Università di Catania

Dipartimento di Matematica e Informatica

Progetto Corso Basi di Dati

a.a. 2023/24

Gestione di una Compagnia di Software

A cura di Brancato Simone Alfio

Specifiche sui dati

Si vuole progettare un sistema per la gestione di una compagnia di sviluppo software. Il database conterrà informazioni su progetti, personale, clienti e versioni del software. I clienti sono aziende che chiedono alla compagnia lo sviluppo di progetti, ogni cliente viene memorizzato nel sistema specificando partita iva, nome dell'azienda, sede, recapiti. Si terrà conto del fatto che i membri del personale possano essere dipendenti dell'azienda o consulenti esterni. Per ogni membro del personale teniamo traccia del'id, nome, cognome, ruolo, stipendio. Memorizziamo in un registro stipendi l'emissione mensile degli stipendi degli sviluppatori, specificandone data di emissione del bonifico e l'importo. Il personale dell'azienda si divide in team di sviluppo e ogni team ha un codice, un nome, una descrizione e il numero dei suoi membri. Ogni team può essere assegnato a uno o più progetti. Per ogni progetto memorizziamo il nome, descrizione, data di inizio, stato di avanzamento. Ad ogni progetto possono lavorare uno o più team di sviluppo. Per portare a compimento ognuno dei progetti serve la realizzazione di uno o più sistemi software. Per ogni software sviluppato dall'azienda teniamo traccia del nome, descrizione e progetto a cui fa riferimento. I software vengono rilasciati periodicamente sottoforma di versioni e ne teniamo traccia dello storico memorizzando versione, data di rilascio e descrizione e il numero di bug associati ad esse. Ogni bug deve essere accompagnato da una descrizione del problema, livello di gravità, stato di risoluzione.

Analisi dei requisiti

Termine	Descrizione	Sinonimi	Termini collegati
Personale	Gruppo di persone che lavorano per la compagnia di sviluppo software, ognuno di essi può essere un dipendente o un consulente esterno	Sviluppatori	Dipendente, consulente esterno
Team	Gruppo di sviluppatori a cui vengono assegnati progetti su cui lavorare		Personale, progetto
Progetto	Lavoro commissionato da un cliente dell'azienda		Team, software, cliente
Cliente	Azienda che commissiona la realizzazione di un progetto		Progetto
Software	Sistema software codificato dai team di sviluppo al fine di soddisfare i requisiti dei progetti		Progetto, team, versione software
Versione software	Versioni software rilasciate dai team di sviluppo, rispecchiano lo stato di avanzamento e l'evoluzione dei software sviluppati		Software, bug

Termine	Descrizione	Sinonimi	Termini collegati
Bug	Problemi software individuati dagli sviluppatori. Sono associati ad una specifica versione rilasciata	Problema	Versione software
Registro degli stipendi	Registro pagamenti degli stipendi del personale dell'azienda		Personale

Dati di carattere generale
Si vuole progettare un sistema per la gestione di una compagnia di sviluppo software. Il database conterrà informazioni su progetti, personale, clienti e versioni del software.

Dati sui clienti
I clienti sono aziende che chiedono alla compagnia lo sviluppo di progetti, ogni cliente viene memorizzato nel sistema specificando partita iva, nome dell'azienda, sede, recapiti.

Dati sul personale
Si terrà conto del fatto che i membri del personale possano essere dipendenti dell'azienda o consulenti esterni. Per ogni membro del personale teniamo traccia del'id, nome, cognome, ruolo, stipendio.

Dati sugli stipendi
Memorizziamo in un registro stipendi l'emissione mensile degli stipendi degli sviluppatori, specificandone data di emissione del bonifico e l'importo.

Dati sui team
Il personale dell'azienda si divide in team di sviluppo e ogni team ha un codice, un nome e una descrizione e il numero dei suoi membri. Ogni team può essere assegnato a uno o più progetti.

Dati sui progetti
Per ogni progetto memorizziamo il nome, descrizione, data di inizio, stato di avanzamento. Ad ogni progetto possono lavorare uno o più team di sviluppo. Per portare a compimento ognuno dei progetti serve la realizzazione di uno o più sistemi software.

