto opportuno creare un indice per velocizzare le ricerche relative alle visite effettuate. Si è supposto che le ricerche sulle visite vengano spesso effettuate utilizzando l'id dell'esemplare (piuttosto che la chiave primaria formata da quattro attributi).

	Planning Time	**Execution Time**
SENZA	0.056 ms	46.298 ms
CON	0.177 ms	0.019 ms

(i risultati sono la media di 10 esecuzioni della query)

Viste

Si è deciso di implementare alcune viste considerate utili nel comune utilizzo del database. Segue un esempio.

Vista GABBIA.id, ABITAZIONE.genere, AREA.nome

Segue il codice per l'implementazione.

```
create view info_gabbia as
select gabbia.id, abitazione.id, abitazione.genere, abitazione.area
from gabbia join abitazione on abitazione.id = gabbia.abitazione;
```

La vista è utile per ottenere velocemente informazioni sul genere di animale presente nella gabbia (o eventualmente quello consentito, nel caso in cui la gabbia sia vuota) e sulla sua posizione all'interno dello zoo.

5. Popolazione e analisi dei dati in R

Successivamente all'implementazione del database, tramite la libreria *RPostgreSQL*, il DB è stato popolato con dati fittizi al fine di eseguire dei test e delle analisi.

Il codice R

Vengono riportati alcuni frammenti di codice utilizzati nella popolazione del DB. Sono stati usati come riferimento i valori presenti nella tabella dei volumi.

Popolazione tabella AREA

```
# POPOLAZIONE DB - AREA

# nome
area.nome = c()

for ( i in 1:10 ){
  area.nome[i] = paste("Area", as.character(i))
}

# numero_abitazioni (inizialmente un vettore di zeri lungo 10)
area.numero_abitazioni = rep(0, 10)

# Creo il DF
area_df = data.frame(nome = area.nome,
                     numero_abitazioni = area.numero_abitazioni)

# Popolo il database
dbWriteTable(con,
             name=c("public","area"),
             value=area_df,
             append = TRUE,
             row.names=FALSE)
```

Popolazione tabella ABITAZIONE

```
# POPOLAZIONE DB - ABITAZIONE

# id
# Creo un vettore con 100 id di abitazioni
abitazione.nome = c()
for ( i in 1:100 ){
  abitazione.nome[i] = 1110000 + i     # per convenzione le abitazioni iniziano con '111'
}

# numero_gabbie (inizialmente un vettore di zeri lungo 100)
abitazione.numero_gabbie = rep(0, 100)

# genere
# Ottengo i generi interrogando il db, poi creo il vettore
vettore_genere = dbGetQuery(con, "select nome from genere")
vettore_genere = vettore_genere$nome

# area
# Ottengo le aree interrogando il db, poi creo il vettore
vettore_area = dbGetQuery(con, "select nome from area")
vettore_area = vettore_area$nome

# Creo il DF
abitazione_df = data.frame(id = abitazione.nome,
                           numero_gabbie = abitazione.numero_gabbie,
                           genere = sample(vettore_genere, 100, replace = TRUE),  # 100 generi rnd con ripet
                           area = sample(vettore_area, 100, replace = TRUE))      # 100 aree rnd con ripet

# Popolo il database
dbWriteTable(con,
             name=c("public","abitazione"),
             value=abitazione_df,
             append = TRUE,
             row.names=FALSE)
```

