

Elaborato di basi di dati – 2024/25

Partenopei Destiny Frontier

14 gennaio 2025

Autori:

Danilo Cioffi N46007095

Francesco Ardolino N46007168

Paolo Altucci N46007260

Università degli Studi di Napoli Federico II

Corso di Laurea in Ingegneria Informatica

A.A. 2024/2025

Indice

1	SPECIFICHE SUI DATI	3
1.1	Scelte di progetto	3
2	PROGETTAZIONE DELLA BASE DI DATI	4
2.1	PROGETTAZIONE CONCETTUALE	4
2.1.1	Modello ER	4
2.2	PROGETTAZIONE LOGICA	5
2.2.1	Fase di trasformazione	5
2.2.2	Fase di traduzione	5
2.3	PROGETTAZIONE FISICA	7
2.3.1	Creazione delle tabelle	7
2.3.2	Scrittura dei vincoli inter-relazionali	12
2.3.3	Creazione delle sequenze	14
2.3.4	Creazione degli indici	14
3	GESTIONE PROGETTO	15
3.1	Gestione della Concorrenza	15
3.2	Gestione dell’Affidabilità	15

4	OPERAZIONI SULLA BASE DATI	17
4.1	Query	17
4.2	Viste	19
4.3	Trigger	21
4.4	Stored procedure	25
5	GESTIONE DELLA SICUREZZA	29
6	CREAZIONE WEB APPLICATION	30
7	CLUSTERING DEI DATI DEI MEMBRI DELL'AGENZIA SPAZIALE	36
7.1	Specifiche sui cluster	36
7.2	Presentazione del Codice	37
7.2.1	Connessione al Database	37
7.2.2	Recupero dei Dati	38
7.2.3	Calcolo delle Metriche	38
7.2.4	Esportazione dei Risultati	38
7.3	Codice	38

1 SPECIFICHE SUI DATI

- **Informazioni generali:**

Si vuole progettare una base di dati per un'agenzia spaziale che contenga informazioni relative alle MISSIONI, ai MEMBRI DELL'EQUIPAGGIO, ai SENSORI, ai ROBOT, alle RILEVAZIONI effettuate dai sensori, alle possibili ANOMALIE dei sensori, agli INTERVENTI per la risoluzione delle anomalie e ai REPORT relativi allo stato di una missione.

- **Informazioni sulle missioni:**

Delle missioni devono essere conservate informazioni che ne indichino inizio e fine, l'obiettivo da portare a termine e lo stato in cui si trova la missione (IN CORSO, ANNULLATA, COMPLETATA).

- **Informazioni sui membri dell'equipaggio:**

Dei membri dell'equipaggio si vogliono memorizzare nome, cognome, ruolo ed un codice univoco all'interno del sistema.

- **Informazioni sui sensori:**

Dei sensori si vuole tenere traccia della posizione (coordinate tridimensionali), tipologia, dello stato operativo, data di installazione e data di ultimo controllo. Inoltre ogni sensore è identificato da un codice univoco all'interno del sistema.

- **Informazioni sui robot:**

Dei robot si vuole tenere traccia della tipologia e del loro ID univoco all'interno del sistema.

- **Informazioni sulle rilevazioni:**

Delle rilevazioni si vogliono memorizzare data e ora di misurazione ed il valore rilevato

- **Informazioni sulle anomalie:**

Delle anomalie si vuole tener traccia di data e ora, livello di priorità e l'evento a valle del quale si sono verificate.

- **Informazioni sugli interventi:**

Degli interventi bisogna memorizzare una descrizione, data di esecuzione ed esito, oltre che al codice univoco che li identifica all'interno del sistema.

- **Informazioni sui report:**

Con un report si vuole memorizzare una descrizione dello stato di una missione.

1.1 Scelte di progetto

Dall'analisi appena condotta si evince che il committente richiede che vengano gestite le seguenti informazioni:

- Sensori: viene inserito nel database e dal momento dell'installazione sulla luna può essere utilizzato in altre missioni, anche contemporaneamente
- Robot: una sua istanza può essere usato in più missioni, anche contemporaneamente

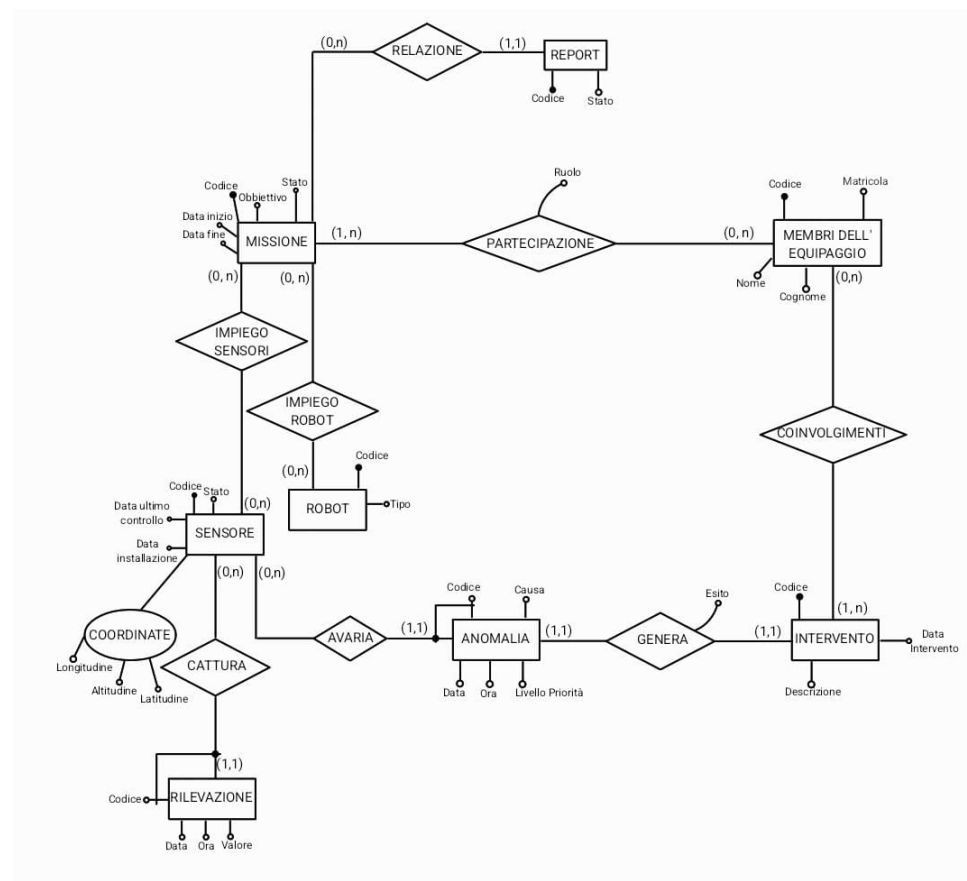
- **Membro:** un membro può partecipare a più missioni con ruoli diversi, in una missione può avere un solo ruolo
- **Anomalia, interventi:** quando un sensore genera un'anomalia viene generato un intervento la cui data dipende dal livello di priorità

2 PROGETTAZIONE DELLA BASE DI DATI

2.1 PROGETTAZIONE CONCETTUALE

La progettazione di una base dati procede secondo tre fasi distinte, la prima delle quali è la progettazione concettuale. Per essere indipendenti dal particolare modello logico dei dati si procede con una progettazione di alto livello che permette di individuare facilmente i concetti fondamentali, le associazioni e i vincoli tra essi.

2.1.1 Modello ER



Individuazione delle associazioni:

- **PARTECIPAZIONE:** Associazione che intercorre tra *MEMBRI DELL'EQUIPAGGIO* e *MISSIONE*. Un membro può partecipare ad una o più missioni e ad una missione possono partecipare uno o più membri, risultando in un'associazione molti a molti.

- **RELAZIONE:** Associazione che intercorre tra *REPORT* e *MISSIONE*. Un report può riferirsi ad una ed una sola missione, una missione può avere dai zero agli N report, risultando quindi in un'associazione uno a molti.
- **COINVOLGIMENTI:** Associazione che intercorre tra *MEMBRI DELL'EQUIPAGGIO* e *INTERVENTO*. Un membro può partecipare a zero o N interventi, un intervento può coinvolgere uno o più membri, risultando in un'associazione molti a molti.
- **IMPIEGO SENSORI:** Associazione che intercorre tra *SENSORI* e *MISSIONI*. Un sensore può essere impiegato per zero o N missioni, una missione prevede l'impiego di uno o più sensori, risultando in un'associazione molti a molti.
- **IMPIEGO ROBOT:** Associazione che intercorre tra *ROBOT* e *MISSIONI*. Un robot può essere impiegato per zero o N missioni, una missione prevede l'impiego di uno o più robot, risultando in un'associazione molti a molti.
- **CATTURA:** Associazione che intercorre tra *SENSORE* e *RILEVAZIONE*. Un sensore può effettuare da zero alle N rilevazioni, e una rilevazione è relativa al sensore che l'ha effettuata, risultando in una relazione uno a molti.
- **AVARIA:** Associazione che intercorre tra *SENSORE* e *ANOMALIA*. Un sensore può riscontrare dalle zero alle N anomalie, un'anomalia è invece relativa al sensore che l'ha riscontrata, risultando così in una relazione uno a molti.
- **GENERA:** Associazione che intercorre tra *ANOMALIA* e *INTERVENTO*. Un'anomalia può generare uno e un solo intervento relativo all'anomalia che l'ha originato, risultando in una relazione uno ad uno.

2.2 PROGETTAZIONE LOGICA

2.2.1 Fase di trasformazione

Nel caso in esame abbiamo deciso di scomporre l'attributo *COORDINATE* e legare le sue componenti all'entità *SENSORE*.

2.2.2 Fase di traduzione

La fase di traduzione comporta le seguenti azioni:

- Ogni entità si trasforma in una relazione avente come attributi gli attributi dell'entità e come chiave primaria l'identificatore dell'entità.
- Le associazioni *AVARIA*, *CATTURA*, e *RELAZIONE* scompaiono e contemporaneamente, gli identificatori, opportunamente ridenominati, dell'entità lato molti si aggiungono agli attributi delle relazioni relative al lato uno. I nuovi attributi inseriti diventano anche chiavi esterne referenzianti le relazioni relative alle entità lato molti.

- Le associazioni PARTECIPAZIONE, COINVOLGIMENTI, IMPIEGO ROBOT e IMPIEGO SENSORI divengono relazioni aventi come chiave primaria la coppia degli identificatori, opportunamente ridenominati, delle due entità che associa. Tali attributi costituiranno anche delle chiavi esterne referenzianti le relazioni relative all'entità da cui provengono.
- L'associazione GENERA scompare e l'identificatore di anomalia, opportunamente ridenominato, viene aggiunto a INTERVENTO. Tale attributo è chiave esterna e gode del vincolo di unicità.