Dati sui software
Per ogni software sviluppato dall'azienda teniamo traccia del nome, descrizione e progetto a cui fa riferimento.

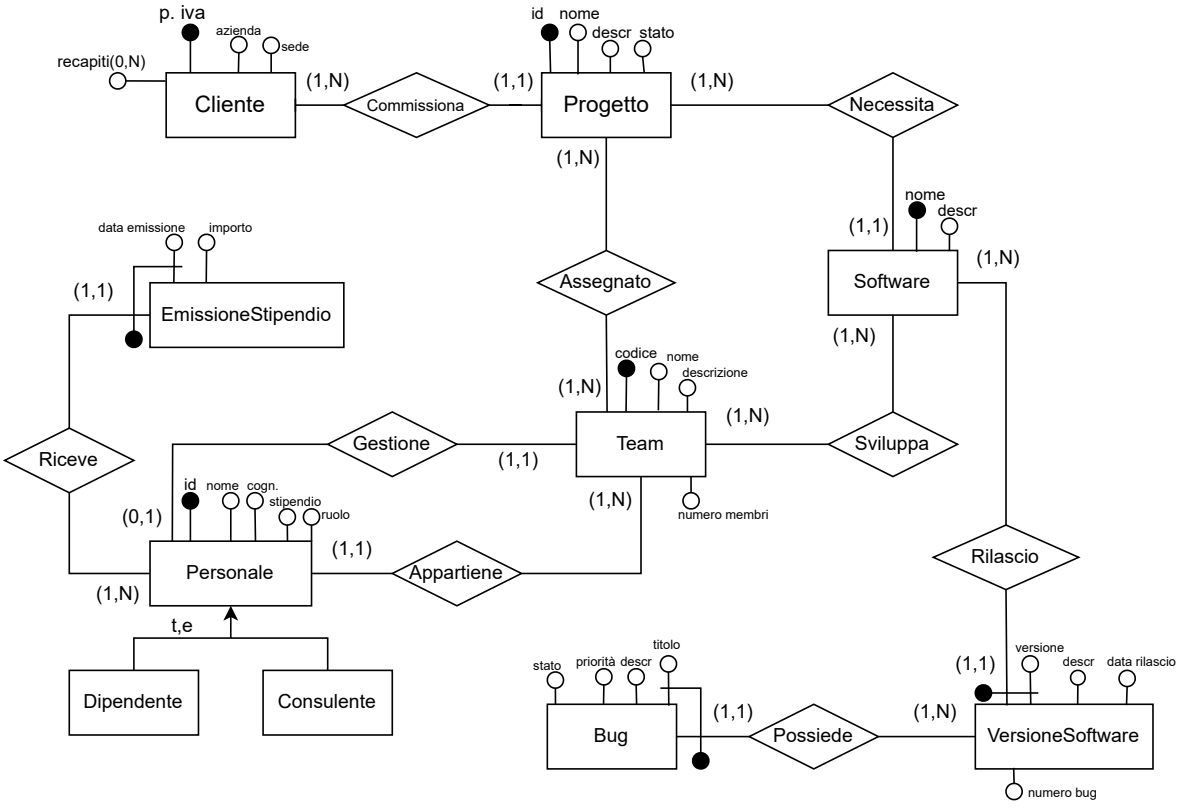
Dati sulle versioni dei software
I software vengono rilasciati periodicamente sottoforma di versioni e ne teniamo traccia dello storico memorizzando versione, data di rilascio e descrizione e il numero di bug associati ad esse.

Dati sui bug
Ogni bug deve essere accompagnato da un titolo, una descrizione del problema, livello di gravità, stato di risoluzione.

Progettazione concettuale

Possiamo costruire il seguente schema scheletro composto da tre entità e due associazioni per descrivere in maniera sintetica le principali caratteristiche del database che vogliamo realizzare: i clienti commissionano all'azienda la realizzazione di progetti, che verranno assegnati ai team di sviluppo.

