ALGORITMI E CALCOLATORI A.A. 2019/2020

PROGETTO DI ALGORITMI

# DATABASE MANAGEMENT

## 1. DESCRIZIONE DEL PROBLEMA

Un database permette di descrivere una serie di tabelle a partire dal significato e dal tipo delle informazioni che devono contenere, organizzate per colonna, e di popolare poi queste tabelle con delle informazioni puntuali, con un paradigma riga per riga (ogni riga è detta *record*). Particolarità di un database è la possibilità di collegare tra loro le tabelle, vincolando il contenuto di una specifica colonna di una tabella ai valori presenti in una colonna di un'altra tabella.

L'obiettivo del progetto è quello di costruire un motore basilare di gestione delle diverse tabelle, sia nella parte di creazione e definizione, sia in quella di riempimento dei dati e ricerca degli stessi.

La Sezione 4 elenca tutte le richieste per gruppi e singoli, mentre le Sezioni 5 e 6 riportano tutte le richieste extra per gruppi e singoli in base alle sessioni di consegna dell'elaborato.

## 2. CONSIDERAZIONI GENERALI

Tutta le funzionalità descritte in seguito dovranno essere sviluppate in modo da consentire all'utente di usare il programma più volte, ritrovando ogni volta lo stato del database come lasciato alla fine dell'ultima esecuzione in modo consistente. Non è mai consentito operare solo ed esclusivamente sui file: l'implementazione deve prevedere la strutturazione dei dati in memoria e fare uso dei file solo per la proprietà di consistenza dei dati.

## 3. SCADENZE PER LA CONSEGNA

Si indicano le tre scadenze, una per sessione, entro cui è prevista la consegna dell'elaborato. Non sarà ammessa la consegna in altra data né eccezioni sugli orari.

- Sessione Giugno/Luglio 2020: 6 Luglio 2020 ore 12.00
- Sessione Settembre 2020: 7 Settembre 2020 ore 12.00
- Sessione Gennaio/Febbraio 2021: 18 Gennaio 2021 ore 12.00

Le Sezioni 8 e 9 includono dettagli sulla valutazione.

# 4. RICHIESTE

Il Sistema è composto da un motore che gestisce la parte di descrizione delle tabelle ed il salvataggio dei dati, ed una interfaccia testuale che permette all'utente di interrogare il database secondo un estratto (semplificato) del linguaggio SQL.

Il motore deve essere in grado di gestire tabelle il cui numero di colonne è variabile (ma sempre almeno una). Ogni colonna è rappresentata da una serie indispensabile di informazioni: nome, tipo ~~e valore di default~~. Nella versione richiesta dal progetto, i tipi di dato sono limitati alla seguente lista: **int, float, char, text, date** e **time**. Dove **text** è da intendersi come un testo di lunghezza variabile, **date** è una data in un formato a scelta del programmatore che consenta di tener traccia di giorno, mese ed anno, e **time** è un ora, sempre in un formato a scelta, volto a memorizzare ora, minuti e secondi. Al programmatore, in ogni punto del programma, è richiesto verificare la congruenza dei dati inseriti con il tipo impostato. La tabella è inoltre provvista di altre informazioni di base: il suo nome, l'indicazione di quale sia la chiave primaria (cioè la colonna il cui valore identifica in modo univoco il singolo record e, per questo motivo, non può trovarsi due volte come valore). Si predisponga il motore per non accettare spazi sia nella definizione del nome tabella sia nel nome assegnato alle colonne. Nel caso in cui invece il progetto fosse già al punto di consentire tale pratica, verrà naturalmente accettata.

In termini applicativi, il motore deve fornire tutti gli strumenti atti alla creazione delle tabelle e la loro manipolazione (del contenuto o della struttura), a seconda delle varie richieste (per tutti, per i singoli e negli appelli successivi).