Schema logico risultante:

PARTECIPAZIONI (Ruolo, Missione: Missioni, Membro: Membri)

COINVOLGIMENTI (Membro: Membri, Intervento: Interventi)

IMPIEGAZIONI SENSORI (Missione: Missioni, Sensore: Sensori)

IMPIEGAZIONI ROBOT (Missione: Missioni, Robot: Robot)

REPORT (Codice, Missione: Missioni)

RILEVAZIONI (Codice, Sensore: Sensori, Data, Ora, Valore)

ANOMALIE (Codice, Sensore: Sensori, Data, Ora, Livello, Priorità)

INTERVENTI (Codice, Descrizione, Esito, Data Intervento, Anomalia*: Anomalie)

MISSIONI (Codice, Obiettivo, Data Inizio, Data Fine, Spato)

SENSORI (Codice, Stato, Data Installazione, Data Ultimo Controllo, Long, Lat, Alt, Tipologia Sensore)

ROBOT (Codice, Tipo)

MEMBRI (Codice, Matricola, Nome, Cognome)

2.3 PROGETTAZIONE FISICA

2.3.1 Creazione delle tabelle

Creazione della tabella SENSORI

Dimensione di ogni riga circa 151 byte.

Column Name	Data Type
CODICE	NUMBER
STATO	VARCHAR2(50)
DATA_INSTALLAZIONE	DATE
TIPOLOGIA	VARCHAR2(50)
DATA_ULTIMO_CONTROLLO	DATE
LONGITUDINE	NUMBER(10,7)
LATITUDINE	NUMBER(10,7)
ALTITUDINE	NUMBER(6,2)

```
1 CREATE TABLE Sensori (  
2     codice NUMBER,  
3     stato VARCHAR2(50) NOT NULL,  
4     data_installazione DATE,  
5     tipologia VARCHAR2(50) NOT NULL,  
6     data_ultimo_controllo DATE,  
7     longitudine DECIMAL(10, 7),  
8     latitudine DECIMAL(10, 7),  
9     altitudine NUMBER(6, 2),  
10    CONSTRAINT PK_SENSORI PRIMARY KEY(codice),  
11    CONSTRAINT CK_TIPOLOGIA_SENSORI CHECK(tipologia='TEMPERATURA '  
12    OR tipologia='PRESSIONE' OR tipologia='GAS' OR  
13    tipologia='RADIAZIONI' OR tipologia='GEOLOGIA'),  
14    CONSTRAINT CK_STATO_SENSORI CHECK(stato='ATTIVO' OR  
15    stato='MANUTENZIONE' OR stato='MALFUNZIONANTE' OR  
16    stato='DISATTIVO' OR stato='STANDBY')  
17 );
```

Creazione della tabella MISSIONI

Dimensione di ogni riga circa 336 byte.

Column Name	Data Type
CODICE	NUMBER
OBIETTIVO	VARCHAR2(250)
DATA_INIZIO	DATE
DATA_FINE	DATE
STATO	VARCHAR2(50)

```

1 CREATE TABLE Missioni (
2     codice NUMBER,
3     obiettivo VARCHAR2(250) NOT NULL,
4     data_inizio DATE,
5     data_fine DATE,
6     stato VARCHAR2(50),
7     CONSTRAINT PK_MISSIONI PRIMARY KEY(codice),
8     CONSTRAINT CK_STATO_MISSIONE CHECK(stato='INCORSO' OR
          stato='PIANIFICATA' OR stato='COMPLETATA' OR
          stato='ANNULLATA')
9 );

```

Creazione della tabella ROBOT

Dimensione di ogni riga circa 72 byte.

Column Name	Data Type
CODICE	NUMBER
TIPO	VARCHAR2(50)

```

1 CREATE TABLE Robot (
2     codice NUMBER,
3     tipo VARCHAR2(50) NOT NULL,
4     CONSTRAINT PK_ROBOT PRIMARY KEY(codice)
5 );

```

Creazione della tabella MEMBRI-EQUIPAGGIO

Dimensione di ogni riga circa 82 byte.

Column Name	Data Type
CODICE	NUMBER
MATRICOLA	VARCHAR2(20)
NOME	VARCHAR2(20)
COGNOME	VARCHAR2(20)


```

1 CREATE TABLE Membri_Equipaggio (
2     codice NUMBER,
3     matricola VARCHAR2(20) UNIQUE,
4     nome VARCHAR2(20) NOT NULL,
5     cognome VARCHAR2(20) NOT NULL,
6     CONSTRAINT PK_MEMBRI_EQUIPAGGIO PRIMARY KEY(codice)
7 );

```

Creazione della tabella PARTECIPAZIONI

Dimensione di ogni riga circa 94 byte.

Column Name	Data Type
RUOLO	VARCHAR2(50)
MISSIONE	NUMBER
MEMBRO	NUMBER

```

1 CREATE TABLE Partecipazioni (
2     ruolo VARCHAR2(50),
3     missione NUMBER,
4     membro NUMBER,
5     CONSTRAINT PK_PARTECIPAZIONI PRIMARY KEY(ruolo, missione,
6     membro)
7 );

```

Creazione della tabella IMPIEGO-SENSORI

Dimensione di ogni riga circa 44 byte.

Column Name	Data Type
MISSIONE	NUMBER
SENSORE	NUMBER

```

1 CREATE TABLE Impiego_Sensori (
2     missione NUMBER,
3     sensore NUMBER,
4     CONSTRAINT PK_IMPIEGO_SENSORI PRIMARY KEY(missione, sensore)
5 );

```

Creazione della tabella IMPIEGO-ROBOT

Dimensione di ogni riga circa 44 byte.

Column Name	Data Type
MISSIONE	NUMBER
ROBOT	NUMBER

```

1 CREATE TABLE Impiego_Robot (
2     missione NUMBER,
3     robot NUMBER,
4     CONSTRAINT PK_IMPIEGO_ROBOT PRIMARY KEY(missione, robot)
5 );

```

Creazione della tabella REPORT

Dimensione di ogni riga circa 266 byte.

Column Name	Data Type
CODICE	NUMBER
MISSIONE	NUMBER
MEMBRO	NUMBER
STATO	VARCHAR2(200)

```

1 CREATE TABLE Report (
2     codice NUMBER,
3     missione NUMBER NOT NULL,
4     membro NUMBER NOT NULL,
5     stato VARCHAR2(200),
6     CONSTRAINT PK_REPORT PRIMARY KEY(codice)
7 );

```

Creazione della tabella RILEVAZIONI

Dimensione di ogni riga circa 67 byte.

Column Name	Data Type
CODICE	NUMBER
SENSORE	NUMBER
DATA	DATE
ORA	TIMESTAMP(6)
VALORE	NUMBER(10,7)

```

1 CREATE TABLE Rilevazioni (
2     codice NUMBER,

```

```

3      sensore NUMBER,
4      data DATE NOT NULL,
5      ora TIMESTAMP NOT NULL,
6      valore DECIMAL(10,7),
7      CONSTRAINT PK_RILEVAZIONI PRIMARY KEY(codice,sensore)
8 );

```

Creazione della tabella ANOMALIE

Dimensione di ogni riga circa 339 byte.

Column Name	Data Type
CODICE	NUMBER
SENSORE	NUMBER
DATA	DATE
ORA	TIMESTAMP(6)
CAUSA	VARCHAR2(255)
LIVELLO_PRIORITA	NUMBER

```

1 CREATE TABLE Anomalie (
2     codice NUMBER,
3     sensore NUMBER,
4     data DATE NOT NULL,
5     ora TIMESTAMP NOT NULL,
6     causa VARCHAR2(255),
7     livello_priorita NUMBER,
8     CONSTRAINT PK_ANOMALIE PRIMARY KEY(codice,sensore)
9 );

```

Creazione della tabella COINVOLGIMENTI

Dimensione di ogni riga circa 44 byte.

Column Name	Data Type
EQUIPAGGIO	NUMBER
INTERVENTO	NUMBER

```

1 CREATE TABLE Coinvolgimenti (
2     equipaggio NUMBER,
3     intervento NUMBER,
4     CONSTRAINT PK_COINVOLGIMENTI PRIMARY KEY(equipaggio, intervento)
5 );

```

Creazione della tabella INTERVENTI

Dimensione di ogni riga circa 151 byte.

Column Name	Data Type
CODICE	NUMBER
DESCRIZIONE	VARCHAR2(50)
ESITO	VARCHAR2(50)
DATA_INTERVENTO	DATE
ANOMALIA	NUMBER

```
1  -- Tabella Interventi
2  CREATE TABLE Interventi (
3      codice NUMBER,
4      descrizione VARCHAR2(100) NOT NULL,
5      esito VARCHAR2(50),
6      data_intervento DATE NOT NULL,
7      anomalia NUMBER UNIQUE,
8      CONSTRAINT PK_INTERVENTI PRIMARY KEY(codice),
9      CONSTRAINT CK_ESITO_INTERVENTO CHECK(esito='RIUSCITO' OR
10         esito='ANNULLATO' OR esito='FALLITO')
11 );
```

2.3.2 Scrittura dei vincoli inter-relazionali

- Partecipazioni

```
1  ALTER TABLE Partecipazioni
2  ADD CONSTRAINT FK_PARTECIPAZIONI_MISSIONI FOREIGN KEY(missione)
3      REFERENCES Missioni(codice) ON DELETE CASCADE;
4
5  ALTER TABLE Partecipazioni
6  ADD CONSTRAINT FK_PARTECIPAZIONI_MEMBRI_EQUIPAGGIO FOREIGN
7      KEY(membro) REFERENCES Membri_Equipaggio(codice) ON DELETE SET
8      NULL;
```

- Impiego Sensori

```
1  ALTER TABLE Impiego_Sensori
2  ADD CONSTRAINT FK_IMPIEGO_SENSORI_MISSIONI FOREIGN KEY(missione)
3      REFERENCES Missioni(codice) ON DELETE CASCADE;
4
5  ALTER TABLE Impiego_Sensori
6  ADD CONSTRAINT FK_IMPIEGO_SENSORI_SENSORI FOREIGN KEY(sensore)
7      REFERENCES Sensori(codice) ON DELETE CASCADE;
```

- Impiego Robot

```

1 ALTER TABLE Impiego_Robot
2 ADD CONSTRAINT FK_IMPIEGO_ROBOT_MISSIONI FOREIGN KEY(missione)
  REFERENCES Missioni(codice) ON DELETE CASCADE;
3
4 ALTER TABLE Impiego_Robot
5 ADD CONSTRAINT FK_IMPIEGO_ROBOT_ROBOT FOREIGN KEY(robot) REFERENCES
  Robot(codice) ON DELETE CASCADE;

```

- Report

```

1 ALTER TABLE Report
2 ADD CONSTRAINT FK_REPORT_MISSIONI FOREIGN KEY(missione) REFERENCES
  Missioni(codice) ON DELETE CASCADE;
3
4 ALTER TABLE Report
5 ADD CONSTRAINT FK_REPORT_MEMBRI_EQUIPAGGIO FOREIGN KEY(membro)
  REFERENCES Membri_Equipaggio(codice) ON DELETE SET NULL;

