Basi di Dati:
Corso di laboratorio

Lezione 1

Raffaella Gentilini

Sommario

Il linguaggio SQL

SQL

- Acronimo per Structured Query Language
- Il linguaggio SQL ha varie funzionalita':

Il linguaggio SQL

SQL

- Acronimo per Structured Query Language
- Il linguaggio SQL ha varie funzionalita':

1) Data Definition Language (DDL):
   - Permette di definire, modificare e cancellare strutture dati e vincoli di integrita'.
   - Istruzioni del DDL:
     `CREATE, ALTER, DROP, RENAME, TRUNCATE` ...

Il linguaggio SQL

SQL

- Acronimo per Structured Query Language
- Il linguaggio SQL ha varie funzionalita':

2) Data Manipulation Language (DML)
   - Consente la manipolazione dei dati. In particolare, il DML consente di estrarre, modificare e cancellare i dati contenuti nella base di dati.
   - Istruzioni del DML:
     `SELECT, INSERT, UPDATE, DELETE` ...

Il linguaggio SQL

SQL

- Acronimo per Structured Query Language
- Il linguaggio SQL ha varie funzionalita':

3) Data Control Language (DCL)
   - Permette di gestire i privilegi degli utenti della base di dati.
   - Istruzioni del DCL:
     `GRANT, REVOKE`

Il linguaggio SQL

SQL

- Acronimo per Structured Query Language
- Il linguaggio SQL ha varie funzionalita':

4) Transaction Control Language (TCL)
   - Permette di gestire le transazioni ed in particolare la validazione e l'annulamento delle modifiche della base di dati definite mediante il DDL ed il DML.
   - Istruzioni del TCL:
     `COMMIT, SAVEPOINT, ROLLBACK, SET TRANSACTION`

Il linguaggio SQL

SQL

- Acronimo per Structured Query Language
- Il linguaggio SQL ha varie funzionalita':

5) Embedded SQL
   - Permette di integrare i comandi SQL in un linguaggio ospite (`C`, `C++`, Java ...)

Il linguaggio SQL nel Tempo

SQL:Un po' di storia

1974 Prima proposta SEQUEL (IBM Research)

1981 Prime implementazioni in SQL/DS (IBM) e Oracle.

1983 Dal 1983 e' uno standard di fatto:

- standard 1986,
- standard 1992,
- poi 1999,
- e infine 2003.
- recepito solo in parte