Popolazione tabella ADDETTO_PULIZIE

```
# POPOLAZIONE DB - ADDETTO PULIZIE

# Per i nomi e cognomi degli impiegati, utilizzo la libreria "randomNames"
library(randomNames)

addetto_pulizie.nome = randomNames(100, which.names="first")    # nome
addetto_pulizie.cognome = randomNames(100, which.names="last")  # cognome

# CF
# algoritmo che crea dei CF univoci, di 16 cifre
id = 4440000000000
indice = 1
addetto_pulizie.cf = c()

for ( n in addetto_pulizie.nome ){
  parte_numerica = id + 1     # per convenzione i CF iniziano con '444'
  id = id + 1
  parte_letterale = toupper(substr(n, 1, 3))
  if (nchar(parte_letterale) == 3) {
    CF = paste(parte_numerica, parte_letterale, sep="")
  }
  else {
    CF = paste(parte_numerica, "SAD", sep="")
  }
  addetto_pulizie.cf[indice] = CF
  indice = indice + 1
}

# stipendio
vettore_stipendio_base = rep(1200, 100)
vettore_stipendio_agg = 0:500
vettore_da_sommare = sample(vettore_stipendio_agg, 100, replace = TRUE)
addetto_pulizie.stipendio = vettore_stipendio_base + vettore_da_sommare

# telefono
vettore_telefono_prefisso = rep(345000000, 100)
vettore_telefono_random = 1:999999
addetto_pulizie.telefono = vettore_telefono_prefisso + sample(vettore_telefono_random, 100, replace = FALSE)

# turno_pulizia
giorno_inizio = c("Lunedi", "Martedi", "Mercoledi")
giorno_fine = c("Giovedi", "Venerdi")
ora_inizio = c("08:00","09:00","10:00")
ora_fine = c("15:00","16:00","17:00","18:00")

addetto_pulizie.turno_pulizia = c()

for ( i in 1:100 ){
  p1 = sample(giorno_inizio, 1)
  p2 = sample(giorno_fine, 1)
  p3 = sample(ora_inizio, 1)
  p4 = sample(ora_fine, 1)

  temp_turno = paste(p1,"-",p2, " ", p3,"-",p4, sep="")
  addetto_pulizie.turno_pulizia[i] = temp_turno
}

# Creo il DF
addetto_pulizie_df = data.frame(cf = addetto_pulizie.cf,
                                nome = addetto_pulizie.nome,
                                cognome = addetto_pulizie.cognome,
                                stipendio = addetto_pulizie.stipendio,
                                telefono = addetto_pulizie.telefono,
                                turno_pulizia = addetto_pulizie.turno_pulizia)

# Popolo il DB
dbWriteTable(con,
             name=c("public","addetto_pulizie"),
             value=addetto_pulizie_df,
             append = TRUE,
             row.names=FALSE)
```

Analisi dei dati in R

Successivamente alla popolazione del database, sono state eseguite alcune semplici analisi sui dati. Di seguito vengono riportati alcuni passaggi fondamentali e risultati ottenuti.