Affiniamo il precedente schema al fine di raggiungere gli obiettivi individuati durante l'analisi dei requisiti



Dizionario dati entità

Entità	Descrizione	Attributi	Identificatore
Cliente	Azienda cliente della compagnia di sviluppo software	partitaiva, azienda, sede, recapiti	partitaiva
Progetto	Insieme di attività correlate, hanno lo scopo di raggiungere un obiettivo	id, nome, descrizione, stato	id
Team	Gruppo di individui che lavorano insieme al fine di raggiungere obiettivi comuni	codice, nome, descrizione	codice
Software	Sistema informatico che svolge specifici compiti	nome, descrizione	nome
Personale	Insieme degli individui che lavorano nella compagnia di sviluppo software	id, nome, cognome, stipendio, ruolo	id
Emissione Stipendio	Registro pagamenti degli stipendi del personale della compagnia di sviluppo software	data emissione, importo	data emissione, idpersona (tramite relazione Riceve)
Versione Software	Registro delle versioni software rilasciate	versione, descrizione, data rilascio	versione, nome software (tramite relazione Rilascio)
Bug	Problema riscontrato dagli sviluppatori in una particolare versione di uno specifico software	titolo, descrizione, priorità, stato	titolo, nome software (tramite relazione Possiede), versione software (tramite relazione Possiede)

Dizionario dati Relazioni

Relazione	Entità partecipanti	Descrizione	Attributi
Commissiona	Cliente, Progetto	I clienti commissionano alla compagnia la realizzazione di progetti	
Assegnato	Progetto, Team	Ogni progetto viene assegnato a dei team di sviluppo	
Necessita	Progetto, Software	Ogni progetto per essere realizzato necessita la codifica di determinati software	
Sviluppa	Team, Software	I team sviluppano i software dei progetti vengono assegnati	
Rilascio	Software, Versione Software	I software vengono rilasciati periodicamente sottoforma di versioni	
Possiede	Versione Software, Bug	Ogni versione software possiede determinati bug	
Appartiene	Personale, Team	Ogni membro del personale appartiene ad un determinato team	
Gestione	Team, Personale	Ogni team viene gestito da un membro del personale	
Riceve	Personale, Emissione Stipendio	Il personale riceve mensilmente l'emissione del pagamento dello stipendio	

Specifiche sulle operazioni

Si vogliono implementare le seguenti operazioni:

- **O1** Inserire un nuovo membro del personale (100 volte al mese)
- **O2** Emettere stipendio di ogni membro del personale (1 volta al mese)
- **O3** Inserire un nuovo cliente (10 volte al mese)
- **O4** Inserire un nuovo progetto (15 volte al mese)
- **O5** Modificare lo stato di un progetto (12 volte al mese)
- **O6** Inserire un nuovo software (20 volte al mese)
- **O7** Rilasciare nuova versione software (30 volte al giorno)
- **O8** Registrare un nuovo bug (50 volte al giorno)
- **O9** Modificare lo stato di un bug (40 volte al giorno)
- **O10** Visualizzare il numero dei membri di un team (2 volte al mese)
- **O11** Calcolare il numero bug individuati in una versione di un software (4 volte al mese)
- **O12** Visualizzare i clienti che hanno commissionato più progetti (1 volta al mese)
- **O13** Visualizzare i team che lavorano a più progetti (2 volte al mese)

Tavola dei volumi

Valutiamo adesso la possibile mole di dati per ogni concetto del database, immaginando lo stato del database dopo qualche anno di utilizzo.

Concetto	Tipo	Volume
Cliente	E	200
Progetto	E	1.000
Team	E	200
Software	E	5.000
Personale	E	2.000
Emissione Stipendio	E	70.000
Versione Software	E	50.000
Bug	E	500.000
Commissiona	R	1.000
Assegnato	R	1.300
Necessita	R	5.000
Sviluppa	R	5.200
Rilascio	R	50.000
Possiede	R	500.000
Appartiene	R	2.000
Gestione	R	200
Riceve	R	70.000

Tavola delle frequenze

Operazione	Descrizione	Frequenza	Tipo
O1	Inserire un nuovo membro del personale	100/mese	I
O2	Emettere stipendio di ogni membro del personale	1/mese	I
O3	Inserire un nuovo cliente	10/mese	I
O4	Inserire un nuovo progetto	15/mese	I
O5	Modificare lo stato di un progetto	12/mese	I
O6	Inserire un nuovo software	20/mese	I
O7	Rilasciare nuova versione software	30/giorno	I
O8	Registrare un nuovo bug	50/giorno	I
O9	Modificare lo stato di un bug	40/giorno	I
O10	Visualizzare il numero dei membri di un team	2/mese	B
O11	Calcolare il numero bug individuati in una versione di un software	4/mese	B
O12	Visualizzare i clienti che hanno commissionato più progetti	1/mese	B
O13	Visualizzare i team che lavorano a più progetti	2/mese	B