SQL-92

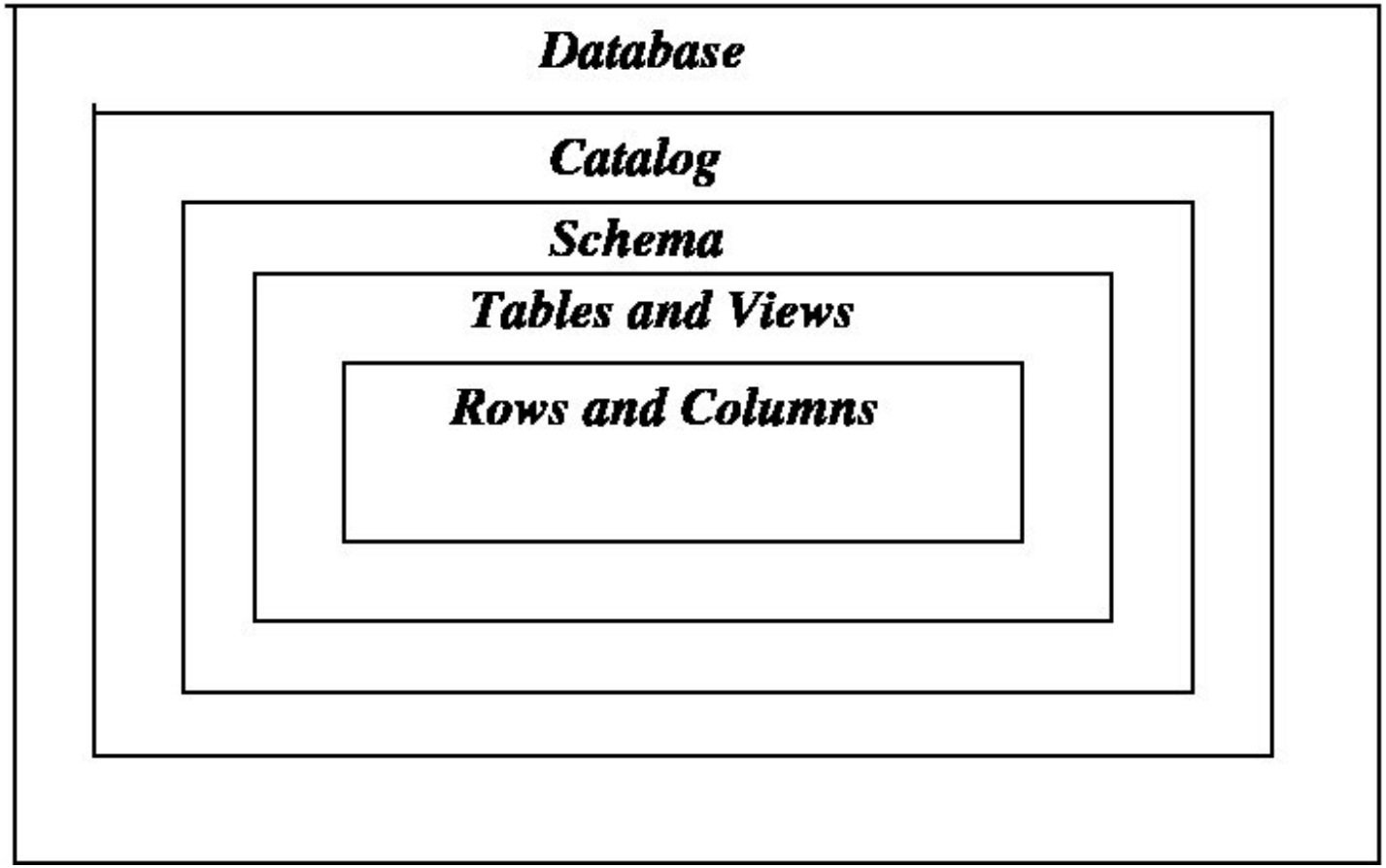
SQL-92

- E' un linguaggio ricco e complesso
- ancora nessun sistema mette a disposizione tutte le funzionalita' del linguaggio
- 3 livelli di aderenza allo standard:
  - Entry SQL: Abbastanza simile a SQL-89
  - Intermediate SQL Supportato dai DBMS tradizionali/ caratteristiche piu' importanti per le esigenze del mercato
  - Full SQL: Funzioni avanzate, in via di inclusione nei sistemi
- i sistemi offrono funzionalita' non standard
  - incompatibilia' tra sistemi/ con i nuovi standard (SQL-99)
- Nuovi standard conservano caratteristiche di base di SQL-92
  - SQL-99: Aggiunge alcune funzionalita' orientate agli oggetti
  - SQL-03: Aggiunge il supporto per dati XML

Utilizzo di un DBMS basato su SQL

- Un DBMS basato su SQL consente di gestire basi di dati relazionali. Dal punto di vista sistemistico e' un server.
- Quando ci si connette ad un DBMS basato su SQL si deve indicare, implicitamente o esplicitamente, su quale base di dati si vuole operare
- Se si vuole utilizzare una base di dati non ancora esistente, si utilizza un meccanismo messo a disposizione dal server per la sua creazione
- Coerentemente con la filosofia del modello relazionale, una base di dati in SQL e' caratterizzata dallo schema (livello intensionale) e da una istanza (quella corrente– livello estensionale)
- Inoltre, una base di dati SQL e' caratterizzata da un insieme di meta-dati (il catalogo)

Astrazione dei dati in SQL



Definizione di uno schema

Concetto di Schema

- Uno schema permette di dividere logicamente una base di dati in parti separate che possono comunicare tra loro.
- E' identificato da un nome di schema