L'interfaccia provvederà a supportare l'interazione con il motore interno da parte dell'utente. Nelle sezioni seguenti verranno individuate tutte le operazioni che l'utente deve poter attivare, descrivendo anche come l'utente può richiederle attraverso l'interfaccia testuale. Ogni qualvolta l'operazione richiesta non prevede in modo chiaro una reportistica, si faccia sempre in modo che vi sia opportuna comunicazione della riuscita dell'operazione. La gestione degli errori e della loro reportistica, laddove non specificato in modo diretto, è demandata alla progettazione degli studenti e farà parte di opportuna valutazione. Si consenta infine di uscire dal programma con il comando QUIT().

## A. Definizione (creazione di una tabella)

Al fine di definire una tabella, l'utente deve poter ricorrere al seguente tipo di codice:

```
CREATE TABLE CUSTOMERS (
   ID   INT              NOT NULL,
   NAME TEXT     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  TEXT ,
   SALARY   FLOAT,
   PRIMARY KEY (ID)
);
```

Nell'esempio, viene creata una tabella il cui nome è CUSTOMERS, i capi sono ID (di tipo int), NAME (di tipo text), AGE (di tipo int), ADDRESS (di tipo text) e SALARY (di tipo float), in cui la chiave primaria è indicata nel campo ID e si definisce nei campi ID, NAME e AGE i campi che NON possono essere lasciati vuoti (NOT NULL). In mancanza di tale indicazione, il campo può essere lasciato vuoto, cioè non conterrà alcun valore. Nel caso in cui, dopo l'indicazione di un campo, venga aggiunta la parola chiave AUTO_INCREMENT, si intende che tale campo non dovrà mai essere specificato dall'utente in modo diretto ma, in maniera automatica, ad ogni inserimento si dovrà a definire tale campo come +1 rispetto all'ultimo usato (in senso assoluto). Si dia per scontato che il primo valore ammesso sia 1 e si faccia attenzione che non è ammesso riutilizzare valori già calcolati (anche se non più presenti nella tabella per via di operazione di cancellazione). Facendo riferimento all'esempio precedente:

```
CREATE TABLE CUSTOMERS (
   ID INT NOT NULL AUTO_INCREMENT,
   NAME TEXT     NOT NULL,
   AGE  INT            NOT NULL,
   ADDRESS  TEXT ,
   SALARY   FLOAT,
   PRIMARY KEY (ID)
);
```

Indica che il campo ID non verrà mai specificato in modo diretto durante la creazione di un nuovo record. Si faccia in modo che l'attributo AUTO_INCREMENT possa essere applicato solo a tipi numerici int.

### B. Cancellazione di una tabella

Una tabella creata può essere sempre rimossa dal sistema attraverso il comando DROP TABLE, come da esempio seguente:

```
DROP TABLE CUSTOMERS;
```

La cancellazione deve prevedere la rimozione dalla tabella e del suo contenuto dal sistema.

### C. Inserimento di nuovi record

Al fine di inserire un singolo record nella tabella, l'utente deve poter usare, da terminale, un codice simile a questo:

```
INSERT INTO CUSTOMERS (AGE, ADDRESS, NAME)
VALUES (20, "via Roma 10, Torino", "Francesco Rossi");
```

Dove il primo blocco di parentesi () specifica quali campi verranno impostati, mentre quelle che seguono il termine VALUES, specificano i valori da impostare. Si noti che l'ordine con cui vengono inseriti i campi del nuovo record risponde allo stesso ordine con cui sono indicati nel primo blocco (). Questo ordine è variabile, perciò il seguente codice deve produrre lo stesso risultato:

```
INSERT INTO CUSTOMERS (ADDRESS, AGE, NAME)
VALUES (“via Roma 10, Torino”, 20, “Francesco Rossi”);
```

In questa fase, ogni meccanismo di controllo della congruenza delle informazioni deve attivarsi, impedendo all’operazione di completarsi in caso di errori.