Analisi delle ridondanze e ristrutturazione schema

Per poter tradurre lo schema E-R verso il modello relazionale dobbiamo ristrutturarlo. Questa fase prevede l'analisi delle ridondanze, eliminazione delle generalizzazioni, scelta identificatori e partizionamento o accorpamento di entità e associazioni.

Procedendo con l'analisi delle ridondanze possiamo notare che all'interno dello schema sono presenti due ridondanze, ovvero:

1. `Numero membri` all'interno dell'entità `Team`

   Questa ridondanza entra in gioco quando utilizziamo le operazioni:

   - **O1** Inserire un nuovo membro del personale (100/mese)
   - **O10** Visualizzare il numero dei membri di un team (2/mese)

   Mantenendo la ridondanza per eseguire O1 dovremo fare una scrittura in `Personale`, una lettura e una scrittura in `Team` quindi considerando una scrittura come due letture avremo 5 letture ad ogni esecuzione e quindi 500 letture ogni mese. Per eseguire O10 dovremo fare semplicemente una lettura in `Team`, quindi 2 letture ogni mese. Sommando le letture delle due operazioni faremo **502 letture al mese**.

Eliminando la ridondanza per eseguire O1 dovremo fare semplicemente una scrittura in `Personale`, quindi 200 letture ogni mese. Per eseguire O10 stavolta dovremo considerare il numero medio di partecipanti per ogni team che è circa 10, ne segue che mediamente dovremo fare 10 letture in `Personale` due volte al mese, quindi 20 letture. Sommando le letture delle due operazioni faremo **220 letture al mese**.

Quindi conviene rimuovere la ridondanza `Numero membri` da `Team`.

2. `Numero bug` all'interno dell'entità `Versione Software`

   Questa ridondanza entra in gioco quando utilizziamo le operazioni:

   - **O8** Registrare un nuovo bug (50/giorno ➡️ 1500/mese)
   - **O11** Calcolare il numero bug individuati in una versione di un software (4/mese)

   Mantenendo la ridondanza per eseguire O8 dovremo fare una scrittura in `Bug`, una lettura e una scrittura in `Versione Software` quindi considerando una scrittura come due letture avremo 5 letture ad ogni esecuzione, e quindi ogni mese avremo 7500 letture. Per eseguire O11 basterà fare una lettura in Versione Software, quindi 4 letture al mese. Sommando le letture delle due operazioni faremo **7504 letture al mese**.

   Eliminando la ridondanza per eseguire O8 basterà fare una scrittura in `Bug`, quindi due letture ad ogni esecuzione e 3000 letture al mese. Per eseguire O11 stavolta dobbiamo considerare il numero medio di bug per ogni versione software rilasciata e questo è circa 10. Quindi faremo mediamente 10 letture in `Bug`. Sommando le letture delle due operazioni faremo **3010 letture al mese**.