Definizione di uno schema

Concetto di Schema

- Uno schema permette di dividere logicamente una base di dati in parti separate che possono comunicare tra loro.
- E' identificato da un nome di schema

Example

La seguente istruzione del DDL di SQL crea uno schema chiamato *Ditta*:

```
CREATE SCHEMA  Ditta;
```

Definizione di uno schema

Sintassi dell'istruzione CREATE SCHEMA

`CREATE SCHEMA` <*nome_database*> [`AUTHORIZATION`<*nome_prop*>]

A seguito del nome, uno schema puo' contenere:

Definizione di uno schema

Sintassi dell'istruzione CREATE SCHEMA

`CREATE SCHEMA <`*nome_database*`> [AUTHORIZATION<`*nome_prop*`>]`

A seguito del nome, uno schema puo' contenere:

- un identificatore di autorizzazione per indicarne il proprietario proprietario (se vuoto: l'utente)

Definizione di uno schema

Sintassi dell'istruzione CREATE SCHEMA

`CREATE SCHEMA <nome_database> [AUTHORIZATION<nome_prop>]`

A seguito del nome, uno schema puo' contenere:

- un identificatore di autorizzazione per indicarne il proprietario proprietario (se vuoto: l'utente)
- definizioni di tabelle, domini, viste, privilegi, vincoli e funzioni.

Metalinguaggio di Sintassi

- [ ] Le parentesi quadre indicano che il contenuto e' opzionale
- < > Indicano che il contenuto e' di libero arbitrio (e.g. scelta del nome dello schema)
- { | } Indicano una scelta (e.g. $\{item1 \mid item2\}$ indica la scelta tra *item*1 ed *item*2)
- ... I punti di sospensione indicano la possiblita' di una ripetizione

Oggetti di uno schema

Dopo aver creato uno schema e' possibile aggiungere oggetti al suo interno qualificandone il nome mediante il nome dello schema seguito da un punto

Example

```
CREATE DOMAIN  Ditta.dom_stipendio AS NUMERIC(8,2)
    CHECK VALUE  >= 900;
CREATE DOMAIN  Ditta.dom_cod_impiegato AS VARCHAR(4);
CREATE TABLE  Ditta.Impiegato (
        cod Ditta.dom_cod_impiegato PRIMARY KEY,
        nome varchar(40) NOT NULL,
        stipendio Ditta.dom_stipendio
      );
```

Oggetti di uno schema

- Di solito lo schema SQL in cui vengono dichirati gli oggetti (tabelle, domini . . .) e' specificato implicitamente dall'ambiente in cui vengono eseguite le rispettive istruzioni di `CREATE`
- In tal caso non e' necessario associare esplicitamente il nome dello schema ai nomi degli oggetti separandoli con un punto

Example

```
CREATE TABLE  Impiegato (
        cod dom_cod_impiegato PRIMARY KEY,
        nome varchar(40) NOT NULL,
        stipendio dom_stipendio
      );
```

Schemi predefiniti in PostgreSQL

In PostgreSQL esiste uno schema predefinito public.
Quando un elemento della base di dati viene creato o riferito senza la qualificazione di uno schema, il sistema assume che la qualificazione sia public.

Example

Ad esempio, l'istruzione:

```
CREATE TABLE R (a CHAR PRIMARY KEY, b CHAR);
```

equivale a:

```
CREATE TABLE public.R (a CHAR PRIMARY KEY, b CHAR);
```

Definizione di Tabelle

CREATE TABLE

L'istruzione del DDL di SQL CREATE TABLE:

- definisce uno schema di relazione (tabella)
- crea un'istanza vuota dello schema
- specifica attributi, domini e vincoli.