### D. Cancellazione di uno o più record

La possibilità di cancellare un record è attivata dal comando DELETE, con una sintassi simile alla seguente:

```
DELETE FROM CUSTOMERS WHERE NAME=”Francesco Rossi”;
```

Nell’esempio, facendo riferimento alla tabella create in precedenza, il comando cancella dalla tabella CUSTOMERS **tutti** i record che verificano la condizione per cui il campo NAME è uguale a Francesco Rossi. Infatti, l’unico modo per cancellare una singola riga, è quello di poter esplicitare una condizione (dopo il WHERE) che selezioni un solo record in modo univoco.

Nel caso in cui si volesse svuotare l’intera tabella, deve essere possibile farlo attraverso il seguente comando:

```
TRUNCATE TABLE CUSTOMERS;
```

### E. Aggiornamento di uno o più record

Al fine di completare le possibilità di manipolazione dei dati, sia possibile anche effettuare un aggiornamento dei dati contenuti in uno o più record. Si noti che tale alterazione vale per tutti i campi che non siano AUTO_INCREMENT. Inoltre, tutti gli aggiornamenti dei dati devono rispettare le regole di congruenza definite. Il comando per modificare i dati è il comando UPDATE. Nell’esempio:

```
UPDATE CUSTOMERS
SET SALARY = 12000
WHERE AGE = 20;
```

Le modifiche riguardano tutti i record che hanno il campo AGE impostato a 20. Si noti quindi che per modificare un singolo record, la selezione tramite WHERE deve essere fatta su un campo univoco.

A titolo di esempio si riporta un comando in cui più di un campo viene impostato:

```
UPDATE CUSTOMERS
SET SALARY = 12000, NAME = “Nuovo nome”
WHERE AGE = 20;
```

### F. Stampa a video del contenuto di una tabella (tramite selezione)

La stampa a video di una singola tabella avviene come risposta all’operatore di selezione SELECT. L’operatore infatti consente, nella sua forma completa, di selezionare, data una

tabella, quali colonne visualizzare e quali record. Se si vuole stampare tutti i record, la forma da utilizzare è:

```
SELECT * FROM CUSTOMERS;
```

Si noti che l'operatore * tra SELECT e FROM, permette la selezione di tutte le colonne che fanno parte della tabella. Deve essere possibile anche selezionare le colonne da visualizzare:

```
SELECT ID, AGE, SALARY  FROM CUSTOMERS;
```

E farlo usando un operatore di selezione:

```
SELECT ID, AGE, SALARY  FROM CUSTOMERS WHERE AGE = 20;
```

Nell'ultimo esempio, ci si aspetta di poter visualizzare tutti i record che rispondono al vincolo di avere la colonna AGE con valore pari a 20, visualizzando solo ID, AGE e SALARY.

### G. Ordinamento dei risultati

La stampa dei risultati deve essere ordinabile. Questo significa che l'utente, data la singola tabella, deve poter scegliere come ordinare i risultati in base ad una colonna e ad un ordinamento crescente (ASC) o decrescente (DESC). Nello specifico, facendo riferimento all'ultimo esempio della sezione precedente:

```
SELECT ID, AGE, SALARY  FROM CUSTOMERS WHERE AGE = 20
ORDER BY NAME DESC;
```

In questo caso, data la selezione condizionale (che non è obbligatoria), si richiede che i risultati siano ordinati per il campo NAME (si noti che non deve necessariamente essere tra i campi selezionati per la visualizzazione), ordinati in modo decrescente.

Sia la tabella riempita in questo modo:

| ID | NAME | AGE | ADDRESS | SALARY |
|---|---|---|---|---|
| 1 | Francesco Rossi | 20 | via Roma 10, Torino | 25000 |
| 3 | Marco Blue | 22 | via Pinerolo 72, Milano | 10000 |
| 12 | Pippo Cercato | 20 | Schillerstraße 5, Leipzig | 20000 |
| 23 | Enea Corinto | 30 | Via Stazione 8, Potenza | 12000 |

Il risultato dell'ordinamento di esempio è:

| ID | AGE | SALARY |
|---|---|---|
| 12 | 20 | 20000 |
| 1 | 20 | 25000 |

## H. Collegamento di diverse tabelle attraverso le chiavi esterne

Al fine di permettere il collegamento tra più tabelle, deve essere possibile definire delle **chiavi esterne**. Una chiave esterna è un campo della tabella che si sta creando che viene collegato ad un campo chiave di un'altra tabella (che deve risultare già creata). Questo collegamento comporta che sia possibile valorizzare tale campo solo il valore esiste in uno dei record della tabella di riferimento.

