Basi di dati relazionali ad oggetti

Sommario

- Introduzione agli ORDBMS
- Sistema di tipi:
  - tipi semplici
  - tipi complessi
  - tipi riferimento
  - tipi collezione
- Ereditarietà
  - di tipi
  - di tabelle
- Large OBject (LOB)
- Metodi, Trigger & Stored Procedure

Utilizzo di un DBMS

- Tre classi di applicazioni:
  - applicazioni gestionali
  - applicazioni navigazionali
  - applicazioni multimediali

Applicazioni gestionali

- Applicazioni OLTP
  - grosse quantità di dati
  - dati ed interrogazioni semplici
  - molti utenti concorrenti (transazioni)
  - grande numero di interrogazioni e modifiche

Applicazioni gestionali

- Applicazioni OLAP
  - grosse quantità di dati
  - dati semplici ma interrogazioni complesse
  - molti utenti
  - accesso quasi esclusivamente in consultazione

Applicazioni gestionali

- Le applicazioni gestionali richiedono:
  - elevate prestazioni
  - scalabilità
  - protezione dei dati
  - supporto all'amministrazione del sistema
  - tool

Applicazioni navigazionali

- Tipici esempi sono le applicazioni CAD e per le telecomunicazioni
- Devono gestire associazioni complesse tra i dati e navigare in esse

Applicazioni multimediali

- Richiedono di memorizzare e gestire immagini, testi, suoni, video, ecc. oltre a dati tradizionali
- Necessità di definire operazioni specifiche per i dati multimediali

Applicazioni: linee di tendenza

- La costante diminuzione del costo dell'hw ha facilitato la memorizzazione di dati complessi, multimediali che richiedono nuovi strumenti per la loro gestione e interrogazione

RDBMS: panorama attuale

- Gestiscono e manipolano dati semplici (tabellari)
- Hanno un linguaggio di interrogazione (SQL) semplice, dichiarativo e standard
- Tool consolidati per lo sviluppo di applicazioni (Oracle Developer 2000, Sybase Power Builder, Microsoft Visual Basic)

RDBMS: panorama attuale

- Buone prestazioni
- Affidabilità
- Basati su una architettura client-server supportano efficientemente un gran numero di utenti
- Forniscono meccanismi di controllo dell'accesso

RDBMS: panorama attuale

- Forniscono strumenti per la distribuzione e replicazione dei dati
- Portabili su diverse piattaforme
- Esempi di RDBMS: IBM DB2, Oracle, Sybase, Informix, Microsoft SQL Server

OODBMS: panorama attuale

- Permettono di modellare direttamente oggetti complessi e le loro associazioni
- Buone prestazioni per applicazioni navigazionali
- Limitato supporto per concorrenza, parallelismo e distribuzione

OODBMS: panorama attuale

- Semplici modelli transazionali
- Limitate funzionalità di controllo dell'accesso
- Coprono un mercato "di nicchia" che richiede accessi navigazionali efficienti (disegno di chip, ecc.)
- Esempi di OODBMS: ObjectStore, Gemstone, ONTOS, O2, Objectivity

RDBMS vs. OODBMS

- RDBMS forniscono un buon supporto per applicazioni che manipolano dati semplici
- OODBMS forniscono un supporto efficiente per alcune classi di applicazioni su dati complessi, ma senza molti degli aspetti positivi dei RDBMS

RDBMS vs. OODBMS

Non esiste un modello dei dati migliore di un altro ma esistono applicazioni per cui un modello è migliore di un altro

DBMS relazionali ad oggetti

- I DBMS relazionali ad oggetti (object-relational) nascono dall'esigenza di assicurare le funzionalità dei RDBMS rispetto alla gestione di dati tradizionali, estendendo il modello dei dati con la possibilità di gestire dati complessi tipica degli OODBMs

Estensione del sistema dei tipi

- Tipi semplici
- Tipi complessi:
  - ADT
  - row types
- Tipi riferimento
- Tipi collezione