```

- Rilevazioni

```

1 ALTER TABLE Rilevazioni
2 ADD CONSTRAINT FK_RILEVAZIONI_SENSORI FOREIGN KEY(sensore)
  REFERENCES Sensori(codice) ON DELETE CASCADE;

```

- Anomalie

```

1 ALTER TABLE Anomalie
2 ADD CONSTRAINT FK_ANOMALIE_SENSORI FOREIGN KEY(sensore) REFERENCES
  Sensori(codice) ON DELETE CASCADE;

```

- Interventi

```

1 ALTER TABLE Interventi
2 ADD CONSTRAINT FK_INTERVENTI_ANOMALIE FOREIGN KEY(anomalia)
  REFERENCES Anomalie(codice) ON DELETE CASCADE;

```

- Coinvolgimenti

```

1 ALTER TABLE Coinvolgimenti
2 ADD CONSTRAINT FK_COINVOLGIMENTI_MEMBRI FOREIGN KEY(equipaggio)
  REFERENCES Membri_Equipaggio(codice) ON DELETE SET NULL;
3
4 ALTER TABLE Coinvolgimenti
5 ADD CONSTRAINT FK_COINVOLGIMENTI_INTERVENTI FOREIGN KEY(intervento)
  REFERENCES Interventi(codice) ON DELETE CASCADE;

```

2.3.3 Creazione delle sequenze

Sequenza 1: sequenza per codice sensori

```
1 CREATE SEQUENCE seq_cod_sensori
2 START WITH 4000 INCREMENT BY 50
3 MINVALUE 4000 NOMAXVALUE NOCYCLE;
```

Sequenza 2: sequenza matricola membri-equipaggio

```
1 CREATE SEQUENCE seq_matr_membri
2 START WITH 10000 INCREMENT BY 100
3 MINVALUE 10000 NOMAXVALUE NOCYCLE;
```

2.3.4 Creazione degli indici

Indice 1: Indice sulla colonna stato nella tabella SENSORI

```
1 CREATE INDEX idx_sensori_stato ON SENSORI (stato);
```

Indice 2: Indice sulla colonna data-installazione nella tabella sensori

```
1 CREATE INDEX idx_view_missioni_data_sensore ON SENSORI
  (data-installazione);
```

Indice 3: Indice sulla colonna sensore nella tabella ANOMALIE

```
1 CREATE INDEX idx_anomalie_sensore ON ANOMALIE (sensore);
```

Indice 4: Indice sulla colonna valore nella tabella RILEVAZIONI

```
1 CREATE INDEX idx_rilevazioni_valore ON RILEVAZIONI (valore);
```

Indice 5: Indice sulla colonna priorit -anomalia nella vista view-missioni-in-corso

```
1 CREATE INDEX idx_view_missioni_priorita_anomalia ON ANOMALIE
  (livello_priorita);
```

3 GESTIONE PROGETTO

3.1 Gestione della Concorrenza

Nel contesto della gestione delle transazioni in un sistema di basi di dati, l'obiettivo di un protocollo di controllo di concorrenza è garantire l'integrità e la coerenza dei dati quando più transazioni operano contemporaneamente sulla base di dati.

Abbiamo scelto di adottare il protocollo di locking a due fasi (**2PL**) per raggiungere questo obiettivo.

Il protocollo 2PL prevede due fasi distinte:

- **Fase crescente:** le transazioni ottengono lock in lettura e scrittura sugli oggetti della base di dati con cui intendono interagire.
- **Fase decrescente:** vengono rilasciate le risorse in possesso al termine delle operazioni.

Questo approccio impedisce la comparsa di anomalie di concorrenza.

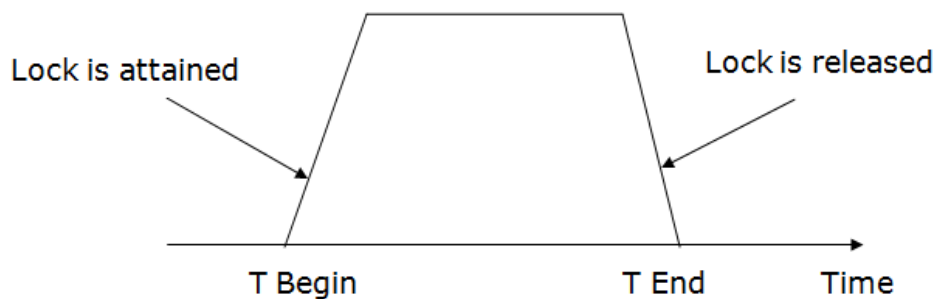


Figura 1: Lock based protocol

3.2 Gestione dell'Affidabilità

Il controllo di affidabilità ha come obiettivo il "ripristino" del corretto stato (**recovery**) di un sistema di basi di dati a valle di un possibile malfunzionamento delle sue componenti hardware o software, dovuto a guasti accidentali o intenzionali.

Gli aspetti principali della gestione dell'affidabilità sono:

- **Garantire le proprietà di atomicità e persistenza** delle transazioni, che rappresentano l'unità base delle attività di recovery.
- **Assicurare la memorizzazione permanente** degli effetti di tutte le transazioni che hanno effettuato il commit, in modo che sia possibile ripristinare il contenuto corretto della base di dati anche a seguito di guasti.

Un problema rilevante è che le operazioni di scrittura delle transazioni non sono atomiche. Gli effetti di una transazione che ha effettuato il commit in memoria primaria potrebbero non essere immediatamente riportati su memoria secondaria e, quindi, resi persistenti.

Concetti base del recovery Il gestore dell'affidabilità deve gestire:

- **Comandi transazionali:** begin transaction, commit, rollback (abort).
- **Operazioni di ripristino dopo guasti.**

Per effettuare queste operazioni, il gestore utilizza:

- **Log:** una sorta di "diario di bordo" che permette, in qualsiasi istante, di ricostituire il contenuto corretto della base di dati a seguito di malfunzionamenti.
- **Checkpoint:** registra le transazioni attive in un determinato istante e verifica che le altre transazioni siano o completate o non iniziate.
- **Dump:** corrisponde alla classica operazione di backup o copia del contenuto della base di dati su memoria stabile, utile per la ricostruzione del contenuto soprattutto in caso di danneggiamento della memoria secondaria non stabile.

```
DP, B(T1,-, -, -), U(T1,-, qtaP,100,90), U(T1,-, qtaC,NULL,10), C(T1,-, -, -),
B(T2,-, -, -), CK(T2), U(T2,-, qtaP,90,70), U(T1,-, qtaC,NULL,20), C(T2,-, -, -),
B(T3,-, -, -), U(T3,-, qtaP,100,90), U(T3,-, qtaC,NULL,10), C(T3,-, -, -),
...
```

Figura 2: File di log

Gestione di ridondanza sui dati Nel nostro caso, abbiamo deciso di utilizzare una configurazione di tipo **RAID 5**, che rappresenta un compromesso ottimale tra:

- **Sicurezza:** grazie all'uso di un blocco di parità interlacciato.
- **Rapidità:** nelle operazioni di lettura e scrittura.

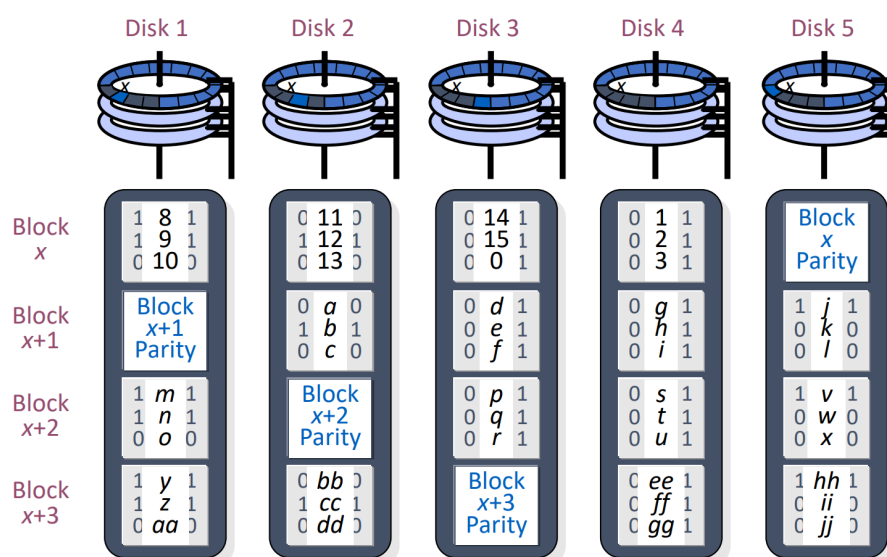


Figura 3: Configurazione RAID 5

4 OPERAZIONI SULLA BASE DATI

4.1 Query

Query di utilità per la dashboard e le varie pagine dell'applicazione.

Query numero 1

```
1 SELECT S.STATO AS "Stato",
2       COUNT(DISTINCT S.codice) AS "Numero di Sensori"
3 FROM MISSIONI M
4 JOIN PARTECIPAZIONI P ON P.missione = M.codice
5 JOIN MEMBRI_EQUIPAGGIO MB ON MB.codice = P.membro
6 JOIN IMPIEGO_SENSORI ISE ON M.codice = ISE.missione
7 JOIN SENSORI S ON ISE.sensore = S.codice
8 WHERE (UPPER(MB.matricola) = UPPER(:APP_USER) AND
9        M.stato='INCORSO') OR UPPER(:APP_USER)='SUPERVISOR'
9 GROUP BY S.stato
```

Questa query restituisce il numero dei sensori che si trovano in un determinato stato, dipendentemente dal fatto che un sensore sia previsto per una missione in corso a cui partecipa il membro dell'equipaggio.

Query numero 2

```
1 SELECT
2     A.livello_priorita AS "Priorità",
3     COUNT(DISTINCT A.codice) AS "Numero di Anomalie",
4     COUNT(DISTINCT R.codice) AS "Numero di Rilevazioni",
5     ROUND(COUNT(DISTINCT R.codice) * 1.0 / COUNT(DISTINCT
6           A.codice), 4) AS "Rapporto RILEVAZIONI_ANOMALIE"
7 FROM MISSIONI M
8 JOIN PARTECIPAZIONI P ON P.missione = M.codice
9 JOIN MEMBRI_EQUIPAGGIO MB ON MB.codice = P.membro
10 JOIN IMPIEGO_SENSORI ISE ON M.codice = ISE.missione
11 JOIN SENSORI S ON ISE.sensore = S.codice
12 JOIN ANOMALIE A ON A.sensore = S.codice
13 LEFT JOIN RILEVAZIONI R ON R.sensore = S.codice
14 WHERE
15     (UPPER(MB.matricola) = UPPER(:APP_USER) AND M.stato = 'INCORSO')
16     OR UPPER(:APP_USER) = 'SUPERVISOR'
17 GROUP BY A.livello_priorita
18 HAVING COUNT(DISTINCT A.codice) > 0
19 ORDER BY A.livello_priorita ASC;
```

Questa query restituisce il rapporto fra numero di rilevazioni e numero di anomalie per ogni sensore attivo nella missione.