Si prende l'esempio precedente e si aggiunga una tabella:

```
CREATE TABLE COUNTRIES(
   ID INT NOT NULL AUTO_INCREMENT,
   NAME TEXT     NOT NULL,
   PRIMARY KEY (ID)
);
```

La tabella intende creare un elenco di nazioni, in modo da poter modificare la tabella CUSTOMERS nel seguente modo:

```
CREATE TABLE CUSTOMERS(
   ID INT NOT NULL AUTO_INCREMENT,
   NAME TEXT     NOT NULL,
   AGE  INT            NOT NULL,
   ADDRESS  TEXT ,
   COUNTRY_ID INT
   SALARY   FLOAT,
   PRIMARY KEY (ID)
   FOREIGN KEY (COUNTRY_ID) REFERENCES COUNTRIES (ID)
);
```

In questa nuova versione della tabella, il campo COUNTRY_ID ammette solo valori presenti nel campo ID della tabella COUNTRIES. Tutte le possibili incongruenze nella creazione della tabella e nell'inserimento dei record devono essere controllate.

# 5. RICHIESTE AGGIUNTIVE PER SINGOLI

## A. Sessione Settembre 2020

### i. Operatori di confronto diversi da =

Sia possibile effettuare operazioni di selezione tramite operatore WHERE (quindi per qualunque operazione in cui sia possibile usarlo), che non siano soltanto di uguaglianza (=), ma siano possibili anche:

- > (Greater than),
- < (Less than),
- >= (Greater than or equal),
- <= (Less than or equal),
- <> Not equal.

### ii. BETWEEN operator

Sia data la possibilità di usare, sempre per le operazioni di confronto, l'operatore BETWEEN che permette di specificare un intervallo di valori ammissibili. Facendo riferimento all'ultimo esempio della sezione 4.E:

```
SELECT ID, AGE, SALARY  FROM CUSTOMERS
WHERE AGE BETWEEN 20 AND 30;
```

L'operazione deve permettere di selezionare tutti i record i cui campo AGE è compreso tra 20 e 30.

### iii. Condizioni Multiple

Considerando il caso delle selezioni tramite operatore WHERE (quindi SELECT ma anche DELETE), consentire all'utente di scrivere condizioni multiple attraverso l'uso di operatori di AND e OR e NOT. Nel primo esempio:

```
SELECT ID, AGE, SALARY  FROM CUSTOMERS
WHERE AGE = 20 AND SALARY < 18000 OR NAME <> "Francesco Rossi";
```

La selezione avviene per tutti i record in cui AGE è 20, il SALARY è minore di 18000 oppure il NAME è diverso da Francesco Rossi.

Nel secondo esempio si illustra l'uso della NOT:

```
SELECT ID, AGE, SALARY  FROM CUSTOMERS
WHERE NOT AGE = 20 AND NOT SALARY <= 18000;
```

Consentendo di selezionare tutti i record in cui l'AGE non è 20 ed il salario più grande di 18000.