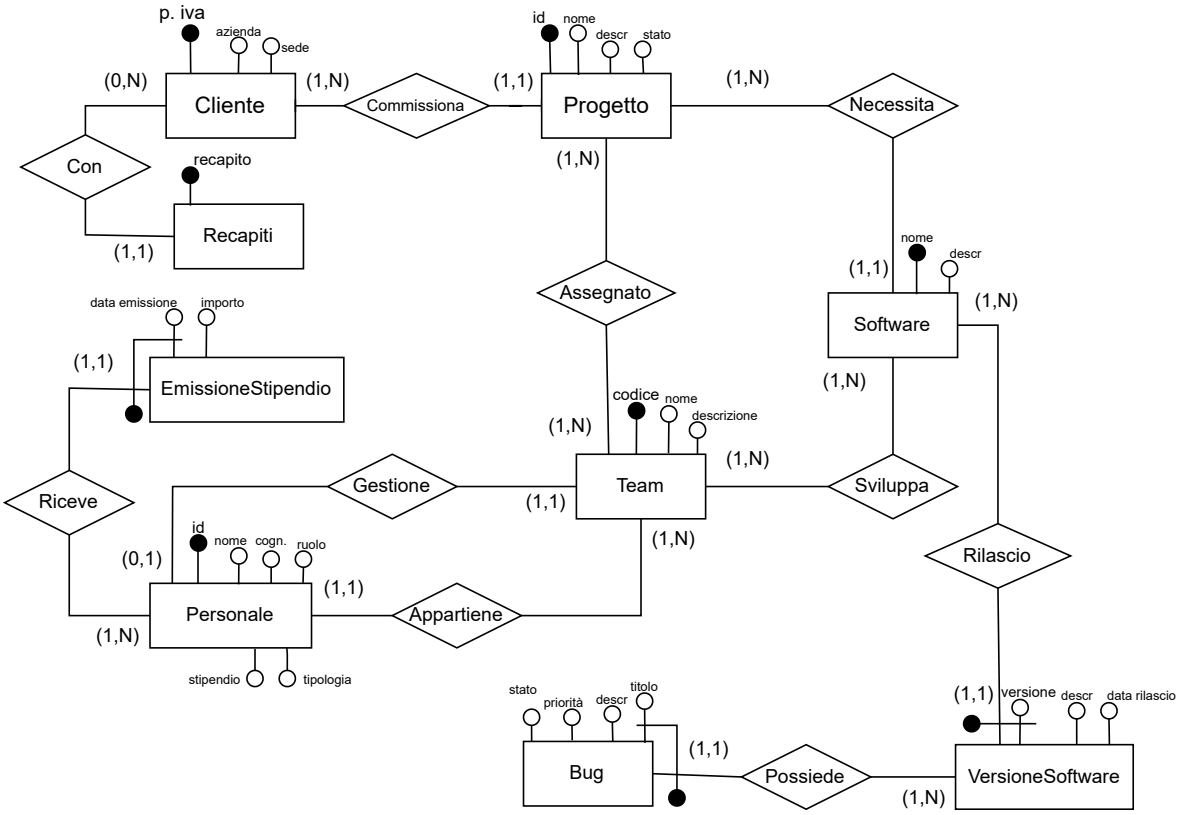
   Quindi conviene rimuovere la ridondanza `Numero bug` da `Versione Software`.

Procediamo col **modificare l'attributo multivalore** opzionale Recapiti all'interno dell'entità Cliente, partizioniamo quindi questa entità in due:

1. Cliente con gli stessi attributi di partenza meno l'attributo multivalore Recapiti
2. Recapiti con attributo Recapito (Nome scelto volutamente generico, identificatore)

Procediamo con **l'eliminazione delle gerarchie**, ne è presente solo una ovvero Personale ⬅️ (Dipendente, Consulente) che è una generalizzazione totale ed esclusiva. Dato che non vi sono particolari differenze tra gli attributi delle entità Dipendente e Consulente possiamo procedere con un collasso verso l'alto (in questo caso è conveniente dato che non avremo valori nulli) e aggiungendo un attributo all'entità Personale ovvero Tipologia, al fine di specificare se un membro del personale è un dipendente o un consulente.

Schema ristrutturato



Progettazione logica nel modello relazionale

Siano le **chiavi primarie** e le chiavi esterne. Arriviamo alla seguente progettazione logica:

Cliente(**p.iva**, azienda, sede)
Recapiti(**recapito**, azienda)
Progetto(**id**, nome, descrizione, stato, azienda)
Team(**codice**, nome, manager, descrizione)
ProgettoTeam(**idprogetto, codiceteam**)
Software(**nome**, descrizione, progetto)
SoftwareTeam(**nomesoftware, codiceteam**)
VersioneSoftware(**nomesoftware**, **versione**, descrizione, datarilascio)
Bug(**titolo, nomesoftware, versionesoftware**, descrizione, priorità, stato)
Personale(**id**, nome, cognome, ruolo, team, stipendio, tipologia)
EmissioneStipendio(**impiegato, dataemissione**, importo)