Interrogazione DB

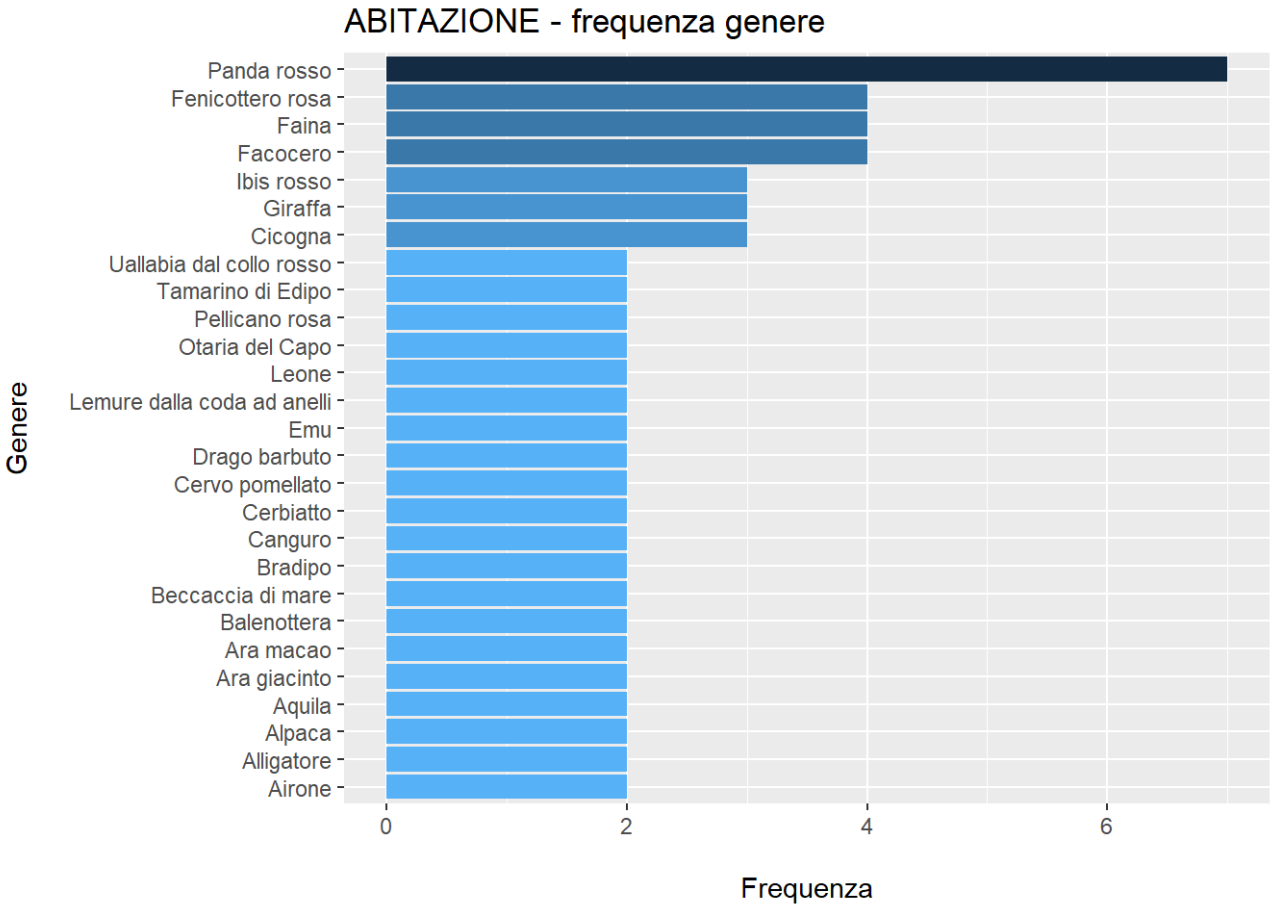
Vengono così ottenuti i dataframe su cui eseguire le successive analisi. Inoltre, viene effettuato un controllo sulla correttezza del numero di entry.

```
area_df = dbGetQuery(con, "select * from area")
paste("Numero aree:", length(area_df$nome))

abitazione_df = dbGetQuery(con, "select * from abitazione")
paste("Numero abitazioni:", length(abitazione_df$id))

gabbia_df = dbGetQuery(con, "select * from gabbia")
paste("Numero gabbie:", length(gabbia_df$id))

genere_df = dbGetQuery(con, "select * from genere")
paste("Numero generi:", length(genere_df$nome))

esemplare_df = dbGetQuery(con, "select * from esemplare")
paste("Numero esemplari:", length(esemplare_df$nome))

addetto_pulizie_df = dbGetQuery(con, "select * from addetto_pulizie")
paste("Numero addetti pulizie:", length(addetto_pulizie_df$cf))

pulire_df = dbGetQuery(con, "select * from pulire")
paste("Numero istanze pulire:", length(pulire_df$addetto_pulizie))

veterinario_df = dbGetQuery(con, "select * from veterinario")
paste("Numero veterinari:", length(veterinario_df$cf))

visite_df = dbGetQuery(con, "select * from visita")
paste("Numero visite:", length(visite_df$data))
```

```
[1] "Numero aree: 10"
[1] "Numero abitazioni: 100"
[1] "Numero gabbie: 5000"
[1] "Numero generi: 80"
[1] "Numero esemplari: 4500"
[1] "Numero addetti pulizie: 100"
[1] "Numero istanze pulire: 300"
[1] "Numero veterinari: 20"
[1] "Numero visite: 270000"
```