Estensione dei tipi primitivi

- In SQL92 i tipi di un attributo in una relazione possono essere:
  - **numerici** (interi, reali, ecc.)
  - **carattere** (stringhe di lunghezza fissa o variabile, caratteri singoli)
  - **temporali** (date, time, datetime, interval)

Estensione dei tipi primitivi

- Per ogni tipo primitivo esistono un insieme fisso e pre-definito di operazioni che su di esso possono essere eseguite
- Queste limitazioni rendono spesso difficile la rappresentazione di dati reali
- In SQL:1999 i tipi vengono estesi con un insieme di UDT (user defined types) creati con il comando CREATE TYPE

Tipi semplici

- I tipi semplici (o distinct type) sono la forma più semplice di estensione del sistema dei tipi fornita da un ORDBMS
- Consentono agli utenti di creare nuovi tipi di dati, basati su un solo tipo primitivo
- Sono usati per definire tipi di dati che richiedono operazioni diverse rispetto al tipo primitivo su cui sono definiti

Tipi semplici

- I tipi semplici sono considerati dal DBMS *totalmente distinti* dal tipo primitivo su cui si basano
- I valori del tipo semplice non sono direttamente confrontabili con quelli del corrispondente tipo primitivo (strong typing)

Tipi semplici

- Confronti con il tipo primitivo o con altri tipi semplici definiti sullo stesso tipo primitivo richiedono operazioni di **cast**
- Spesso l'ORDBMS crea automaticamente una funzione di cast quando un nuovo tipo semplice viene creato
- Non è fornito alcun meccanismo di ereditarietà e subtyping per i tipi semplici

Esempio

- Si supponga di creare un nuovo tipo id_impiegato basato sul tipo intero
- Come il tipo intero, id_impiegato è utilizzato per memorizzare valori numerici ma il DBMS tratterà i due tipi come tipi distinti
- Per i due tipi possono essere definite operazioni diverse (ad esempio la somma di due identificatori non ha senso)

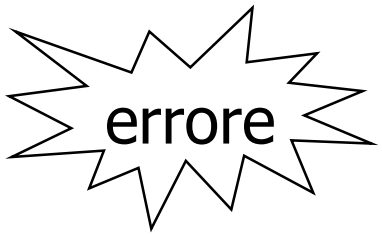
Esempio

```
CREATE TYPE id_impiegato AS INTEGER;
CREATE TABLE Impiegati(
   id             id_impiegato,
   nome           VARCHAR(50),
   id_manager     id_impiegato);
```

[in realtà in SQL:1999 c'è clausola FINAL obbligatoria per i distinct type]

Esempio

```
SELECT nome
FROM Impiegati
WHRER id_manager = 123;
```




```
SELECT nome
FROM Impiegati
WHERE id_manager = id_impiegato(123);
```

Esempio

```
CREATE TYPE Dollaro_Canadese AS Decimal(8,2);

CREATE TYPE Dollaro_USA AS Decimal(8,2)
    METHOD "+"(Dollaro_USA) RETURNS Dollaro_USA;

CREATE TABLE Vendite_Canadesi(
    n_cliente           INTEGER,
    n_ordine            INTEGER,
    totale              Dollaro_Canadese);

CREATE TABLE Vendite_USA(
    n_cliente           INTEGER,
    n_ordine            INTEGER,
    totale              Dollaro_USA);
```

Esempio

```
SELECT n_cliente,n_ordine
 FROM Vendite_Canadesi CDN, Vendite_USA USA
 WHERE CDN.n_ordine = USA.n_ordine
    AND CDN.totale > USA.totale
```

```
SELECT n_cliente,n_ordine
 FROM Vendite_Canadesi CDN, Vendite_USA USA
 WHERE CDN.n_ordine = USA.n_ordine
    AND CDN.totale > Dollaro_Canadese(USA.totale)
```