Query numero 3

```
1 SELECT
2     count(R.codice) as NUMERO_RILEVAZIONI,
3     S.codice,
4     AVG(R.valore)
```

```

5 FROM MISSIONI M
6 JOIN PARTECIPAZIONI P ON P.missione = M.codice
7 JOIN MEMBRI_EQUIPAGGIO MB ON MB.codice = P.membro
8 JOIN IMPIEGO_SENSORI ISE ON M.codice = ISE.missione
9 JOIN SENSORI S ON ISE.sensore = S.codice
10 JOIN RILEVAZIONI R ON R.sensore = S.codice
11 WHERE
12     (M.stato = 'INCORSO' AND UPPER(MB.matricola) = UPPER(:APP_USER))
13     OR UPPER(:APP_USER)='SUPERVISOR'
14 GROUP BY
15     S.codice

```

Questa query restituisce il numero di rilevazioni e la loro media aritmetica per ogni sensore attivo nella missione.

Query numero 4

```

1 SELECT M.obiettivo, TO_CHAR(data_inizio, 'DD/MM/YYYY') AS
   DATA_INIZIO,
2     CASE
3         WHEN data_fine IS NULL OR data_fine > SYSDATE THEN 'IN
   CORSO'
4         ELSE TO_CHAR(data_fine, 'DD/MM/YYYY')
5     END AS data_fine,
6     M.stato
7 FROM MISSIONI M
8 JOIN PARTECIPAZIONI P ON P.missione=M.codice
9 JOIN MEMBRI_EQUIPAGGIO MB ON MB.codice=P.membro
10 WHERE UPPER(MB.matricola) = UPPER(:APP_USER) OR
   UPPER(:APP_USER)='SUPERVISOR'
11 GROUP BY M.codice, M.obiettivo, M.stato, M.data_inizio, M.data_fine

```

Questa query restituisce le informazioni su una missione a cui ha partecipato un membro dell'equipaggio. Se la missione è ancora in corso, di conseguenza la data di fine missione è NULL, restituisce 'IN CORSO' anziché il valore NULL.

Query numero 5

```

1 SELECT
   AN.codice, AN.causa, AN.livello_priorita, AN.sensore, AN.data, AN.ora
2 FROM ANOMALIE AN
3 JOIN IMPIEGO_SENSORI ISE ON ISE.sensore=AN.sensore
4 JOIN MISSIONI M ON ISE.missione=M.codice
5 JOIN PARTECIPAZIONI P ON P.missione=M.codice
6 JOIN MEMBRI_EQUIPAGGIO MB ON MB.codice=P.membro
7 WHERE UPPER(MB.matricola) = UPPER(:APP_USER) OR
   UPPER(:APP_USER)='SUPERVISOR'
8 GROUP BY AN.codice, AN.causa, AN.livello_priorita, AN.data, AN.ora,
   AN.sensore

```

Questa query restituisce i dati delle anomalie che un sensore ha riscontrato in una missione in cui ha partecipato l'utente che accede all'applicazione.

Query numero 6

```
1 SELECT I.codice,I.descrizione,I.esito,TO_CHAR(I.data_intervento,
   'DD/MM/YYYY') AS data_intervento,I.anomalia
2 FROM INTERVENTI I
3 JOIN COINVOLGIMENTI CO ON I.codice=CO.intervento
4 JOIN MEMBRI_EQUIPAGGIO MB ON CO.equipaggio=MB.codice
5 WHERE UPPER(MB.matricola) = UPPER(:APP_USER) OR
   UPPER(:APP_USER)='SUPERVISOR'
6 GROUP BY I.codice,I.descrizione,I.esito,I.data_intervento,I.anomalia
```

Questa query restituisce le informazioni riguardo gli interventi svolti per risolvere le anomalie.

4.2 Viste

Vista delle missioni completate da ogni membro

```
1 CREATE VIEW missioni_completate AS
2 SELECT
3     p.membro as codice_membro,
4     COUNT() as num_missioni_completate
5 FROM Partecipazioni p
6 JOIN Missioni m ON p.missione = m.codice
7 WHERE m.stato = 'COMPLETATA'
8 GROUP BY p.membro;
```

Vista della durata media delle missioni per ogni membro

```
1 CREATE VIEW durata_media_missioni as
2 select p.membro as codice_membro,
3     avg(m.data_fine-m.data_inizio) as media_durata_missioni
4 FROM Partecipazioni p
5 JOIN Missioni m ON p.missione = m.codice
6 WHERE m.stato = 'COMPLETATA'
7 GROUP BY p.membro;
```

Vista per numero totali interventi di ogni membro

```
1 CREATE OR REPLACE VIEW num_totali_interventi as
2 SELECT me.codice as codice_membro,
3     COUNT() as num_interventi
4 FROM MEMBRI_EQUIPAGGIO ME
5 JOIN COINVOLGIMENTI C ON ME.codice = c.equipaggio
6 GROUP BY ME.codice
```

Vista per numero totali interventi riusciti di ogni membro

```
1 CREATE OR REPLACE VIEW num_totali_interventi_riusciti as
2 SELECT me.codice as codice_membro,
3     COUNT(*) as num_interventi_riusciti
4 FROM MEMBRI_EQUIPAGGIO ME
```

```

5 JOIN COINVOLGIMENTI C ON ME.codice = C.equipaggio
6 join INTERVENTI I ON I.CODICE=C.intervento
7 WHERE I.esito='RIUSCITO'
8 GROUP BY ME.codice

```

Vista numero totali ruoli ricoperti da un membro

```

1 CREATE VIEW ruoli_ricoperti AS
2 SELECT
3     membro as codice_membro,
4     COUNT(DISTINCT ruolo) as numero_ruoli
5 FROM Partecipazioni
6 GROUP BY membro;

```

Vista dataset membri

```

1 CREATE OR REPLACE VIEW dati_membri AS
2 SELECT
3     MC.codice_membro,
4     MC.num_missioni_completate,
5     DME.media_durata_missioni,
6     RR.numero_ruoli,
7     NTI.num_interventi,
8     NTIR.num_interventi_riusciti
9 FROM missioni_completate MC
10 JOIN durata_media_missioni DME ON MC.codice_membro =
    DME.codice_membro
11 JOIN ruoli_ricoperti RR ON MC.codice_membro = RR.codice_membro
12 LEFT JOIN num_totali_interventi NTI ON
    NTI.codice_membro=RR.codice_membro
13 LEFT JOIN num_totali_interventi_riusciti NTIR ON NTI.codice_membro
    = NTIR.codice_membro;

```

vista per ottenere i sensori e le loro anomalie di missioni in corso

```

1 CREATE VIEW view_missioni_in_corso AS
2 SELECT M.codice AS codice_missione,
3     M.stato AS stato_missione,
4     S.codice AS codice_sesnore,
5     s.stato AS stato_sensore,
6     s.data_installazione AS data_sensore,
7     AN.codice AS codice_anomalia,
8     AN.livello_priorita AS priorit_a_anomalia
9 FROM MISSIONI M JOIN IMPIEGO_SENSORI ISE ON M.codice=ISE.missione
10 JOIN SENSORI S ON S.codice=ISE.sensore
11 JOIN ANOMALIE AN ON AN.sensore=S.codice
12 WHERE M.stato = 'INCORSO'

```

4.3 Trigger

Trigger n.1: per la politica di update 'cascade' sui SENSORI

```
1 CREATE TRIGGER trg_update_sensori
2 AFTER UPDATE ON Sensori
3 FOR EACH ROW
4 BEGIN
5     UPDATE Impiego_Sensori
6     SET sensore = :NEW.codice
7     WHERE sensore = :OLD.codice;
8
9     UPDATE RILEVAZIONI
10    SET sensore = :NEW.codice
11    WHERE sensore = :OLD.codice;
12
13    UPDATE ANOMALIE
14    SET sensore = :NEW.codice
15    WHERE sensore = :OLD.codice;
16
17 END;
```

Trigger n.2: per la politica di update 'cascade' sui MEMBRI

```
1 CREATE TRIGGER trg_update_membri
2 AFTER UPDATE ON MEMBRI_EQUIPAGGIO
3 FOR EACH ROW
4 BEGIN
5     UPDATE COINVOLGIMENTI
6     SET EQUIPAGGIO = :NEW.codice
7     WHERE EQUIPAGGIO = :OLD.codice;
8
9     UPDATE PARTECIPAZIONI
10    SET MEMBRO = :NEW.codice
11    WHERE MEMBRO = :OLD.codice;
12 END;
```

Trigger n.3: per la politica di update 'cascade' sulle MISSIONI

```
1 CREATE OR REPLACE TRIGGER trg_update_missioni
2 AFTER UPDATE ON MISSIONI
3 FOR EACH ROW
4 BEGIN
5     UPDATE IMPIEGO_SENSORI
6     SET MISSIONE = :NEW.codice
7     WHERE MISSIONE = :OLD.codice;
8
9     UPDATE IMPIEGO_ROBOT
10    SET MISSIONE = :NEW.codice
11    WHERE MISSIONE = :OLD.codice;
12
```

```

13 UPDATE PARTECIPAZIONI
14 SET MISSIONE = :NEW.codice
15 WHERE MISSIONE = :OLD.codice;
16
17 UPDATE REPORT
18 SET MISSIONE = :NEW.codice
19 WHERE MISSIONE = :OLD.codice;
20
21 END;

```

Trigger n.4: per la politica di update del codice dei SENSORI con la sequenza

```

1 CREATE OR REPLACE TRIGGER trg_sensori_codice
2 BEFORE INSERT ON SENSORI
3 FOR EACH ROW
4 BEGIN
5     :NEW.codice := seq_cod_sensori.NEXTVAL;
6 END;

```

Trigger n.5: incremento del codice della rilevazione e controllo sulla sua data

```

1 CREATE OR REPLACE TRIGGER trg_insert_rilevazioni
2 BEFORE INSERT ON RILEVAZIONI
3 FOR EACH ROW
4 DECLARE
5     num_rilevazioni NUMBER;
6     stato_sensore   SENSORI.STATO%TYPE;
7     data_sensore    SENSORI.data_installazione%TYPE;
8 BEGIN
9     SELECT stato, data_installazione
10    INTO stato_sensore, data_sensore
11   FROM Sensori
12  WHERE codice = :NEW.sensore;
13
14   IF stato_sensore <> 'ATTIVO' OR :NEW.data < data_sensore THEN
15       RAISE_APPLICATION_ERROR(-20001, 'Impossibile inserire
16         rilevazione');
17   END IF;
18
19   SELECT count(*) INTO num_rilevazioni
20  FROM RILEVAZIONI
21  WHERE sensore = :NEW.sensore;
22
23   :NEW.codice := (num_rilevazioni + 1);
24 END;

```

Trigger n.6 per la politica di update matricola con sequenza

```

1 CREATE OR REPLACE TRIGGER trg_membri_matricola
2 BEFORE INSERT ON MEMBRI_EQUIPAGGIO
3 FOR EACH ROW
4 BEGIN

