Relazione Basi di Dati

2019-2020

Andrea Mansi - 137857
Daniele Passabì - 137758

Indice

Introduzione

<<introduzione al progetto da aggiungere>>

Requisiti

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

Glossario dei termini

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

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 deve 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 unico all'interno del genere di appartenenza
  - data di arrivo nello zoo e 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 e 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 destinata ad un determinato genere di animale
  - possiede un insieme di gabbie ad essa assegnate (ciascuna contenente un esemplare)
  - ha un addetto alla pulizia delle gabbie
- Il sistema deve permettere di:
  - gestire (aggiungere, modificare e rimuovere) le abitazioni dello zoo
  - tenere conto del numero di gabbie assegnate a ciascuna abitazione
  - cambiare (per ciascuna abitazione) il genere assegnato

Gabbia

- Ciascuna gabbia è identificata da un ID univoco.
- Ciascuna gabbia contiene un solo animale (il cui genere deve 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:
    - aggiungere e 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 e 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

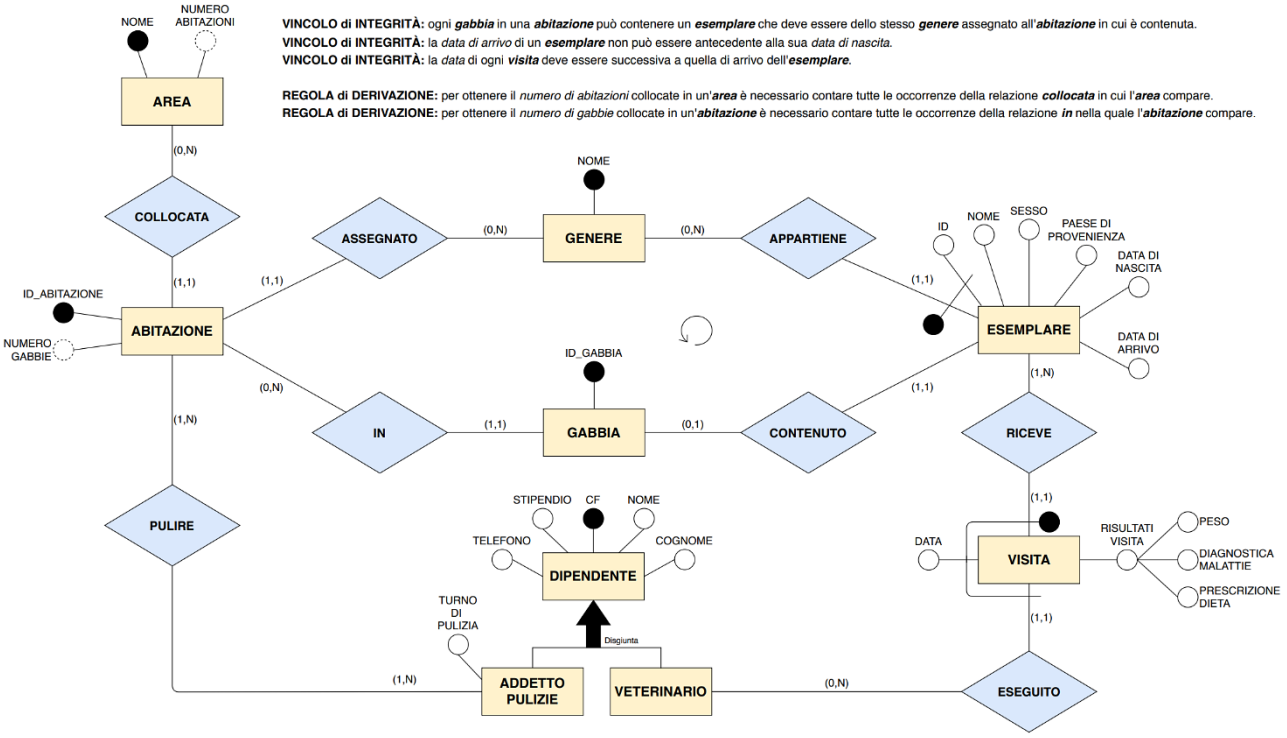
- Ogni visita è caratterizzata da:
    - data, esemplare visitato e veterinario che ha effettuato la visita
    - rilevamento del peso
    - diagnostica di eventuali malattie
    - prescrizione tipologia di dieta
- Il sistema deve permettere di:
    - inserire, modificare e rimuovere nuove visite una volta che vengono effettuate
    - 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 una 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 i seguenti due vincoli relativi 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 ancora che sia arrivato nello zoo.