Tipi complessi

- Un tipo complesso, o strutturato,include uno o più attributi
- Analoghi alle struct del C o ai record del Pascal
- Possono essere usati come:
  - **tipi di una colonna** in una relazione (ADT)
  - **tipi di una tabella** (row types)

Abstract data types

- Si supponga di voler rappresentare l'indirizzo di un impiegato in un RDBMS

  sono possibili due alternative:

  - indirizzo VARCHAR(n)
  - rappresentare ogni componente dell'indirizzo come un attributo separato

Abstract data types

```
CREATE TYPE t_indirizzo AS
    numero_civico     INTEGER,
    via               VARCHAR(50),
    città             CHAR(20),
    stato             CHAR(2),
    cap               INTEGER;
```

t_indirizzo è un tipo complesso i cui attributi hanno tipi predefiniti

[in realtà in SQL:1999 è obbligatorio specificare FINAL o NOT FINAL]

Abstract data types

- Gli ADT possono essere usati come tipi di una colonna di una relazione:

```
CREATE TABLE Impiegati (
        imp#        id_impiegato,
        nome        CHAR(20),
        curriculum TEXT,
        indirizzo  t_indirizzo);
```

Abstract data types

Tabella Impiegati

imp#	nome	curriculum	indirizzo numero_civico via città stato cap

Abstract data types

- Gli ADT possono anche essere annidati:

```
CREATE TYPE t_impiegato AS
        id              id_impiegato,
        nome            CHAR(20),
        curriculum      TEXT,
        indirizzo       t_indirizzo;
```

Abstract data types

- Sugli ADT possono essere definiti metodi come parte della definizione del tipo

```
CREATE TYPE t_impiegato AS
                id              id_impiegato,
                nome            CHAR(20),
                curriculum      TEXT,
                indirizzo       t_indirizzo
METHOD cognome() RETURNS CHAR(10),
 METHOD confronta(t_impiegato) RETURNS BOOLEAN;
```

Abstract data types

- Gli ADT sono totalmente incapsulati
- La loro manipolazione può avvenire solo mediante apposite funzioni automaticamente create dal DBMS al momento della creazione dell'ADT

Abstract data types

- Tre tipi di funzioni
  - funzioni **costruttore**: per creare una nuova instanza di un ADT
  - funzioni **observer**: per formulare interrogazioni sugli ADT
  - funzioni **mutator**: per cambiare valori ad istanze di un ADT

Funzioni costruttore

- Ad ogni ADT è automaticamente associato un metodo (costruttore) con lo stesso nome del tipo
- Il costruttore crea un'istanza del tipo

Esempio

t_indirizzo() ----> t_indirizzo

crea una nuova istanza del tipo t_indirizzo con gli attributi inizializzati ai valori di default (tali valori possono anche essere NULL)

t_indirizzo(39, ‘Comelico', 'Milano', 'IT', 20135)

crea una nuova istanza del tipo t_indirizzo con gli attributi inizializzati in base ai valori forniti

Esempio

```
INSERT INTO Impiegati
VALUES(SM123,'Smith',NULL,
        t_indirizzo(14,'Sauli', 'Milano','IT', 20135));
```

imp#	nome	curriculum	indirizzo numero_civico via città stato cap
SM123	Smith	NULL	14 Sauli Milano IT 20135

Funzioni observer

- Per ogni componente di un ADT è automaticamente creata una funzione observer con lo stesso nome della componente:

  numero_civico() ----> INTEGER

  via() ----> VARCHAR(50)

  città() ----> CHAR(20)

  stato() ----> CHAR(2)

  cap() ----> INTEGER

Esempio

```
SELECT nome
FROM Impiegati
WHERE indirizzo.città() = 'Milano'
        OR indirizzo.città() = 'Roma';
```

Funzioni mutator

- Servono per modificare istanze di un ADT

numero_civico(INTEGER)
via(VARCHAR(50))
città(CHAR(20))
stato(CHAR(2))
cap(INTEGER)