```

```

5      :NEW.matricola := 'MEM' || TO_CHAR(seq_matr_membri.NEXTVAL);
6  END

```

Trigger n.7: all'atto di un'anomalia genera un intervento la cui data dipende dal livello di priorità

```

1  CREATE OR REPLACE TRIGGER trg_inserimento_intervento
2  AFTER INSERT ON ANOMALIE
3  FOR EACH ROW
4  DECLARE
5      data_intervento INTERVENTI.data_intervento%TYPE;
6      descrizione INTERVENTI.descrizione%TYPE;
7  BEGIN
8      DBMS_OUTPUT.PUT_LINE('Data Anomalia Inserita: ' ||
9          TO_CHAR(:NEW.data, 'DD/MM/YYYY'));
10
11      IF :NEW.livello_priorita = '1' THEN
12          descrizione := 'Causa: ' || :NEW.causa || ', calibrazione
13              del sensore ';
14          data_intervento := ADD_MONTHS(:NEW.data, 1);
15      ELSIF :NEW.livello_priorita = '2' THEN
16          descrizione := 'Causa: ' || :NEW.causa || ', riparazione
17              sul sensore ';
18          data_intervento := :NEW.data + 7;
19      ELSE
20          descrizione := 'Causa: ' || :NEW.causa || ', sostituzione
21              immediata del sensore ';
22          data_intervento := :NEW.data + 1;
23      END IF;
24
25      DBMS_OUTPUT.PUT_LINE('Data Intervento Calcolata: ' ||
26          TO_CHAR(data_intervento, 'DD/MM/YYYY'));
27      DBMS_OUTPUT.PUT_LINE('Descrizione -> ' || descrizione);
28
29      INSERT INTO INTERVENTI (codice, descrizione, esito,
30          data_intervento, anomalia)
31      VALUES (
32          :NEW.codice + 1,
33          descrizione,
34          NULL,
35          data_intervento,
36          :NEW.codice
37      );
38  END;

```

Questo trigger rispetta le scelte di progettazione

```
Data Anomalia Inserita: 13/01/2025
Data Intervento Calcolata: 20/01/2025
Descrizione -> Causa: Guasto, riparazione sul sensore

1 row(s) inserted.
```

Trigger n.8: per generazione codice in automatico, controllo che la data dell'anomalia sia maggiore della data di installazione del trigger

```
1 CREATE OR REPLACE TRIGGER trg_insert_anomalie
2 BEFORE INSERT ON ANOMALIE
3 FOR EACH ROW
4 DECLARE
5     num_anomalie NUMBER;
6     data_sensore SENSORI.data_installazione%TYPE;
7 BEGIN
8
9     SELECT data_installazione
10    INTO data_sensore
11   FROM Sensori
12  WHERE codice = :NEW.sensore;
13
14  IF :NEW.data < data_sensore THEN
15      RAISE_APPLICATION_ERROR(-20001, 'Errore data anomalia');
16  END IF;
17
18  select count(*) INTO num_anomalie
19  FROM ANOMALIE
20  WHERE sensore = :NEW.sensore;
21  :NEW.codice := (num_anomalie+1);
22
23 END;
```

Trigger n.9: per aggiornare lo stato sensore all'atto della generazione di un'anomalia

```
1 CREATE OR REPLACE TRIGGER trg_update_sensore_stato_anomalie
2 AFTER INSERT ON ANOMALIE
3 FOR EACH ROW
4 BEGIN
5     DECLARE
6         new_stato SENSORI.stato%TYPE;
7     BEGIN
8         IF :NEW.livello_priorita > 2 THEN
9             new_stato := 'MALFUNZIONANTE';
10        ELSIF :NEW.livello_priorita < 3 THEN
11            new_stato := 'MANUTENZIONE';
12        END IF;
13
14        UPDATE SENSORI
15        SET stato = new_stato
```



```

16         WHERE codice = :NEW.sensore;
17     END;
18 END;

```

Trigger n.10: per aggiornare lo stato sensore alla terminazione di un intervento

```

1 CREATE OR REPLACE TRIGGER trg_update_sensore_stato
2 AFTER UPDATE ON INTERVENTI
3 FOR EACH ROW
4 DECLARE
5     new_stato SENSORI.stato%TYPE;
6     livelloprio ANOMALIE.livello_priorita%TYPE;
7     codice_sensore SENSORI.codice%TYPE;
8 BEGIN
9     SELECT livello_priorita, sensore
10    INTO livelloprio, codice_sensore
11   FROM ANOMALIE
12  WHERE codice = :NEW.anomalia;
13
14   IF :NEW.esito IN ('FALLITO', 'ANNULLATO') THEN
15       new_stato := 'DISATTIVO';
16   ELSIF :NEW.esito = 'RIUSCITO' THEN
17       new_stato := 'ATTIVO';
18   END IF;
19
20   UPDATE SENSORI
21  SET stato = new_stato
22  WHERE codice = codice_sensore;
23 END;

```

4.4 Stored procedure

Procedura n.1: per l'inserimento di un'anomalia

```

1 CREATE OR REPLACE PROCEDURE inserimento_anomalia(
2     sensore IN SENSORI.codice%TYPE,
3     causa IN ANOMALIE.causa%TYPE,
4     livello_priorita IN ANOMALIE.livello_priorita%TYPE)
5 AS
6 BEGIN
7     INSERT INTO ANOMALIE (sensore,data,ora,causa,livello_priorita)
8     VALUES(sensore,TRUNC(SYSDATE),SYSTIMESTAMP,causa,livello_priorita);
9 END;

```

Procedura n.2: per l'inserimento di una rilevazione

```

1 CREATE OR REPLACE PROCEDURE inserimento_rilevazione(
2     sensore IN SENSORI.codice%TYPE,
3     valore IN RILEVAZIONI.valore%TYPE

```

```

4      )
5  AS
6  BEGIN
7      INSERT INTO RILEVAZIONI (sensore,data,ora,valore)
8      VALUES(sensore,TRUNC(SYSDATE),SYSTIMESTAMP,valore);
9  END;

```

Procedura n.3 per impostare lo stato di un sensore in STAND-BY

```

1  CREATE OR REPLACE PROCEDURE stand_by_sensore_peggiore(
2      input_missione IN MISSIONI.codice%TYPE
3  )
4  AS
5      codice_sensore_sby SENSORI.codice%TYPE;
6      missione_esiste NUMBER := 0;
7      codice_inesistente EXCEPTION;
8  BEGIN
9      --controllo esistenza id missione in input
10     SELECT COUNT(*)
11     INTO missione_esiste
12     FROM view_missioni_in_corso
13     WHERE codice_missione = input_missione;
14     --se la missione esiste allora inserisci in codice_sensore_sby
        il sensore della missione selezionata che ha generato più
        anomalie di livello priorit  maggiore di uno e pi 
        datato(data installazione minore di tutti)
15     IF missione_esiste > 0 THEN
16         SELECT codice_sensore
17         INTO codice_sensore_sby
18         FROM (
19             SELECT codice_sensore, COUNT(*) AS num_anomalie,
20                 data_sensore
21             FROM view_missioni_in_corso
22             WHERE codice_missione = input_missione AND
23                 priorit _anomalia > 1 AND stato_sensore='ATTIVO'
24             GROUP BY codice_sensore, data_sensore
25             ORDER BY num_anomalie DESC, data_sensore ASC
26         )
27         WHERE ROWNUM = 1;
28
29         UPDATE SENSORI
30         SET stato = 'STANDBY'
31         WHERE codice = codice_sensore_sby;
32
33         DBMS_OUTPUT.PUT_LINE('il sensore ' || codice_sensore_sby ||
34             '   in stato standby');
35     ELSE
36         RAISE codice_inesistente;
37     END IF;

```

```

37 EXCEPTION
38     WHEN NO_DATA_FOUND THEN
39         DBMS_OUTPUT.PUT_LINE('Nessun sensore trovato per la
           missione ' || input_missione);
40     WHEN codice_inesistente THEN
41         DBMS_OUTPUT.PUT_LINE('Non esiste missione con codice ' ||
           input_missione || ' in corso');
42     WHEN OTHERS THEN
43         DBMS_OUTPUT.PUT_LINE('Errore sconosciuto: ' || SQLERRM);
44 END stand_by_sensore_peggior;

```

In particolare viene selezionato il sensore che ha generato più anomalie di alte priorità ed è più datato, l'obiettivo è quello di diminuire il carico di lavoro in situazioni di sovraccarico.

Procedura n.4: per analizzare l'efficienza dei membri

```

1 CREATE OR REPLACE PROCEDURE efficienza_membro(
2     codice_membro IN MEMBRI_EQUIPAGGIO.CODICE%TYPE
3 ) AS
4     interventi_riusciti NUMBER;
5     nome_membro VARCHAR2(100);
6     m_success_rate NUMBER;
7     p_total_missions NUMBER;
8     p_roles VARCHAR2(4000);
9     interventi_totali NUMBER;
10 BEGIN
11     SELECT nome || ' ' || cognome
12     INTO nome_membro
13     FROM Membri_Equipaggio
14     WHERE codice = codice_membro;
15
16     SELECT COUNT(*) INTO interventi_totali
17     FROM MEMBRI_EQUIPAGGIO ME
18     JOIN COINVOLGIMENTI C ON ME.codice = c.equipaggio
19     WHERE ME.codice = codice_membro;
20
21     SELECT COUNT(*) INTO interventi_riusciti
22     FROM MEMBRI_EQUIPAGGIO ME
23     JOIN COINVOLGIMENTI C ON ME.codice = c.equipaggio
24     JOIN Interventi I ON I.CODICE = c.INTERVENTO
25     WHERE ME.codice = codice_membro AND I.esito = 'RIUSCITO';
26
27     IF interventi_totali > 0 THEN
28         m_success_rate := (interventi_riusciti / interventi_totali)
29         * 100;
30     ELSE
31         m_success_rate := 0;
32     END IF;
33
34     SELECT COUNT(missione)
35     INTO p_total_missions
36     FROM Partecipazioni

```

```

36 WHERE membro = codice_membro;
37
38 DBMS_OUTPUT.PUT_LINE('=====');
39 DBMS_OUTPUT.PUT_LINE('ANALISI EFFICIENZA MEMBRO ID: ' ||
    codice_membro);
40 DBMS_OUTPUT.PUT_LINE('=====');
41 DBMS_OUTPUT.PUT_LINE('Nome: ' || nome_membro);
42 DBMS_OUTPUT.PUT_LINE('-----');
43 DBMS_OUTPUT.PUT_LINE('Interventi totali: ' ||
    interventi_totali);
44 DBMS_OUTPUT.PUT_LINE('Interventi riusciti: ' ||
    interventi_riusciti);
45 DBMS_OUTPUT.PUT_LINE('Percentuale di successo: ' ||
    ROUND(m_success_rate, 2) || '%');
46 DBMS_OUTPUT.PUT_LINE('-----');
47 DBMS_OUTPUT.PUT_LINE('Missioni totali: ' || p_total_missions);
48 DBMS_OUTPUT.PUT_LINE('=====');
49 EXCEPTION
50 WHEN NO_DATA_FOUND THEN
51     RAISE_APPLICATION_ERROR(-20001, 'Errore nell''analisi
        dell''efficienza: ' || SQLERRM);
52 END efficienza_membro;

```

```

=====
ANALISI EFFICIENZA MEMBRO ID: 8
=====
Nome: Giacomo Lombardi
-----
Interventi totali: 83
Interventi riusciti: 67
Percentuale di successo: 80.72%
-----
Missioni totali: 8
=====

```

Figura 4: esempio di output (indice membro 8)

5 GESTIONE DELLA SICUREZZA

Con riferimento alle specifiche sulle politiche di sicurezza, si è deciso di creare i seguenti utenti.