2. Progettazione concettuale

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



<……scrivere qualcosa……..>

3. Progettazione logica

Seguono le fasi di progettazione logica.

Analisi delle prestazioni

Come prima fase si è partiti effettuando un'analisi delle prestazioni basandosi su tabella dei volumi e operazioni. Il risultato dell'analisi è stato poi utilizzato come base di scelta 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
NB: si ipotizza che il numero di animali nello zoo rimanga pressoché costante, inoltre, il numero di visite è relativo a un periodo di 10 anni (metà vita del database).	

<< TODO pulire 2 righe>>

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.

Frequenza operazioni

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

Operazioni di scrittura:

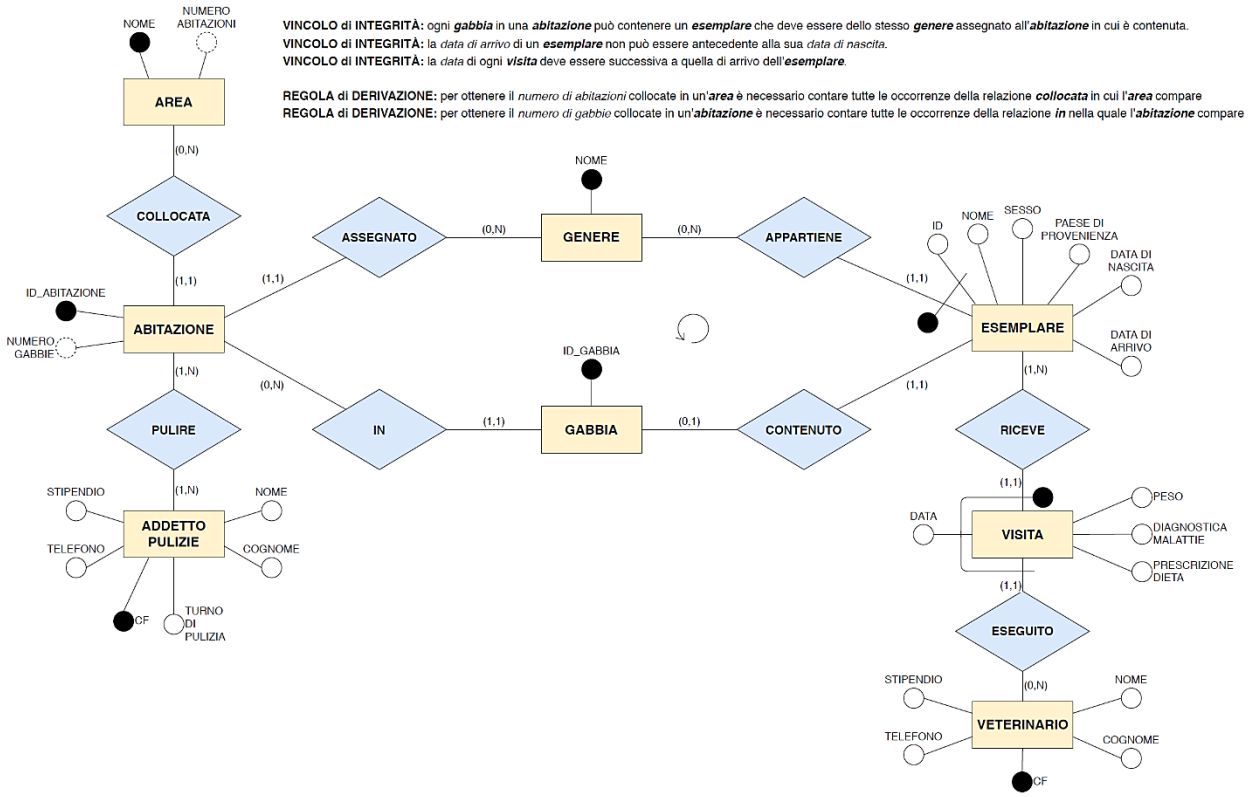
- **Operazioni più frequenti (a cadenza giornaliera)**
  - Aggiunta/rimozione di un esemplare ad una gabbia
  - Aggiunta/rimozione esemplari
  - Aggiunta nuove visite veterinarie (scrittura nello “storico”)
- **Operazioni meno frequenti**
  - Modifiche delle assegnazioni degli addetti alle pulizie (si suppone siano modifiche fatte di rado)
  - Aggiunta/rimozione dipendenti
  - Modifica della struttura gerarchica area/abitazione/gabbia (modificati molto raramente)