Esempio

```
BEGIN
   DECLARE i t_indirizzo     /* dichiarazione variabile temporanea */
   SET u = t_indirizzo();                    /* costruttore */
   SET u = u.cap(20135);                     /* mutator */
   SET u = u.via('Sauli');                   /* mutator */
   …
   INSERT INTO IMPIEGATI VALUES ('Rossi', '…', u);
END;

UPDATE Impiegati
  SET indirizzo = indirizzo.cap(20135)
WHERE nome = 'Smith';
```

Row types

- Un dato complesso può anche essere usato come tipo di una intera tabella (row type)
- Le righe della tabella sono istanze del tipo complesso mentre le colonne coincidono con gli attributi del tipo complesso
- Permettono di:
  - definire un insieme di tabelle che condividono la stessa struttura
  - modellare in modo intuitivo le associazioni tra dati in tabelle diverse
  - definire gerarchie di tabelle

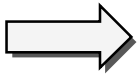
Esempio

- Si supponga di voler memorizzare informazioni sugli impiegati ed i progetti a cui lavorano
- In un RDBMS avrei due tabelle Impiegati e Progetti
- Nella tabella Impiegati è presente una colonna che indica il progetto a cui l'impiegato lavora (chiave esterna)

Esempio

Impiegati

imp#		prj#
SM123	...	12



Progetti

prj#	nome	
12	Oracle	....

Row types

- In un ORDBMS ho due opzioni in più:
  - **definire un ADT** t_progetto e usare questo come tipo di una colonna della relazione Impiegati (ridondanza dei dati perché lo stesso progetto può essere memorizzato molte volte in Impiegati)
  - **definire una tabella** basata su un nuovo tipo complesso e riferire le colonne istanza di questo nuovo tipo

Esempio

```
CREATE TYPE t_progetto AS
    prj#            INTEGER,
    nome            VARCHAR(20),
    descrizione     VARCHAR(50),
    budget          INTEGER;

CREATE TABLE Progetti OF t_progetto;
```

Esempio

Progetti

prj#	nome	descrizione	budget
12	Oracle	ORDBMS	10,000,000

Row types

- Nessun meccanismo di incapsulazione
- L'incapsulazione c'e' solo quando un ADT è usato come tipo di una colonna
- Gli attributi del row type sono visti come colonne della tabella
- Le interrogazioni sono eseguite nel modo standard

Esempio

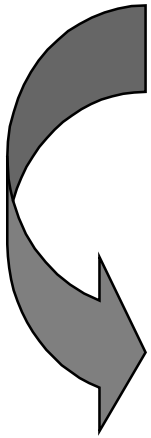
```
SELECT *
  FROM Progetti
  WHERE budget > 1,000,000;
```

Tipi riferimento

- I row type possono essere combinati con i tipi riferimento (REF type)
- Permettono di rappresentare facilmente le associazioni tra istanze di tipi
- Tali tipi permettono ad una colonna di riferire una tupla in un'altra relazione
- Una tupla in una relazione viene identificata tramite il suo OID

Esempio

```
CREATE TABLE Progetti OF t_progetto
 (prog_ref              REF(t_progetto));
```



prog_ref	prj#	nome	descrizione	budget
xxxx	433	DB2	ORDBMS	20,000,000

Riferimento univoco creato automaticamente dal DBMS

Esempio

- A questo punto è possibile creare una tabella Impiegati contenente dei riferimenti alla tabella Progetti
- Questi riferimenti servono per modellare l'associazione tra un impiegato e i progetti a cui lavora

Esempio

```
CREATE TABLE Impiegati(
   imp#            id_impiegato,
   nome            VARCHAR(50),
   indirizzo       t_indirizzo,
   assegnamento    REF(t_progetto) SCOPE Progetti);
```