Agenzia__dba

Gli vengono conferiti tutti i permessi possibili sulla base di dati.

```
1 CREATE ROLE agenzia_dba IDENTIFIED BY agenzia_dba
2 GRANT DBA TO agenzia_dba
```

Supervisor

E' l'amministratore di sistema, un operatore dell'azienda che deve avere tutti i permessi necessari ad agire sulla base dati al fine di garantire il massimo supporto ai membri dell'equipaggio. Avrà i permessi di esecuzione trigger e procedure, potrà creare trigger e procedure ed inoltre avrà tutti i permessi sulle tabelle(select,insert,uodate). Infine accedendo alla web-app avrà la possibilità di accedere a tutte le informazioni in ogni pagina.

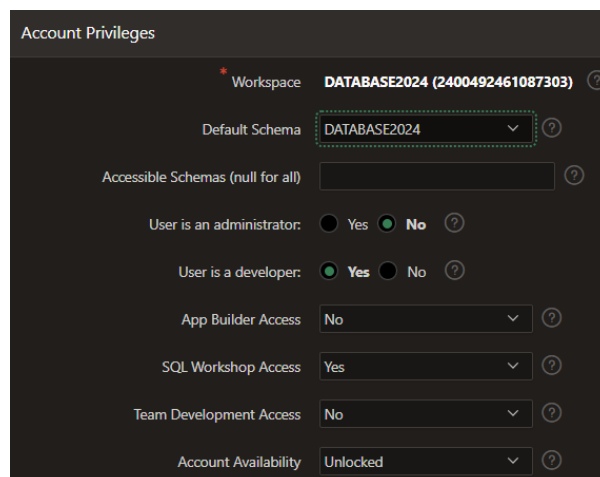


Figura 5: Supervisor

Membri equipaggio

I membri dell'equipaggio risultano essere end user, quindi potranno solo utilizzare la web-app accedendo con credenziali (Matricola, password).definite dal supervisor all'atto dell'inserimento di un nuovo membro.

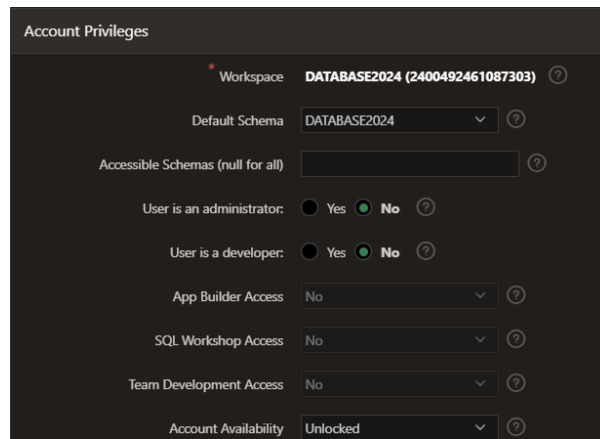


Figura 6: End-User

data-analyst

E' l'utente creato dal dba, al quale viene dato il permesso di connessione al database e di poter fare operazione di select sulla vista dati_membri

6 CREAZIONE WEB APPLICATION

Per l'implementazione dell'applicazione web, il team di progetto ha utilizzato il portale Oracle APEX, che ha permesso di strutturare la suddetta applicazione in più pagine al fine di avere maggiore user-friendliness.

L'applicazione presenta un totale di 8 pagine.

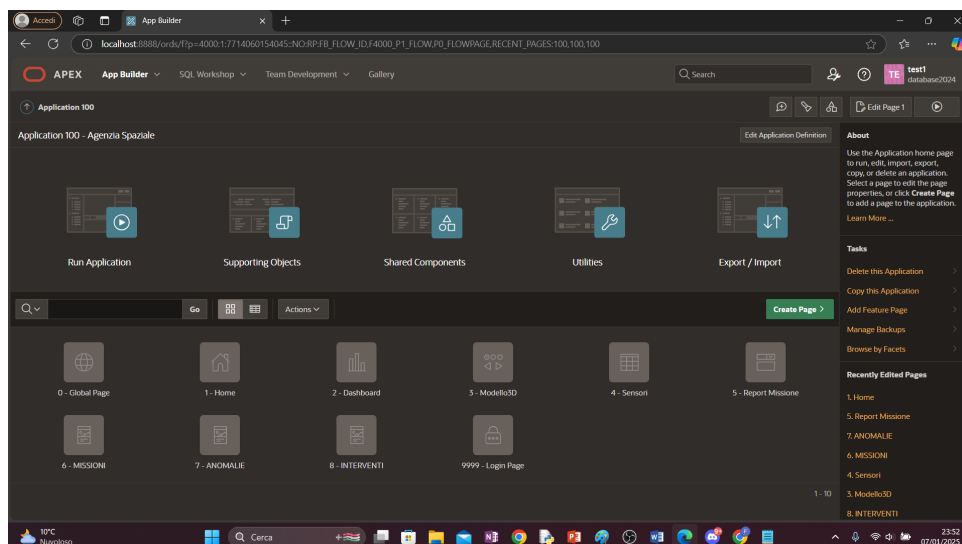


Figura 7: Pagine dell'applicazione

- Pagina di LOGIN

La pagina di **LOGIN** è stata creata associando ad ogni membro dell'equipaggio una matricola univoca all'interno del sistema assegnata all'utente dal reparto competente dell'agenzia al momento della sua iscrizione al programma lunare.

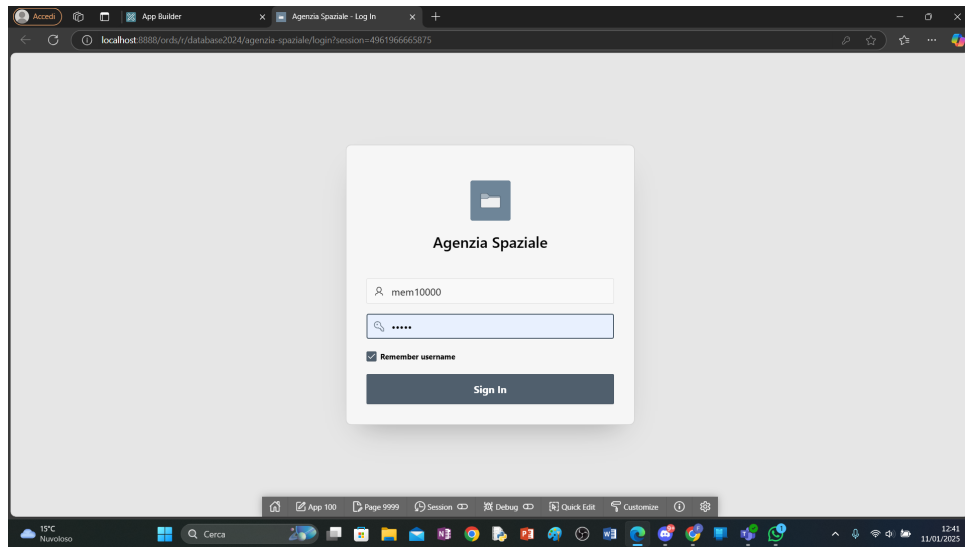


Figura 8: Pagina di LOGIN

- Home page

L'**HOME PAGE** contiene una breve descrizione dell'agenzia proprietaria dell'applicazione.

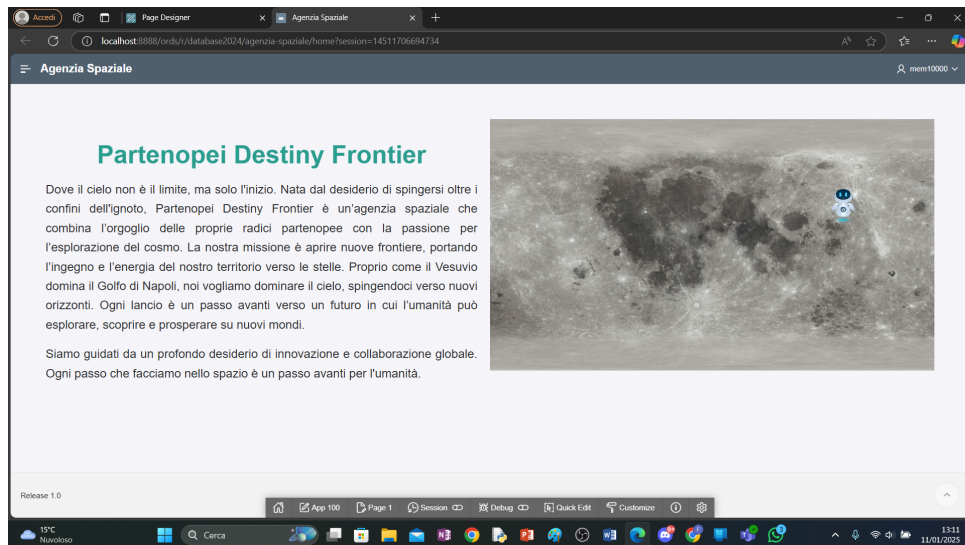


Figura 9: HOME PAGE

- Pagina di DASHBOARD

La pagina di **DASHBOARD** è stata implementata al fine di avere una istantanea per ogni utente dell'applicazione che rappresenti la situazione corrente dei sensori utilizzati nella propria missione (Per utenti che non siano membri dell'equipaggio si ha una rappresentazione customizzata in base alle funzioni degli stessi utenti: e.g. SUPERVISOR).