## B. Sessione Gennaio/Febbraio 2021

### i. LIKE operator e caratteri wildcard

Al fine di effettuare le ricerche in modo ampio, sia possibile definire l'operatore di similitudine LIKE. L'operatore fa uso di alcuni caratteri speciali (wildcard) che consentono di definire la similitudine. Nello specifico, si faccia riferimento ai seguenti esempi per i vari caratteri di speciali e all'uso della similitudine:

```
SELECT * FROM CUSTOMERS WHERE NAME                Inizia con F
```

```
LIKE ”F%”;
SELECT * FROM CUSTOMERS WHERE NAME LIKE ”%i”;
SELECT * FROM CUSTOMERS WHERE NAME LIKE ”% Ross_”;
```

Finisce con I

Finisce con “Ross” seguito da un solo carattere (es: Rossi, Rosso, Rossa, ecc)

### ii. Concatenamento di Tabelle

Sia possibile ampliare la definizione di selezione delle tabelle in modo da selezionare record provenienti da tabelle diverse se soddisfano una specifica condizione. Per comprendere meglio, siano le tabelle usate fino ad ora riempite con i seguenti dati:

| ID | NAME | AGE | ADDRESS | COUNTRY_ID | SALARY |
|---|---|---|---|---|---|
| 1 | Francesco Rossi | 20 | via Roma 10, Torino | 1 | 25000 |
| 3 | Marco Blue | 25 | via Pinerolo 72, Milano | 1 | 10000 |
| 5 | Till Lindemann | 47 | Schillerstraße 5, Leipzig | 3 | 90000000 |
| 9 | Dennis Lyxzén | 48 | Rådhustorget, 903, Umeå | 5 | 3700000 |
| 12 | Umberto Giardini | 42 | Strada Ponti 17, Bologna | 1 | 412000 |

| ID | NAME |
|---|---|
| 1 | Italia |
| 3 | Germania |
| 5 | Svezia |

Il concatenamento prevede che, selezionando da entrambe le tabelle, con opportune regole di ricerca, sia possibile visualizzare tutti i record della prima tabella con il nome della nazione al posto dell’id (rielaborando l’esempio in Sezione 4.G):

```
SELECT CUSTOMERS.ID, CUSTOMERS.NAME, CUSTOMERS.AGE, COUNTRIES.NAME
FROM CUSTOMERS, COUNTRIES
WHERE AGE > 20 AND CUSTOMERS.COUNTRY_ID = COUNTRIES.ID
ORDER BY CUSTOMERS.SALARY DESC;
```

Il risultato dovrà essere:

| CUSTOMERS.ID | CUSTOMERS.NAME | CUSTOMERS.AGE | COUNTRIES.NAME |
|---|---|---|---|
| 5 | Till Lindemann | 47 | Germania |
| 9 | Dennis Lyxzén | 48 | Svezia |
| 12 | Umberto Giardini | 42 | Italia |
| 3 | Marco Blue | 25 | Italia |

# 6. RICHIESTE AGGIUNTIVE PER GRUPPI

## A. Sessione Giugno/Luglio 2020

### i. Operatori di confronto diversi da =

Sia possibile effettuare operazioni di selezione tramite operatore WHERE (quindi per qualunque operazione in cui sia possibile usarlo), che non siano soltanto di uguaglianza (=), ma siano possibili anche:

- > (Greater than),
- < (Less than),
- >= (Greater than or equal),
- <= (Less than or equal),
- <> Not equal.

### ii. BETWEEN operator

Sia data la possibilità di usare, sempre per le operazioni di confronto, l'operatore BETWEEN che permette di specificare un intervallo di valori ammissibili. Facendo riferimento all'ultimo esempio della sezione 4.E:

```
SELECT ID, AGE, SALARY  FROM CUSTOMERS
WHERE AGE BETWEEN 20 AND 30;
```

L'operazione deve permettere di selezionare tutti i record i cui campo AGE è compreso tra 20 e 30.

## B. Sessione Settembre 2020

### i. Condizioni Multiple

Considerando il caso delle selezioni tramite operatore WHERE (quindi SELECT ma anche DELETE), consentire all'utente di scrivere condizioni multiple attraverso l'uso di operatori di AND e OR e NOT. Nel primo esempio:

```
SELECT ID, AGE, SALARY  FROM CUSTOMERS
WHERE AGE = 20 AND SALARY < 18000 OR NAME <> "Francesco Rossi";
```

La selezione avviene per tutti i record in cui AGE è 20, il SALARY è minore di 18000 oppure il NAME è diverso da Francesco Rossi.