Implementazione tabelle

```
CREATE TABLE Cliente (
    partitaiva BIGINT NOT NULL,
    azienda VARCHAR(30) NOT NULL,
    sede VARCHAR(50) NOT NULL,
    PRIMARY KEY(partitaiva)
);
```

```
CREATE TABLE Recapiti (
    recapito VARCHAR(40) NOT NULL,
    azienda BIGINT NOT NULL,
    PRIMARY KEY(recapito),
    FOREIGN KEY(azienda) REFERENCES Cliente(partitaiva)
);
```

```
CREATE TABLE Progetto (
    id INT NOT NULL,
    nome VARCHAR(20) NOT NULL,
    descrizione VARCHAR(70),
    stato VARCHAR(20) NOT NULL,
    azienda BIGINT NOT NULL,
    PRIMARY KEY(id),
    FOREIGN KEY(azienda) REFERENCES Cliente(partitaiva)
);
```

```
CREATE TABLE Software(
    nome VARCHAR(20) NOT NULL,
    descrizione VARCHAR(50),
    progetto INT NOT NULL,
    PRIMARY KEY (nome),
    FOREIGN KEY(progetto) REFERENCES Progetto(id)
);
```

```
CREATE TABLE Team (
    codice INT NOT NULL,
    nome VARCHAR(20) NOT NULL,
    manager INT NOT NULL,
    descrizione VARCHAR(70) NOT NULL,
    PRIMARY KEY(codice)
);
```

```
CREATE TABLE ProgettoTeam(
    idprogetto INT NOT NULL,
    codiceteam INT NOT NULL,
    PRIMARY KEY(idprogetto, codiceteam),
    FOREIGN KEY(idprogetto) REFERENCES Progetto(id),
    FOREIGN KEY(codiceteam) REFERENCES Team(Codice)
);
```

```sql
CREATE TABLE Personale(
    id INT NOT NULL,
    nome VARCHAR(20) NOT NULL,
    cognome VARCHAR(20) NOT NULL,
    ruolo VARCHAR(20) NOT NULL,
    team INT NOT NULL,
    stipendio INT NOT NULL,
    tipologia VARCHAR(30) NOT NULL CHECK(tipologia="Dipendente" OR
tipologia="Consulente"),
    PRIMARY KEY(id),
    FOREIGN KEY(team) REFERENCES Team(codice)
);

ALTER TABLE Team ADD CONSTRAINT FOREIGN KEY(manager) REFERENCES Personale(id);
```

```sql
CREATE TABLE SoftwareTeam (
    nomesoftware VARCHAR(20) NOT NULL,
    codiceteam INT NOT NULL,
    PRIMARY KEY(nomesoftware, codiceteam),
    FOREIGN KEY(nomesoftware) REFERENCES Software(nome),
    FOREIGN KEY(codiceteam) REFERENCES Team(Codice)
);
```

```sql
CREATE TABLE VersioneSoftware (
    nomesoftware VARCHAR(20) NOT NULL,
    versione VARCHAR(10) NOT NULL,
    descrizione VARCHAR(50),
    datarilascio DATE NOT NULL,
    PRIMARY KEY(nomesoftware, versione),
    FOREIGN KEY(nomesoftware) REFERENCES Software(nome)
);
```

```sql
CREATE TABLE Bug (
    titolo VARCHAR(40) NOT NULL,
    nomesoftware VARCHAR(20) NOT NULL,
    versionesoftware VARCHAR(10) NOT NULL,
    descrizione VARCHAR(50),
    priorità INT NOT NULL,
    stato VARCHAR(20) NOT NULL,
    PRIMARY KEY(titolo, nomesoftware, versionesoftware),
    FOREIGN KEY(nomesoftware, versionesoftware) REFERENCES
VersioneSoftware(nomesoftware, versione)
);
```

```sql
CREATE TABLE EmissioneStipendio (
    impiegato INT NOT NULL,
    dataemissione DATE NOT NULL,
    importo INT NOT NULL,
    PRIMARY KEY(impiegato, dataemissione),
    FOREIGN KEY(impiegato) REFERENCES Personale(id)
);
```

Implementazione operazioni

O1: Inserire un nuovo membro del personale

```
INSERT INTO Personale VALUES
(5564, "Fabio", "Fazio", "Data Analyst", 355, 2200, "Dipendente");
```