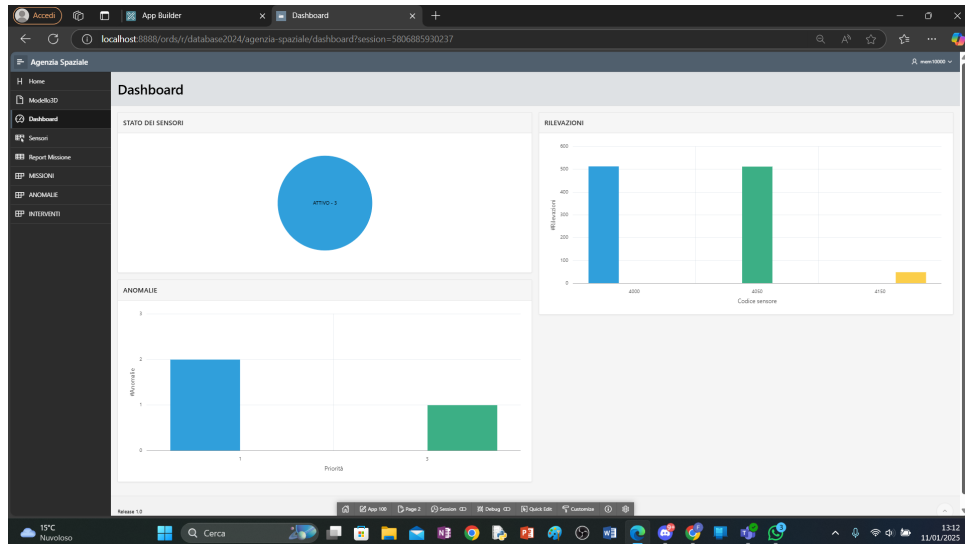


Figura 10: Pagina di DASHBOARD

- **Pagina con il MODELLO 3D**

La pagina con il **MODELLO 3D** permette la rappresentazione spaziale della Luna con una texture realistica (prelevata dal database della NASA) in maniera da poter illustrare in maniera grafica e più efficiente all'utente l'ubicazione fisica dei singoli sensori sulla superficie lunare. I sensori sono marcati nel modello con dei 'pin' verdi che, se cliccati con il cursore, forniscono le informazioni di base del sensore che identificano (ID del sensore, coordinate, tipologia e date di installazione ed ultimo controllo). Il modello prevede uno script interattivo, che permetta all'utente di cambiare l'angolo di visualizzazione della superficie lunare.

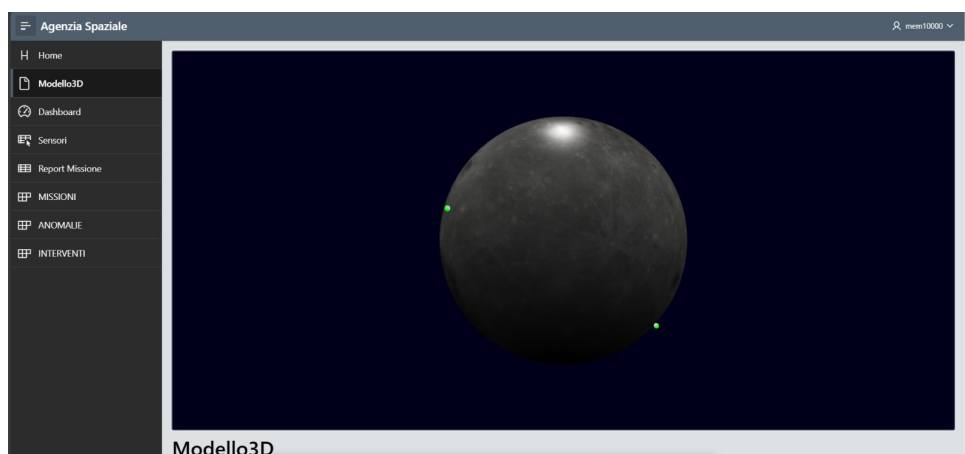


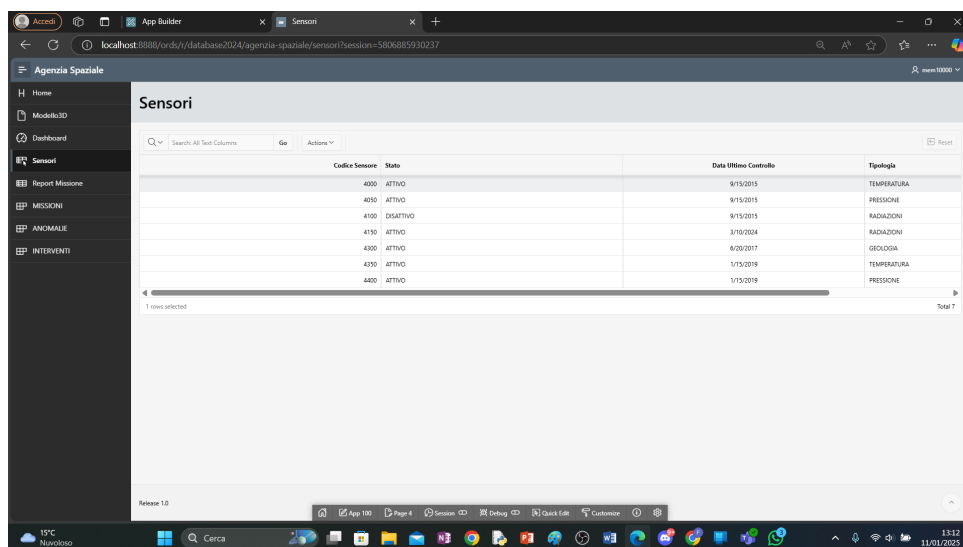
Figura 11: Pagina del MODELLO 3D

- **Pagina dei SENSORI**

Nella pagina dei **SENSORI** sono presenti le informazioni riguardo i sensori associati alla missione nella quale è impegnato l'utente della piattaforma. Le informazioni sono:

- Codice Sensore
- Stato del sensore
- Data di ultimo controllo
- Tipologia del sensore

Ogni dato sui sensori presente in questa pagina è filtrato in modo che siano disponibili ad un determinato utente le informazioni riguardanti solamente la missione nella quale è impegnato, mentre per quanto riguarda l'utente SUPERVISOR sono disponibili le informazioni su tutti i sensori impiegati dall'agenzia.



Codice Sensore	Stato	Data Ultimo Controllo	Tipologia
4000	ATTIVO	9/15/2015	TEMPERATURA
4000	ATTIVO	9/15/2015	PRESSIONE
4000	DISATTIVO	9/15/2015	RADIAZIONI
4150	ATTIVO	9/16/2024	RADIAZIONI
4000	ATTIVO	6/08/2017	GEOLÓGIA
4000	ATTIVO	1/15/2019	TEMPERATURA
4000	ATTIVO	1/15/2019	PRESSIONE

Figura 12: Pagina dei SENSORI

- **Pagina dei REPORT di missione**

All'interno della pagina dei **REPORT di missione** sono presenti due funzionalità: visualizzazione dei report effettuati in precedenza e creazione di un report.

Per quanto riguarda la *visualizzazione* sono mostrati i dati riguardo:

- Codice della missione sulla quale si vuole scrivere il report
- Obbiettivo della missione
- Cognome dell'utente che effettua il report
- Contenuto del report (Stato)

Per quanto riguarda la *creazione* dei report invece è presente un'interfaccia da riempire con i dati che si vuole vengano registrati.

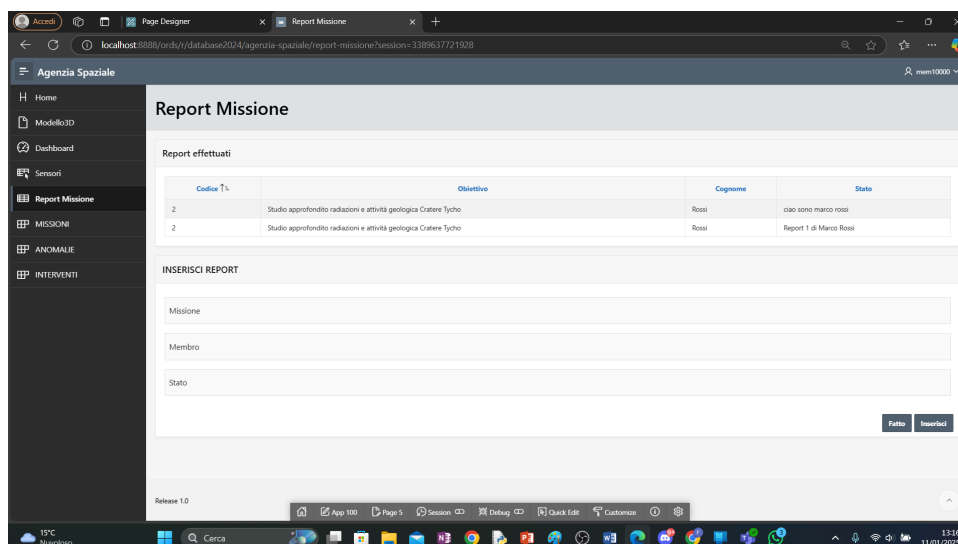


Figura 13: Pagina dei REPORT

• Pagina delle MISSIONI

La pagina delle **MISSIONI** presenta al suo interno una lista delle missioni alle quali l'utente partecipa o ha partecipato in passato. La lista indica per ogni missione:

- Nome della missione
- Data di inizio
- Data di termine
- Stato della missione

Inoltre la lista può essere ordinata in base al criterio che si preferisce.

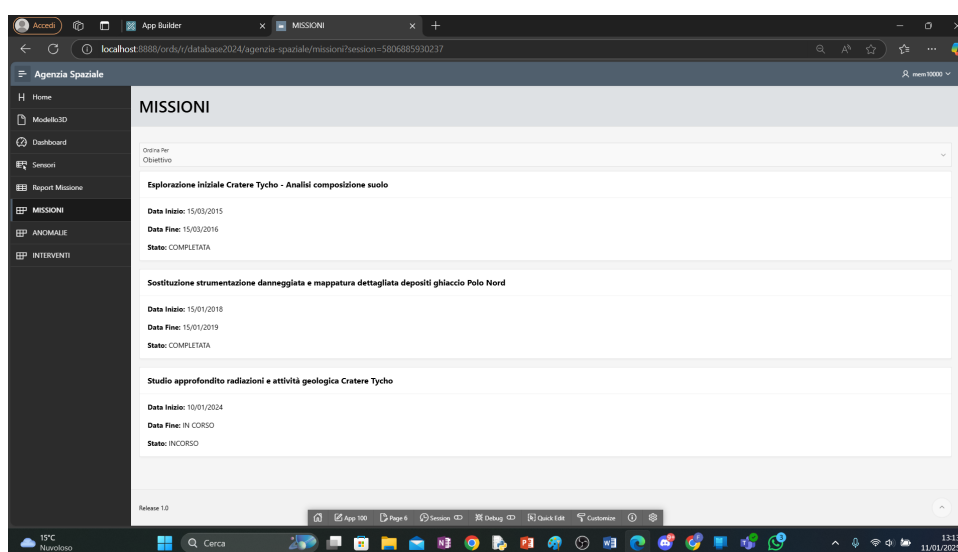


Figura 14: Pagina delle MISSIONI

- **Pagina delle ANOMALIE**

Nella pagina delle **ANOMALIE** sono mostrate tutte le anomalie dei sensori che sono impiegati nella missione a cui partecipa l'utente.

Le informazioni memorizzate sono:

- Codice univoco dell'anomalia
- Sensore che ha subito l'anomalia
- Data in cui si è verificata l'anomalia
- Causa dello stato anomalo del sensore

Inoltre le informazioni possono essere ordinate in base a diversi criteri, come per le missioni.