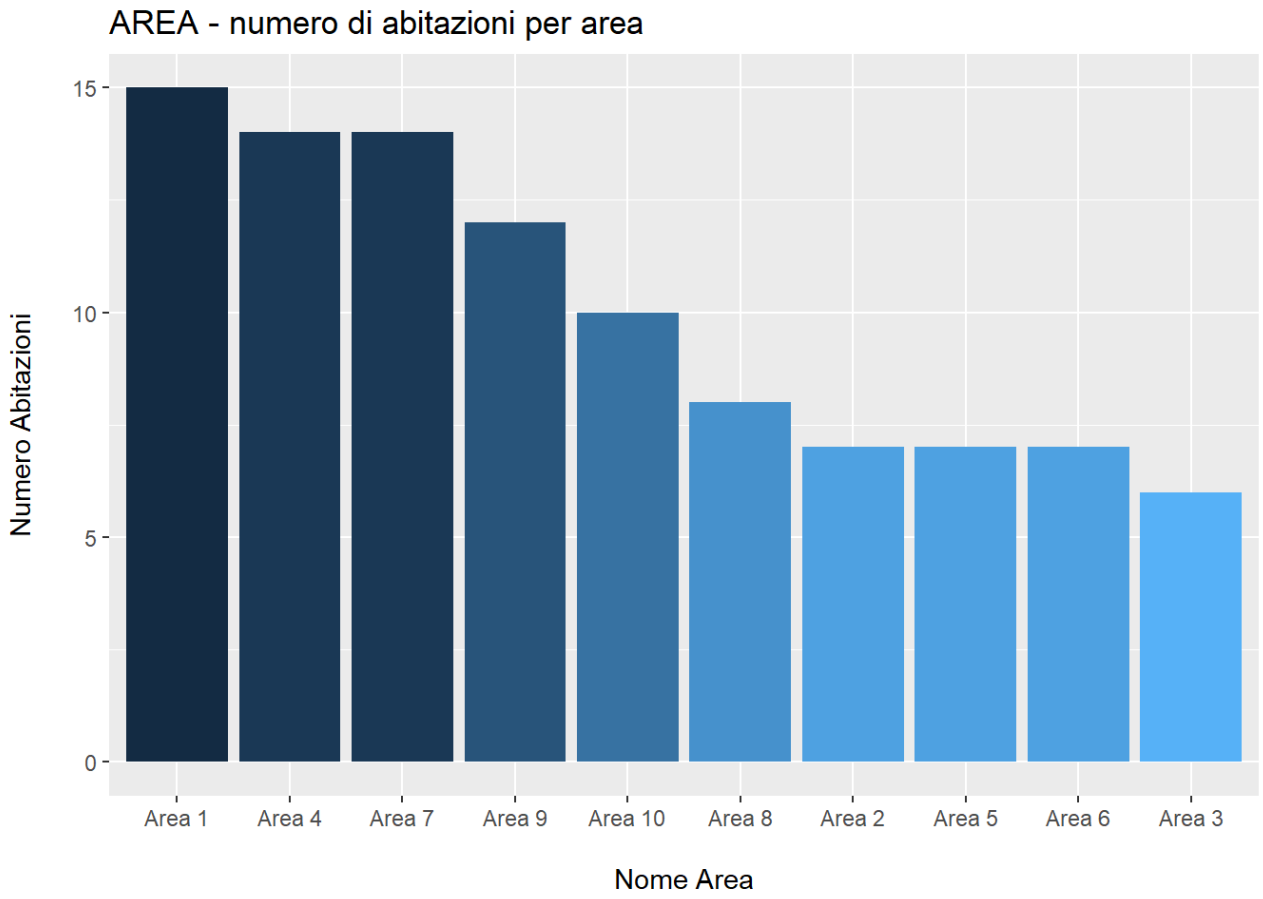
Abitazioni

```
# ABITAZIONE - frequenza genere
# Generi più presente nelle abitazioni, non considero i generi presenti solo 1 volta
temp_df = abitazione_df %>%
  count(genere) %>%
  filter(n > 1)

ggplot(data=temp_df, aes(y=n, x=reorder(genere, n), fill=-n)) +
  geom_bar(stat="identity", show.legend = FALSE) +
  labs(title="ABITAZIONE - frequenza genere", x="Genere\n", y = "\nFrequenza") +
  coord_flip()
```