O2: Emettere stipendio di ogni membro del personale

```
INSERT INTO EmissioneStipendio(impiegato, dataemissione, importo)
SELECT p.id, "2024-08-01" , p.stipendio
FROM Personale p
```

O3: Inserire un nuovo cliente

```
INSERT INTO Cliente VALUES
(56477745699, "STMicroelectronics", "via Termini 66, Roma");
```

O4: Inserire un nuovo progetto

```
INSERT INTO Progetto VALUES
(899, "Embedded AMD", "Sviluppo software embedded per AMD", "In corso",
47428823111);
```

O5: Modificare lo stato di un progetto

```
UPDATE Progetto p
SET p.stato = "Completato"
WHERE p.id = 102;
```

O6: Inserire un nuovo software

```
INSERT INTO Software VALUES
("FrontendTN", "Frontend per TourismNOW", 101);
```

O7: Rilasciare nuova versione software

```
INSERT INTO VersioneSoftware VALUES
("WhatsAds", "1.6", "Aggiunto Facade", '2024-03-21');
```

O8: Registrare un nuovo bug

```
INSERT INTO Bug VALUES
("Facade error", "WhatsAds", "1.6", "Compile error linea 61 modulo Facade", 2,
"Aperto");
```

O9: Modificare lo stato di un bug

```
UPDATE Bug b
SET b.stato = "Risolto"
WHERE b.titolo = "Dati inconsistenti"
AND b.nomesoftware = "InstaAnalysis"
AND b.versionesoftware = "1.2";
```

O10: Visualizzare il numero dei membri di un team

```
SELECT t.nome as Team, COUNT(*) as NumeroMembri
FROM Personale p, Team t
WHERE p.team = t.codice
AND t.codice = 355;
```

O11: Calcolare il numero bug individuati in una versione di un software

```
SELECT b.nomesoftware as Software, b.versionesoftware as Versione, COUNT(*) as
NumeroBug
FROM Bug b
WHERE b.nomesoftware = "InstaAnalysis"
AND b.versionesoftware = "1.2";
```