Nel secondo esempio si illustra l'uso della NOT:

```
SELECT ID, AGE, SALARY  FROM CUSTOMERS
WHERE NOT AGE = 20 AND NOT SALARY <= 18000;
```

Consentendo di selezionare tutti i record in cui l'AGE non è 20 ed il salario più grande di 18000.

### ii. LIKE operator e caratteri wildcard

Al fine di effettuare le ricerche in modo ampio, sia possibile definire l'operatore di similitudine LIKE. L'operatore fa uso di alcuni caratteri speciali (wildcard) che consentono di definire la similitudine. Nello specifico, si faccia riferimento ai seguenti esempi per i vari caratteri di speciali e all'uso della similitudine:

```
SELECT * FROM CUSTOMERS WHERE NAME                Inizia con F
```

```
LIKE ”F%”;
SELECT * FROM CUSTOMERS WHERE NOT NAME
LIKE ”%i”;
SELECT * FROM CUSTOMERS WHERE NAME
LIKE ”% Ross_”;
```

(Esclusione se) Finisce con i
Finisce con “Ross” seguito da un solo carattere (es: Rossi, Rosso, Rossa, ecc)

## C. Sessione Gennaio/Febbraio 2021

### i. Concatenamento di Tabelle

Sia possibile ampliare la definizione di selezione delle tabelle in modo da selezionare record provenienti da tabelle diverse se soddisfano una specifica condizione. Per comprendere meglio, siano le tabelle usate fino ad ora riempite con i seguenti dati:

| ID | NAME | AGE | ADDRESS | COUNTRY_ID | SALARY |
|---|---|---|---|---|---|
| 1 | Francesco Rossi | 20 | via Roma 10, Torino | 1 | 25000 |
| 3 | Marco Blue | 25 | via Pinerolo 72, Milano | 1 | 10000 |
| 5 | Till Lindemann | 47 | Schillerstraße 5, Leipzig | 3 | 90000000 |
| 9 | Dennis Lyxzén | 48 | Rådhustorget, 903, Umeå | 5 | 3700000 |
| 12 | Umberto Giardini | 42 | Strada Ponti 17, Bologna | 1 | 412000 |

| ID | NAME |
|---|---|
| 1 | Italia |
| 3 | Germania |
| 5 | Svezia |

Il concatenamento prevede che, selezionando da entrambe le tabelle, con opportune regole di ricerca, sia possibile visualizzare tutti i record della prima tabella con il nome della nazione al posto dell’id (rielaborando l’esempio in Sezione 4.G):

```
SELECT CUSTOMERS.ID, CUSTOMERS.NAME, CUSTOMERS.AGE, COUNTRIES.NAME
FROM CUSTOMERS, COUNTRIES
WHERE AGE > 20 AND CUSTOMERS.COUNTRY_ID = COUNTRIES.ID
ORDER BY CUSTOMERS.SALARY DESC;
```

Il risultato dovrà essere:

| CUSTOMERS.ID | CUSTOMERS.NAME | CUSTOMERS.AGE | COUNTRIES.NAME |
|---|---|---|---|
| 5 | Till Lindemann | 47 | Germania |
| 9 | Dennis Lyxzén | 48 | Svezia |
| 12 | Umberto Giardini | 42 | Italia |
| 3 | Marco Blue | 25 | Italia |

### ii. Uso di ALIAS

La richiesta precedente introduce la problematica di dover visualizzare a video campi con nomi di difficile lettura. Si consenta perciò di definire l'operatore di alias AS che permetta di impostare etichette alternative nella visualizzazione di una selezione. A titolo di esempio, in riferimento al precedente punto:

```
SELECT CUSTOMERS.ID AS CustomerID, CUSTOMERS.NAME AS CustomerName,
CUSTOMERS.AGE AS CustomerAge, COUNTRIES.NAME AS CustomerCountry
FROM CUSTOMERS, COUNTRIES
WHERE AGE > 20 AND CUSTOMERS.COUNTRY_ID = COUNTRIES.ID
ORDER BY CUSTOMERS.SALARY DESC;
```

Consentirà di visualizzare il risultato della selezione con i nomi delle colonne impostati attraverso l'AS.