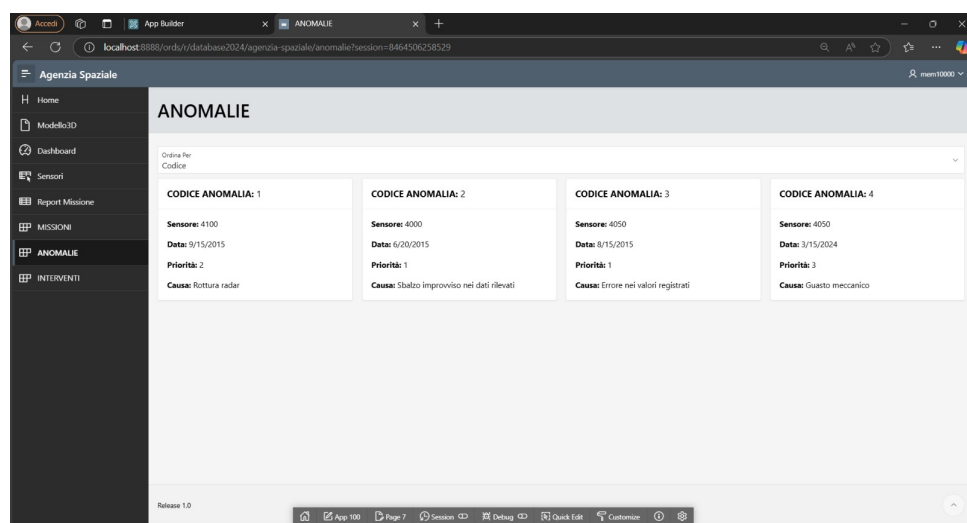


Figura 15: Pagina delle ANOMALIE

- **Pagina degli INTERVENTI**

All'interno della pagina degli **INTERVENTI** sono presenti le informazioni chiave sugli interventi effettuati dall'utente su sensori malfunzionanti.

Le informazioni memorizzate sono:

- Codice dell'intervento, univoco nel sistema
- Descrizione dell'operazione
- Esito dell'intervento
- Data dell'intervento
- Codice dell'anomalia che si vuole risolvere con l'intervento

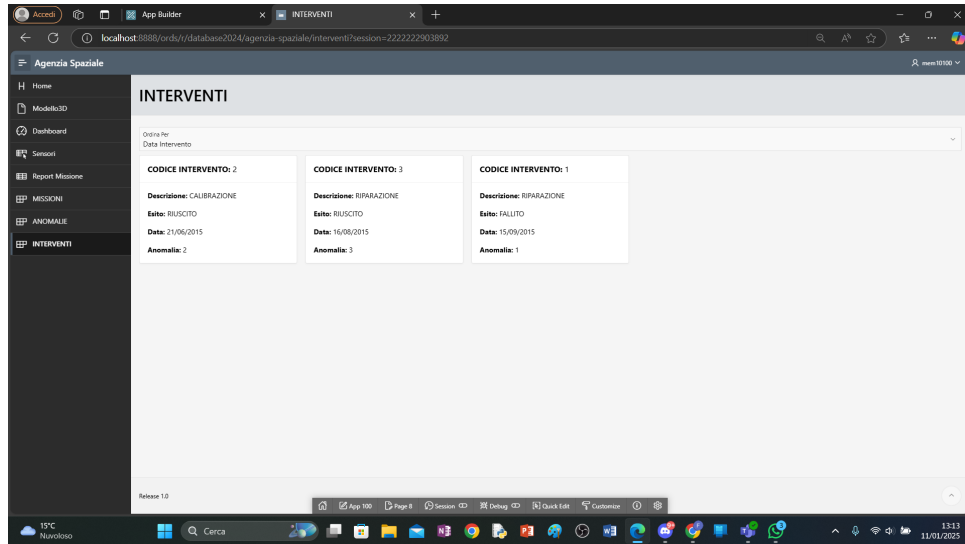


Figura 16: Pagina degli INTERVENTI

7 CLUSTERING DEI DATI DEI MEMBRI DELL'AGENZIA SPAZIALE

L'obiettivo di questo lavoro è sviluppare e presentare un modello di clustering applicato ai dati dei membri di un team operativo, utilizzando metriche chiave come il tasso di successo, l'efficienza e la complessità operativa. Il progetto si propone di fornire un supporto pratico nella selezione dei membri per le missioni, identificando gruppi di individui con caratteristiche simili. Questo approccio permette di creare team equilibrati e ottimizzati, favorendo una distribuzione efficace delle competenze e dei carichi di lavoro.

7.1 Specifiche sui cluster

Ogni cluster identifica un gruppo di membri del team con caratteristiche comuni in base alle metriche calcolate:

- **Tasso di Successo:** Percentuale di interventi riusciti rispetto a quelli tentati, calcolata come:

$$\text{Tasso di Successo} = \frac{\text{Numero di Interventi Riusciti}}{\text{Numero Totale di Interventi}} \times 100$$

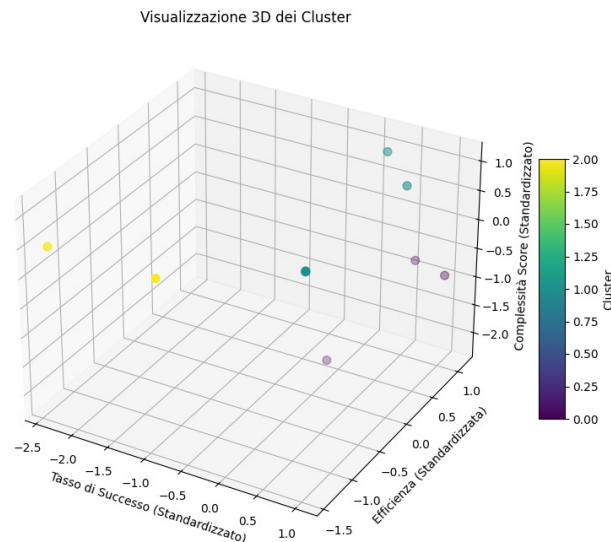
- **Efficienza:** Numero di interventi riusciti rispetto alla durata media delle missioni, calcolata come:

$$\text{Efficienza} = \frac{\text{Numero di Interventi Riusciti}}{\text{Media della Durata delle Missioni}}$$

- **Complessità Score:** Misura del carico operativo, calcolata come:

$$\text{Complessità Score} = \text{Media della Durata delle Missioni} \times \text{Numero di Ruoli}$$

La figura seguente illustra come i membri del team siano stati raggruppati in tre cluster distinti, basandosi su queste metriche. Questa rappresentazione tridimensionale permette di cogliere visivamente i pattern identificati durante l'analisi.



7.2 Presentazione del Codice

Per condurre l'analisi sui dati dei membri del team e ottenere i risultati discussi nei paragrafi precedenti, è stato sviluppato un codice Python. Di seguito, ogni funzione implementata nel codice viene spiegata singolarmente.

7.2.1 Connessione al Database

La funzione `connessione_db()` si occupa di stabilire una connessione al database Oracle utilizzando le credenziali fornite. Questa funzione garantisce l'accesso ai dati memorizzati, necessari per l'analisi successiva. In caso di errore, il sistema stampa un messaggio di errore e restituisce `None`.

La connessione viene stabilita grazie alla libreria `Oracledb`. Puoi connetterti direttamente al database Oracle 12.1 o versioni successive senza la necessità di librerie client di Oracle.

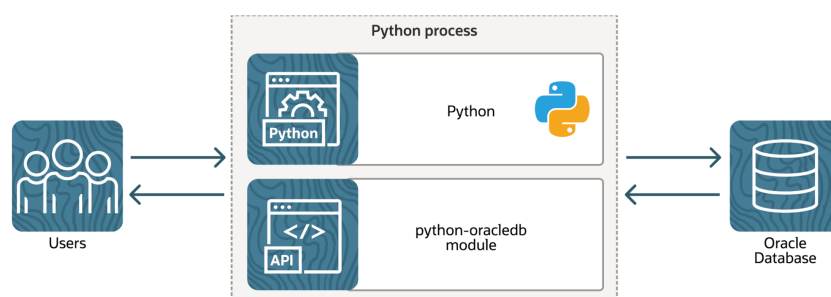


Figura 17: Architettura di connessione al database


```

11         )
12         return connection
13     except oracledb.DatabaseError as e:
14         print("Errore durante la connessione:", e)
15         return None
16
17     # Funzione per ottenere i dati dal database
18     def get_data(connection):
19         query = "SELECT * FROM ADMIN1.dati_membri"
20         cursor = connection.cursor()
21         cursor.execute(query)
22         rows = cursor.fetchall()
23         columns = [col[0] for col in cursor.description]
24         return pd.DataFrame(rows, columns=columns)
25
26     # Analisi avanzata dei membri del team
27     def analisi_membri(df):
28         analysis = pd.DataFrame({
29             'CODICE_MEMBRO': df['CODICE_MEMBRO'],
30             'Tasso_Successo': ((df['NUM_INTERVENTI_RIUSCITI'] /
31                                df['NUM_INTERVENTI']) * 100).round(2),
32             'Efficienza': (df['NUM_INTERVENTI_RIUSCITI'] /
33                             df['MEDIA_DURATA_MISSIONI']).round(2),
34             'Complessità_Score': (df['MEDIA_DURATA_MISSIONI'] *
35                                   df['NUMERO_RUOLI']).round(2)
36         })
37
38         # Rimozione dei dati non validi
39         analysis = analysis.dropna()
40         analysis = analysis[~(analysis['Tasso_Successo'].isna()) &
41                               (analysis['Efficienza'] > 0)]
42
43         # Normalizzazione dei dati
44         scaler = StandardScaler()
45         scaled_features =
46             scaler.fit_transform(analysis[['Tasso_Successo',
47                                             'Efficienza', 'Complessità_Score']])
48
49         kmeans = KMeans(n_clusters=3, random_state=42)
50         analysis['Cluster'] = kmeans.fit_predict(scaled_features)
51
52         return analysis, kmeans, scaled_features
53
54     # Esporta i risultati in un file CSV
55     def esporta_risultati_csv(analysis,
56                               filename="risultati_analisi.csv"):
57         try:
58             analysis.to_csv(filename, index=False)
59             print(f"Risultati salvati con successo nel file {filename}")
60         except Exception as e:
61             print(f"Errore durante il salvataggio del file: {e}")

```

```

55
56
57 # Uso delle funzioni
58 connection = connessione_db()
59 if connection:
60     df = get_data(connection)
61     df = df.fillna(df.mean())
62     analysis, kmeans, scaled_features = analisi_membri(df)
63     esporta_risultati_csv(analysis)
64     connection.close()

```

CODICE_MEMBRO	Tasso_Successo	Efficienza	Complessità_Score	Cluster
2	66.67	0.01	731.0	2
3	50.0	0.01	731.0	2
6	84.34	0.36	587.0	0
7	89.47	0.35	582.6	0
8	80.72	0.34	796.0	1
9	85.51	0.31	766.67	1
10	76.71	0.26	432.8	0
1	82.49	0.11	731.0	1
5	82.49	0.11	730.0	1

Figura 19: file .csv generato