Operazioni di lettura:

- **Operazioni più frequenti (a cadenza giornaliera)**
  - Informazioni sugli esemplari (molto frequenti)
    - Attributi
    - Collocazione: le entità gabbia, abitazione, area verranno coinvolte nelle operazioni di lettura
  - Informazioni sulle visite (si ipotizza che prima di effettuare una visita, un veterinario consulti quelle precedenti)
- **Operazioni meno frequenti**
  - Informazioni sugli attributi dei dipendenti

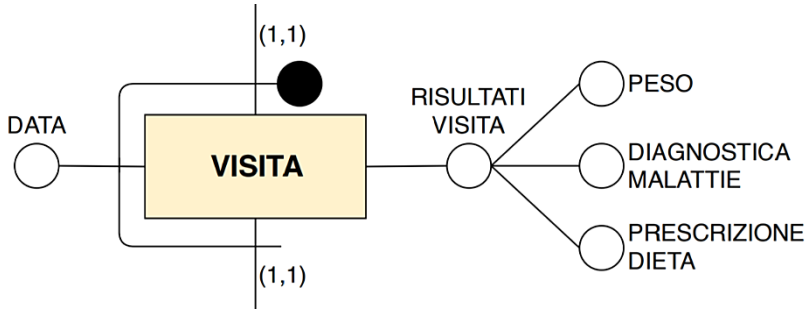
Ristrutturazione dello schema ER

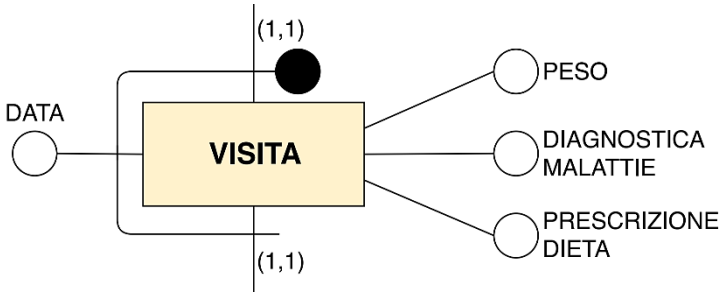
Segue lo schema ER ristrutturato e le relative modifiche effettuate.



Rimozione attributi multipli

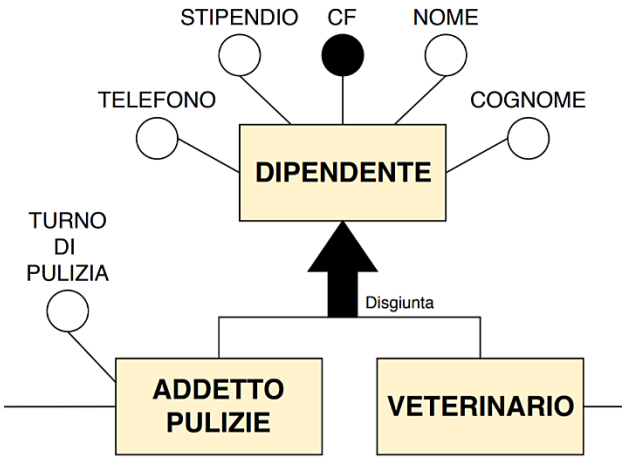
Attributo multiplo risultati visita (entità visita)





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 dipendenti che non siano addetti alle pulizie o veterinari
- Si suppone siano più frequenti le operazioni di lettura riguardanti gli attributi specifici (di addetto alle 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 alle pulizie o un veterinario
    - Inoltre, l'attributo turno di pulizia sarebbe stato sempre presente per ogni dipendente, anche per i veterinari (a cui non serve)
- Mantenere l'entità dipendente avrebbe reso più difficoltoso il controllo e la gestione delle inconsistenze (ad es. un addetto alle pulizie che visita un esemplare

Analisi delle ridondanze

Attributo genere (ridondanza su ESEMPLARE-ABITAZIONE)