## 7. ISTRUZIONI DI CONSEGNA

I gruppi di lavoro possono essere composti da un minimo di 1 ad un massimo di 3 persone. Per sottomettere l'elaborato, occorre creare un archivio zip con il nome

Prog2020_<matricole>.zip

In cui il campo matricole deve essere sostituito con l'elenco (ordinato) delle matricole dei partecipati, separate dal carattere '_'. Il file dovrà essere caricarlo attraverso il portale della didattica entro le date di scadenza pubblicate ed indicate in **Sezione 2**.

L'archivio zip **deve** essere organizzato secondo le seguenti cartelle:

- *src*: contiene tutti i file sorgenti ed il file CMakeLists.txt per compilare il progetto con CLion in totale autonomia;
- *input*: contiene tutti i file di input utili;
- *doc*: contiene una relazione dettagliata (formato pdf) su (di almeno 4 pagine):
    - struttura del programma
    - algoritmi e strutture dati utilizzati
    - analisi dettagliata della complessità di tutti gli algoritmi sviluppati.

La creazione del file .zip che non rispetti le specifiche di consegna, influirà negativamente sulla valutazione.

## 8. REGOLE DI VALUTAZIONE

**In funzione della situazione di incertezza creata dalla situazione di emergenza sanitaria, al fine di minimizzare la necessità di spostamenti e di assembramenti, si è deciso di procedere, a seguito di ogni scadenza di consegna, alla valutazione dei progetti sottomessi senza diretto coinvolgimento degli studenti. Alla fine del processo di valutazione, verrà reso noto il voto del progetto, senza necessità di ulteriori passaggi**.

Per tutti i gruppi o singoli che non si dovessero riconoscere nel voto, sarà garantito l'accesso ad un orale aggiuntivo che, come da regole d'esame, prevederà un'ulteriore discussione del progetto, incluse domande di teoria, volto ad integrare il voto. Tale integrazione è da intendersi in termini migliorativi o peggiorativo del voto, in funzione delle conoscenze mostrate in fase di orale. **Non sarà possibile sottomettere un progetto in gruppo e richiedere un orale da singoli.** Sarà sempre possibile rifiutare il voto secondo le regole specificante nel metodo di valutazione.

**Il testo del progetto è valido fino all'ultima sessione di esame antecedente l'inizio di un nuovo corso (secondo semestre dell'A.A. 2020/21)**. Dopo tale data si dovrà fare riferimento alle regole decise per il corso in essere. Non saranno fatte eccezioni di nessun genere.

## 9. METODO DI VALUTAZIONE

Per l'elaborato, i criteri di valutazione comprendono:

- Rispetto delle specifiche;
- Funzionamento complessivo del programma;
- Algoritmi e strutture dati utilizzati;
- Flessibilità delle soluzioni adottate;
- Progettazione (es: organizzazione in classi, meccanismi di ereditarietà, ecc.);
- Gestione degli errori;
- Accuratezza nella generazione di un numero opportuno di casistiche di test (set di file di input per la verifica del programma);
- Aderenza al linguaggio di programmazione (es: uso della STL, const reference, ecc.).

Nel caso in cui il progetto non sia sufficiente (o il voto venga rifiutato), sarà richiesto di rivedere l'intero progetto e sottometterlo nuovamente in una sessione successiva, includendo tutte le richieste aggiuntive. La non sufficienza del progetto genera un decremento del voto finale massimo raggiungibile: ogni rifiuto diminuisce il voto massimo raggiungibile di 2 punti (es: progetto non sufficiente una volta sola, voto massimo all'appello successivo 28; due volte, voto massimo 26, ecc.).