Più impiegati possono riferire lo stesso progetto

Un impiegato è assegnato al massimo ad un progetto

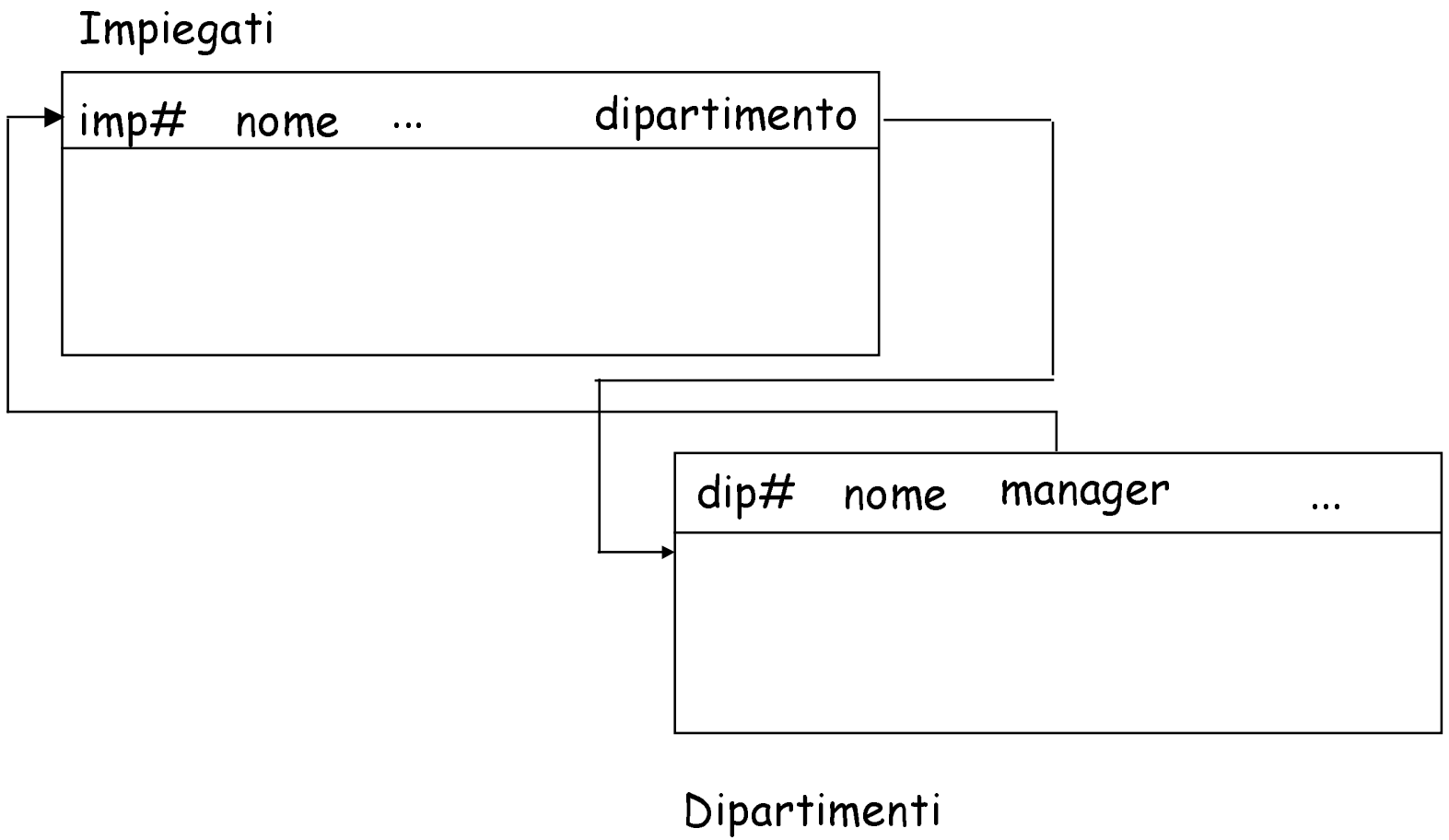
Esempio

```
CREATE TYPE t_impiegato(
   imp#              id_impiegato,
   nome              CHAR(20),
   curriculum        TEXT,
   indirizzo         t_indirizzo,
   dipartimento      REF(t_dipartimento));

CREATE TABLE Impiegati OF t_impiegato;
```