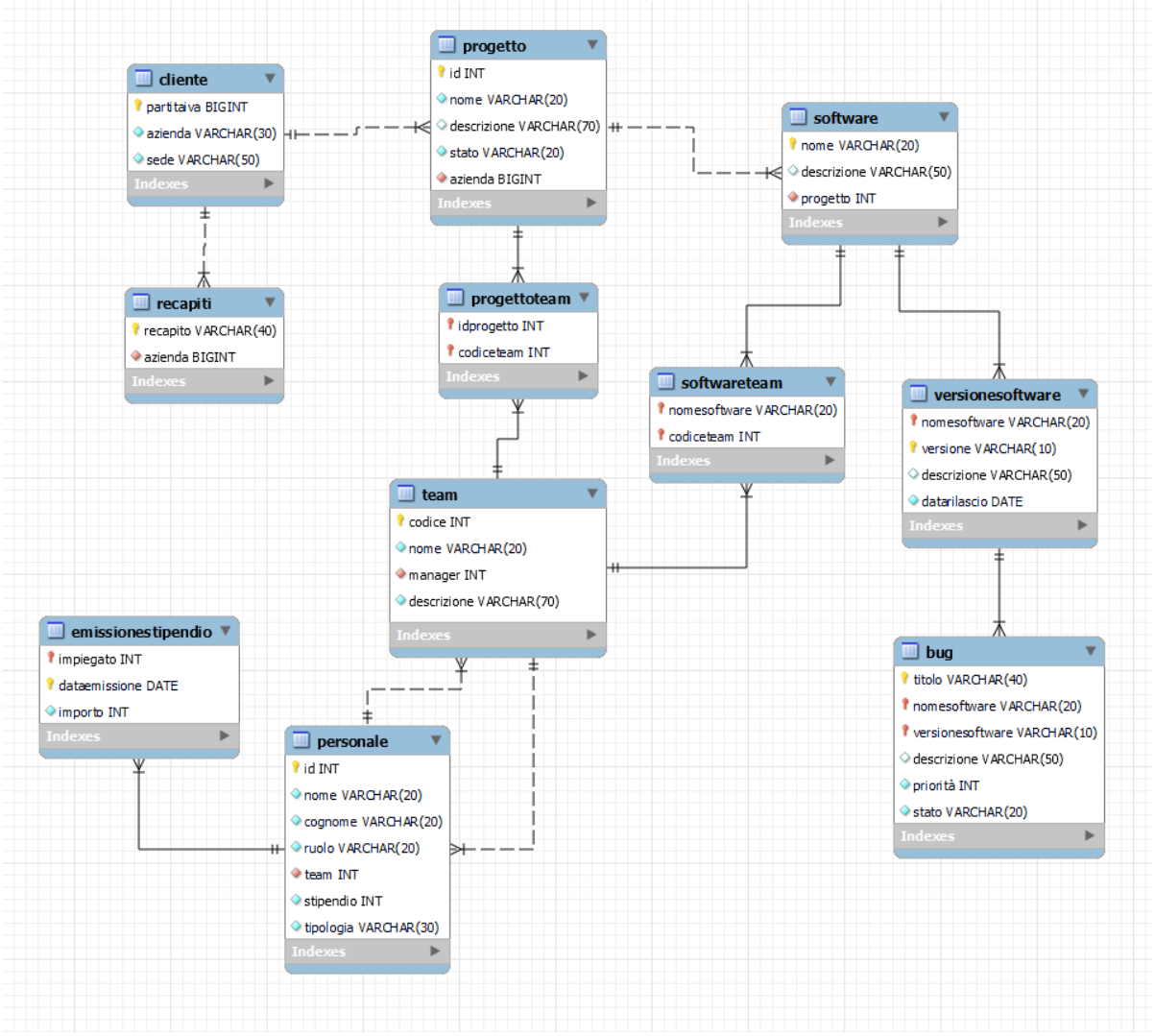
O12: Visualizzare i clienti che hanno commissionato più progetti

```
SELECT c.azienda, COUNT(*) as NumeroProgetti
FROM Cliente c JOIN Progetto p
ON c.partitaiva = p.azienda
GROUP BY c.azienda
HAVING NumeroProgetti >= ALL ( SELECT COUNT(*)
                               FROM Cliente c2 JOIN Progetto p2
                               ON c2.partitaiva = p2.azienda
                               GROUP BY c2.azienda );
```

O13: Visualizzare i team che lavorano a più progetti

```
SELECT t.codice, t.nome, COUNT(*) as NumeroProgetti
FROM Team t
JOIN ProgettoTeam pt ON t.codice = pt.codiceteam
JOIN Progetto p ON pt.idprogetto = p.id
GROUP BY t.codice
HAVING NumeroProgetti >= ALL (  SELECT COUNT(*)
                                FROM Team t2
                                JOIN ProgettoTeam pt2
                                ON t2.codice = pt2.codiceteam
                                JOIN Progetto p2
                                ON pt2.idprogetto = p2.id
                                GROUP BY t2.nome );
```

EER Diagram


cliente
partitaiva BIGINT
azienda VARCHAR(30)
sede VARCHAR(50)
Indexes
progetto
id INT
nome VARCHAR(20)
descrizione VARCHAR(70)
stato VARCHAR(20)
azienda BIGINT
Indexes
software
nome VARCHAR(20)
descrizione VARCHAR(50)
progetto INT
Indexes
recapiti
recapito VARCHAR(40)
azienda BIGINT
Indexes
progettoteam
idprogetto INT
codiceteam INT
Indexes
softwareteam
nomesoftware VARCHAR(20)
codiceteam INT
Indexes
versionesoftware
nomesoftware VARCHAR(20)
versione VARCHAR(10)
descrizione VARCHAR(50)
datarilascio DATE
Indexes
team
codice INT
nome VARCHAR(20)
manager INT
descrizione VARCHAR(70)
Indexes
emissionestipendio
impiegato INT
dataemissione DATE
importo INT
Indexes
personale
id INT
nome VARCHAR(20)
cognome VARCHAR(20)
ruolo VARCHAR(20)
team INT
stipendio INT
tipologia VARCHAR(30)
Indexes
bug
titolo VARCHAR(40)
nomesoftware VARCHAR(20)
versionesoftware VARCHAR(10)
descrizione VARCHAR(50)
priorità INT
stato VARCHAR(20)
Indexes