CREATE TABLE: Sintassi

Example

```
CREATE TABLE utente(
   email VARCHAR(40) NOT NULL,
   nome VARCHAR(30) NOT NULL,
   cognome VARCHAR(30) NOT NULL,
   anno_nascita INTEGER,
   PRIMARY KEY (email));
```

CREATE TABLE: Sintassi

Example

```
CREATE TABLE utente(
   email VARCHAR(40) NOT NULL,
   nome VARCHAR(30) NOT NULL,
   cognome VARCHAR(30) NOT NULL,
   anno_nascita INTEGER,
   PRIMARY KEY (email));
```

Sintassi dell'istruzione CREATE TABLE

CREATE TABLE <*nome_tab*> (
<*nome_col*><*dominio*> [DEFAULT<*val*>] [<*vincolo_col*> [...]]
[, {<*nome_col*><*dominio*> [DEFAULT<*val*>] [<*vincolo_col*> [...]] |
<*vincolo_tab*>}[, ...]])

CREATE TABLE: Sintassi

Example

```
CREATE TABLE utente(
   email VARCHAR(40) NOT NULL,
   nome VARCHAR(30) NOT NULL,
   cognome VARCHAR(30) NOT NULL,
   anno_nascita INTEGER,
   PRIMARY KEY (email));
```

Sintassi dell'istruzione CREATE TABLE

CREATE TABLE <*nome_tab*> (
<*nome_col*><*dominio*> [DEFAULT<*val*>] [<*vincolo_col*> [...]]
[, {<*nome_col*><*dominio*> [DEFAULT<*val*>] [<*vincolo_col*> [...]] |
<*vincolo_tab*>}[, ...]])

CREATE TABLE: Sintassi

Example

```
CREATE TABLE utente(
   email VARCHAR(40) NOT NULL,
   nome VARCHAR(30) NOT NULL,
   cognome VARCHAR(30) NOT NULL,
   anno_nascita INTEGER,
   PRIMARY KEY (email));
```

Sintassi dell'istruzione CREATE TABLE

CREATE TABLE <*nome_tab*> (
<*nome_col*><*dominio*> [DEFAULT<*val*>] [<*vincolo_col*> [...]]
[, {<*nome_col*><*dominio*> [DEFAULT<*val*>] [<*vincolo_col*> [...]] |
<*vincolo_tab*>}[, ...]])

CREATE TABLE: Sintassi

Example

```
CREATE TABLE utente(
   email VARCHAR(40) NOT NULL,
   nome VARCHAR(30) NOT NULL,
   cognome VARCHAR(30) NOT NULL,
   anno_nascita INTEGER,
   PRIMARY KEY (email));
```

Sintassi dell'istruzione CREATE TABLE

CREATE TABLE <*nome_tab*> (
<*nome_col*><*dominio*> [DEFAULT<*val*>] [<*vincolo_col*> [...]]
[, {<*nome_col*><*dominio*> [DEFAULT<*val*>] [<*vincolo_col*> [...]] |
<*vincolo_tab*>}[, ...]])

CREATE TABLE: Sintassi

Example

```
CREATE TABLE utente(
   email  VARCHAR(40) NOT NULL,
   nome VARCHAR(30) NOT NULL,
   cognome VARCHAR(30) NOT NULL,
   anno_nascita INTEGER,
   PRIMARY KEY (email));
```

Sintassi dell'istruzione CREATE TABLE

CREATE TABLE <*nome_tab*> (
<*nome_col*><*dominio*> [DEFAULT<*val*>] [<*vincolo_col*> [...]]
[, {<*nome_col*><*dominio*> [DEFAULT<*val*>] [<*vincolo_col*> [...]] |
<*vincolo_tab*>}[, ...]])

CREATE TABLE: Sintassi

Example

```
CREATE TABLE utente(
   email  VARCHAR(40) NOT NULL,
   nome VARCHAR(30) NOT NULL,
   cognome VARCHAR(30) NOT NULL,
   anno_nascita INTEGER,
   PRIMARY KEY (email));
```

Sintassi dell'istruzione CREATE TABLE

CREATE TABLE <*nome_tab*> (
<*nome_col*><*dominio*> [DEFAULT<*val*>] [<*vincolo_col*> [...]]
[, {<*nome_col*><*dominio*> [DEFAULT<*val*>] [<*vincolo_col*> [...]] |
<*vincolo_tab*>}[, ...]])