Esempio

```
CREATE TYPE t_dipartimento(
    dip#        INTEGER,
    nome        CHAR(10),
    manager     REF(t_impiegato),
    piantina    PICTURE);

CREATE TABLE Dipartimenti of t_dipartimento;
```

Esempio


Impiegati
imp# nome ... dipartimento
dip# nome manager ...
Dipartimenti

Esempio

- La colonna dipartimento di Impiegati *punta* ad una tupla della tabella Dipartimenti (quella corrispondente al dipartimento in cui l'impiegato lavora)
- La colonna manager di Dipartimenti *punta* ad una tupla della tabella Impiegati (quella corrispondente al direttore del dipartimento)

Tipi riferimento

- In SQL:1999 sono definite due nuove funzioni per la manipolazione dei tipi riferimento:
  - DEREF che riceve in input un attributo di tipo riferimento e restituisce la riga puntata:

    DEREF(Impiegati.dipartimento)

    restituisce una riga della tabella Dipartimenti
  - -> che riceve in input un attributo di tipo riferimento e permette di accedere ad un attributo della riga puntata:

    Impiegati.dipartimento -> nome

Esempio

SELECT manager

FROM Dipartimenti

WHERE nome = “Dischi”;

Restituisce un puntatore ad un impiegato (cioè l’oid dell’impiegato che è manager del dipartimento Dischi)

SELECT DEREF(manager)

FROM Dipartimenti

WHERE nome = “Dischi”;

Restituisce informazioni sul manager del dipartimento Dischi (un’intera riga della tabella Impiegati)

Esempio

```
SELECT DEREF(manager).nome
   FROM Dipartimenti
   WHERE nome = “Dischi”;
```

è equivalente a:

```
SELECT manager -> nome
   FROM Dipartimenti
   WHERE nome = “Dischi”;
```

Restituisce il nome del manager del dipartimento Dischi

Tipi collezione

- I tipi collezione definiscono dei *contenitori* per oggetti con struttura simile
- Non esiste ancora una standardizzazione sull'insieme di tipi collezione supportati dai vari ORDBMS
- In particolare, il solo tipo collezione incluso in SQL:1999 è ARRAY

Tipi collezione

- Set, Bag, List
- Array (a differenza delle liste la dimensione è prefissata)

sintassi: tipo ARRAY[dimensione]

Esempio

```
CREATE TABLE Impiegati(
    imp#              id_impiegato,
    nome              VARCHAR(50),
    competenze        VARCHAR(20) ARRAY[5]),
    titolo_di_studio  VARCHAR(30));
```

Esempio

```
CREATE TYPE t_impiegato(
    imp#         id_impiegato,
    nome         VARCHAR(30),
   indirizzo     t_indirizzo,
    manager      REF(t_impiegato),
    progetti     REF(t_progetto) ARRAY[10]),
    figli        REF(t_persona) ARRAY[10]),
    hobby        VARCHAR(20) ARRAY[5]);

CREATE TABLE Impiegati2 OF t_impiegato;
```