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

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'attributo genere venga mantenuto (nell'entità ABITAZIONE)
- Non c'è necessità di modificare l'attributo genere nell'entità ESEMPLARE (se non in seguito a un inserimento errato)
- Nel caso in cui si modifichi l'attributo genere nell'entità ABITAZIONE 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: è stato deciso di mantenere la ridondanza sull'attributo "genere". Sarà necessario quindi:

- Gestire correttamente il cambiamento dell'attributo genere nell'entità ABITAZIONE: cambiare il genere assegnato ad una abitazione crea inconsistenza con gli esemplari ospitati nelle sue gabbie.
- Ad ogni assegnamento di un esemplare a una gabbia bisogna effettuare un controllo sul vincolo di integrità: "ogni gabbia assegnata a una abitazione deve contenere lo stesso genere presente nell'attributo GENERE dell'abitazione"

Attributi numero abitazioni e numero gabbie

L'attributo numero di 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.

Le considerazioni sovrastanti risultano analoghe per l'attributo "numero gabbie".

Relazione assegnato

La relazione "assegnato" è ridondante in quanto risulta possibile ricavare l'informazione relativa al genere di un'abitazione controllando il genere degli animali contenuti nelle sue gabbie.
Si considera opportuno mantenere la relazione assegnato in quanto l'operazione di lettura di quale genere sia assegnato ad una determinata abitazione risulta effettuato ogniqualvolta si voglia inserire/spostare un esemplare in una gabbia (operazione eseguita frequentemente).

<< aggiungere numero letture/scritture (accessi) con e senza ridondanze >>

Schema relazionale

4. Progettazione fisica e implementazione

Conclusa la progettazione logica si è passato con il progettare il database. Questa fase 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 solo di quelle scelte che si considerano meno ovvie e di qualche esempio rappresentativo per ogni fase.

DBMS

Il database è stato implementato utilizzando il DBMS ad oggetti PostgreSQL. Segue la descrizione delle varie fasi con relativi commenti. 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 2°:** Samsung 970 EVO Plus
- **SO:** 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*** già 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 di dato ***date*** già fornito da PostgreSQL
- i CF dei dipendenti con ***char(16)*** (non viene effettuato un controllo sulla sua correttezza)
- 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) affinché nel database siano 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 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>

on delete casade nella tabella visita:
Nella tabella VISITA è stato scelto di utilizzare **on delete cascade** nell'attributo esemplare (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_Visista_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
```

È possibile visionare il codice sorgente in cui nella prima parte sono presenti le definizioni di tutte le tabelle con relativi commenti.

Vincoli di integrità

Si riportano i vincoli di integrità da controllare:

ID	Descrizione
1	ogni ***gabbia*** in una ***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 n°1 **NB:** 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 data arrivo >= data nascita.
4	All'aggiunta **(INSERT/UPDATE)** di una visita bisogna controllare che data visita > data arrivo esemplare.
5	Alla modifica di un genere assegnato **(UPDATE)** ad un'abitazione, bisogna controllare che non vengano violati i vincoli di genere (in riferimento al n°1) **NB:** non serve il check sull'insert perché non è possibile inserire una abitazione già con delle gabbie (non c'è 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 (e relativa funzione) in quanto 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:

- inseriamo un nuovo esemplare
- aggiorniamo i campi ***data_arrivo, data_nascita, genere*** e ***gabbia*** (in quanto gli altri non rischiano di violare alcun vincolo).

La funzione che viene chiamata e che esegue i tre controlli è la 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 visistato 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. Iniziamo elencando 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 funzioni SQL relativi agli attributi derivati

Iniziamo illustrando 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 identico approccio, seguirà quindi l'illustrazione solo del primo trigger e relativa funzione per evitare di illustrare una seconda implementazione pedissequa.

Definizione del trigger per l'aggiornamento del n° 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 del n° 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 citato precedentemente nell'illustrazione del codice della funzione **aggiorna_numero_gabbie** è il seguente: **deny_modifica_manuale_numero_gabbie** e 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 una 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 subito dopo e notificarglielo (via log ad esempio). Nel nostro caso è stato seguito il secondo approccio. Segue 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 successivamente a una modifica dell'attributo derivato al valore 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
<<…scelte…>>

Implementazione
<<…implementazione indici in postgresql…>

Analisi delle performance
<<… query speed with and without indexes comparison…>>

5. Popolazione e analisi dei dati in R
<<…intro…>>

Il codice R(??)
nb: ricordarsi di scrivere che la popolazione dei dati segue la tabella dei volumi

Analisi dei dati in R(??)