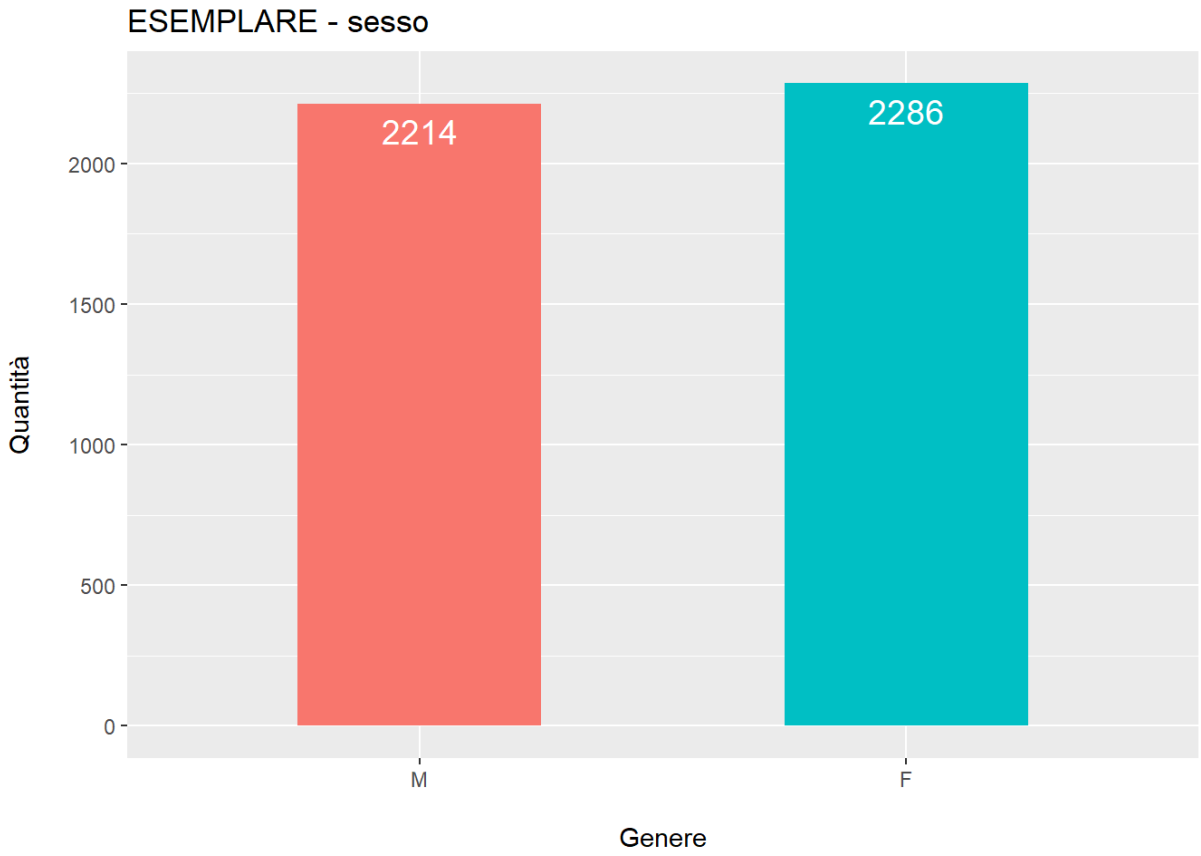
Aree

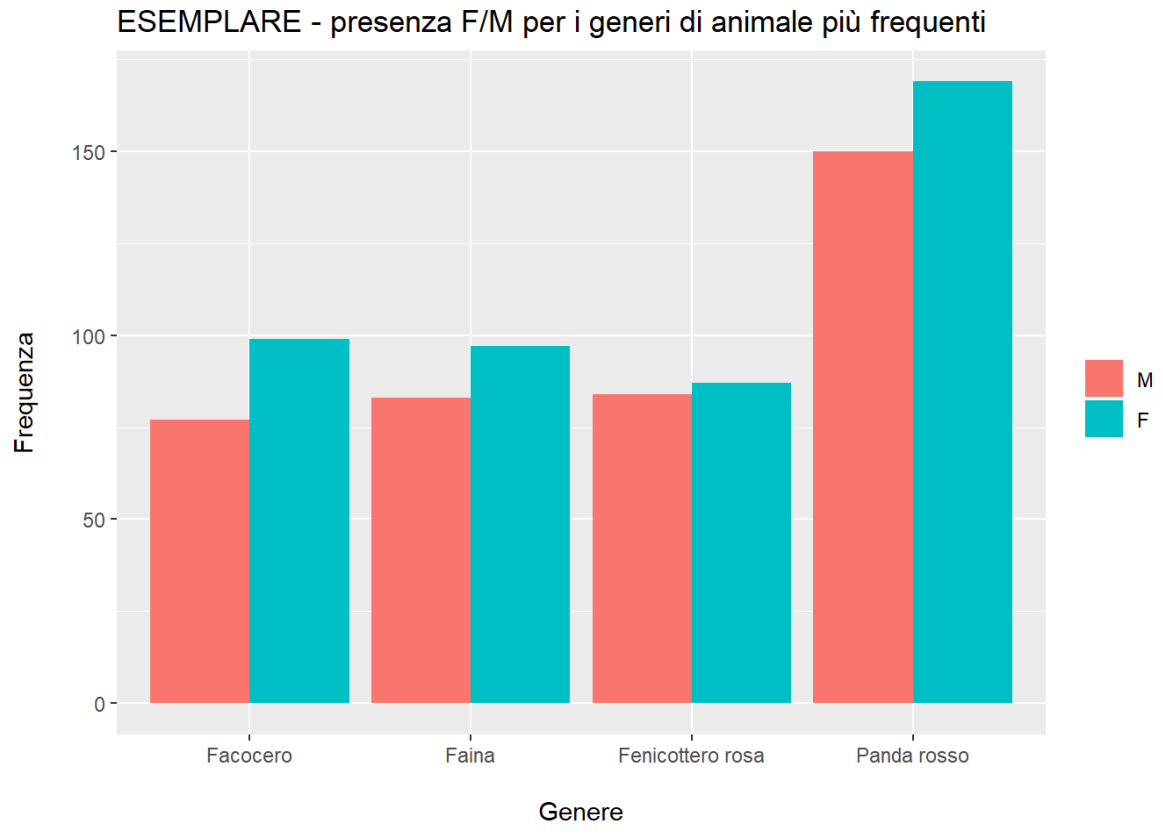
```
# AREA - frequenza abitazioni
temp_df = abitazione_df %>%
  count(area)

ggplot(data=temp_df, aes(y=n, x=reorder(area, -n), fill=-n)) +
  geom_bar(stat="identity", show.legend = FALSE) +
  labs(title="AREA - numero di abitazioni per area", x="\nNome Area", y = "Numero Abitazioni\n")
```



Esemplari

Analisi sul sesso degli esemplari. Come si nota dai grafici, il numero di esemplari maschili e femminili è bilanciato all'interno dello zoo.





Addetti pulizie e veterinari

Segue il calcolo dell'istogramma e della densità degli stipendi degli addetti alle pulizie e dei veterinari.