Ereditarietà

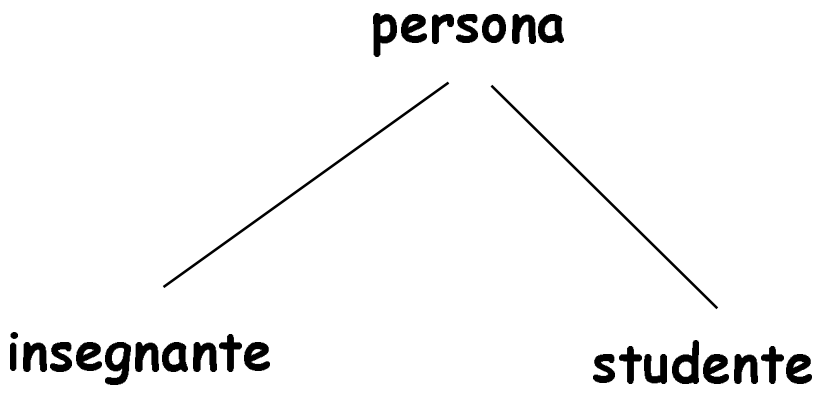
- Possibilità di definire relazioni di supertipo/sottotipo
- L'ereditarietà consente di specializzare i tipi esistenti a seconda delle esigenze dell'applicazione
- Un sottotipo eredita gli attributi e i metodi dei suoi supertipi
- Si possono distinguere due tipi di ereditarietà
  - ereditarietà di tipi
  - ereditarietà di tabelle

Ereditarietà di tipi

```
CREATE TYPE t_persona AS
   nome              CHAR(20),
   id                INTEGER,
   data_di_nascita   DATE,
   indirizzo         t_indirizzo;

CREATE TYPE t_insegnante AS
   stipendio         DECIMAL(8,2),
   dipartimento      REF(t_dipartimento),
   corso_insegnato   REF(t_corso)
UNDER t_persona;
```

Ereditarietà di tipi

```
CREATE TYPE t_studente AS
   corsi_seguiti REF(t_corso) ARRAY[30]
UNDER t_persona;
```



Esempio

- Supponiamo che l'editore di un quotidiano voglia mantenere una tabella Abbonati con informazioni sui suoi abbonati
- La tabella contiene due colonne: nome ed indirizzo dell'abbonato
- Gli abbonati possono ricevere il quotidiano sia a casa sia sul posto di lavoro

Esempio

```
CREATE TYPE t_indirizzo AS
   numero_civico     INTEGER,
   via               VARCHAR(50),
   città             CHAR(20),
   stato             CHAR(2),
   cap               INTEGER;

CREATE TYPE t_indirizzo_lavoro AS
   posta_interna     VARCHAR(20)
UNDER t_indirizzo;
```

Ereditarietà di tabelle

- Come visto in precedenza i row type possono essere usati come tipi di tabelle
- I row type possono essere organizzati in gerarchie di ereditarietà
- La gerarchia definita sui row type impone anche una gerarchia sulle tabelle definite con questi tipi

Esempio

- CREATE TABLE Persone OF t_persona;
- CREATE TABLE Insegnanti of t_insegnante
  UNDER Persone;
- CREATE TABLE Studenti OF t_studente
  UNDER Persone;
- E' stata creata una gerarchia tra le tabelle Persone, Insegnanti e Studenti
- La gerarchia influenza i risultati delle interrogazioni

Interrogazioni

- Una interrogazione fatta su una tabella si propaga automaticamente alle sottotabelle
- Lo stesso vale per le operazioni di cancellazione mentre una operazione di inserimento coinvolge solo una specifica tabella

Large Objects (LOBs)

- Facilitano la memorizzazione di dati multimediali (documenti, immagini, audio, ecc.)
- Possono contenere fino a 4GB di dati (di solito i RDBMS non vanno oltre 2-32KB)
- Il DBMS non associa nessuna interpretazione a questi datiSi distinguono in:
  - BLOB (Binary Large Object)
  - CLOB (Character Large Object)
- Sono fisicamente memorizzati esternamente alle tabelle ma internamente al DB (comportamento transazionale)

LOB

```
CREATE TABLE Pazienti(
    id                  INTEGER,
    età                 INTEGER,
    cartella_clinica    CLOB(1M),
    radiografia         BLOB(10M));
```

Metodi

- I metodi sono funzioni definite dall'utente associate ai tipi
- Possono essere scritti in diversi linguaggi
- La sintassi varia notevolmente a seconda del DBMS utilizzato
- I metodi sono ereditati dai sottotipi allo stesso modo degli attributi
- Overloading, overriding e late binding dei metodi

Metodi

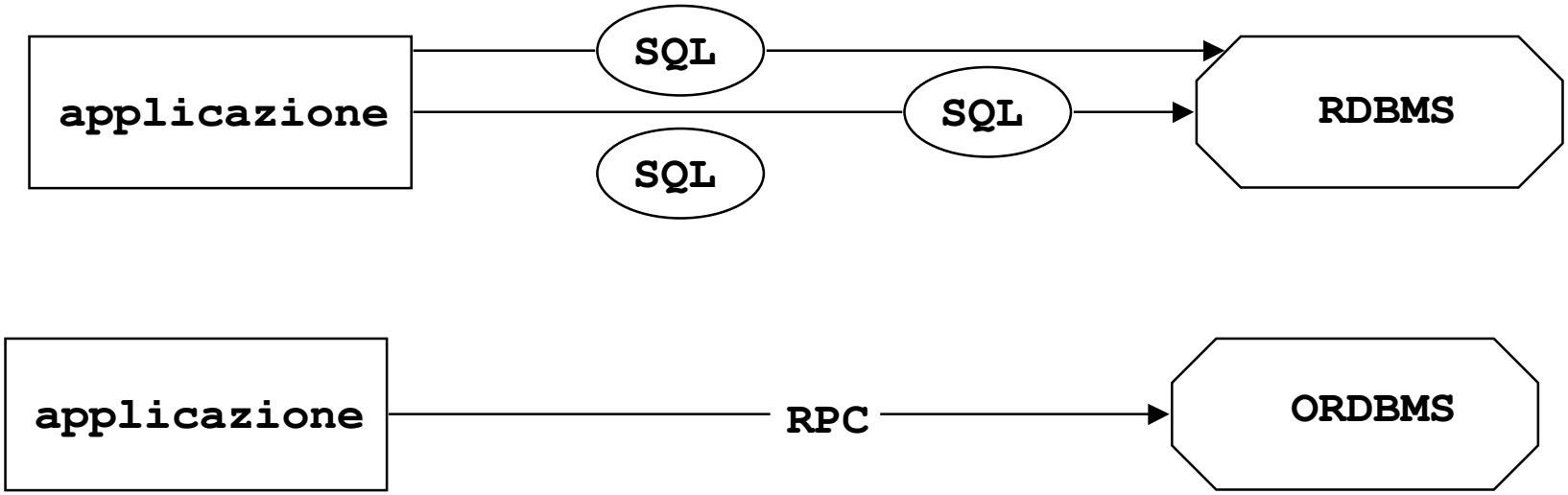
```
CREATE TYPE t_persona AS
    nome                CHAR(20),
    id                  INTEGER,
    data_di_nascita     DATE,
    indirizzo           t_indirizzo
 METHOD età() RETURNS INTEGER;
```

Trigger

- I trigger sono delle regole attive (scritte in SQL) che possono essere associate alle tabelle
- I trigger vengono automaticamente attivati al verificarsi di una certa condizione
- La loro attivazione comporta una serie di operazioni nel DB (inserimenti in tabelle, ecc.)
- Struttura generale:

  ON <evento> **WHERE** <condizione> **DO** <azione>
- Consentono di specificare facilmente vincoli e controlli

Stored Procedure

- Sono procedure memorizzate persistentemente nello schema del DB
- Possono essere invocate da molteplici applicazioni
- Migliorano le prestazioni in quanto riducono la comunicazione tra client (applicazione) e DBMS (server)
- Possono essere scritte in un linguaggio di programmazione (Java,C++,ecc.) o in un'estensione procedurale del linguaggio SQL (Oracle PL/SQL,Sybase Transact/SQL, Informix 4GL,)
- Il maggiore svantaggio delle stored procedure è la mancanza di standardizzazione

Stored Procedure