Domini

```
CREATE TABLE <nome_tab> (
   <nome_col><dominio> [DEFAULT<val>] [<vincolo_col> [...]]
   [, {<nome_col><dominio> [DEFAULT<val>] [<vincolo_col> [...]] |
      <vincolo_tab>}[, ...]])
```

Domini

1. Domini Elementari (predefiniti)
   - Carattere: singoli caratteri o stringhe anche di lunghezza variabile
   - Bit: flag booleani o stringhe
   - Domini Numerici esatti ed approssimati
   - data, ora, intervalli di tempo
2. Domini definiti dall'utente

Dominio Carattere

Dominio	**Descrizione**
char(n)	Stringhe di n caratteri (lunghezza fissa). A stringhe piu' corte sono aggiunti spazi in coda.
char	sinonimo di **char**(1)
varchar(n)	Stringhe di al piu' n caratteri

Domini Numerici: Tipi Numerici Esatti (I)

Permette di rappresentare valori interi o decimali mediante una rappresentazione in virgola fissa.

Dominio	**Descrizione**
smallint	Intero. 2 byte, range $[-2^{15}, 2^{15} - 1]$
integer	Intero. 4 byte, range $[-2^{31}, 2^{31} - 1]$

Domini Numerici: Tipi Numerici Esatti (II)

Permette di rappresentare valori interi o decimali mediante una rappresentazione in virgola fissa.

Dominio	**Descrizione**
numeric(prec,scala)	per calcoli esatti fino a 1000 cifre significative. Scala e' il numero di cifre dopo la virgola. Prec e' il numero di cifre significative Esempio: 22.4454 ha precisione 6 e scala 4. Gli interi hanno scala 0.
numeric	Memorizza numeri decimali fino alla precisione massima consentita dall'implementazione, e assume scala 0.
decimal(prec,scala)	come **numeric**(prec,scala) ma assume che *prec* sia limite inferiore alla precisione.

Domini Numerici: Tipi Numerici Approssimati

Permette di rappresentare valori numerici approssimati mediante una rappresentazione in virgola mobile.

Dominio	**Descrizione**
real	Numeri in virgola mobile. Tipicamente nel range $[10^{-37}, 10^{37}]$, con almeno 6 cifre corrette
double precision	Numeri in virgola mobile. Tipicamente nel range $[10^{-307}, 10^{307}]$, con almeno 15 cifre corrette
float(prec)	*prec* specifica la precisione minima accettabile come numero di cifre binarie.

Domini Temporali

Dominio	**Descrizione**
timestamp	Data e ora. Esempio:'10-may-2004 14:30:10
date	Data. Esempio: (formato raccomandabile 'anno-mese-giorno') '2009-07-22'
time	Ora. Esempio:'5.50 pm'
interval	Intervalli di tempo. Esempio: '1 day 12 hours 50 min 10 sec ago'

Domini Booleani

Dominio	Descrizione
boolean	Tipo booleano. Esempi di valori booleani: TRUE, FALSE/ 't','f'/'true','false','y','n'/'yes','no','1','0'

Domini definiti dall'utente

Domini definiti dall'utente

L'utente puo' definire i propri domini con l'istruzione:

```
CREATE DOMAIN
```

Example

```
CREATE DOMAIN provincia AS CHAR(2) NOT NULL;
```

Domini definiti dall'utente

Example

```
CREATE DOMAIN voto AS INTEGER
    CHECK (VALUE BETWEEN 18 AND 30)
```

oppure

```
CREATE DOMAIN voto AS INTEGER
    CHECK (VALUE > 17 AND VALUE ≤ 30)
```

Example

```
CREATE DOMAIN nat_pari AS INTEGER
    CONSTRAINT positivo CHECK (VALUE ≥ 0)
    CONSTRAINT positivo CHECK (VALUE %20)
```

Vincoli Intrarelazionali

Vincoli di Integrita' Intrarelazionali

- Proprieta' che devono essere soddisfatte da ogni istanza di relazione della base di dati.
- Si riferiscono a singole relazioni della base di dati.

Vincoli di Integrita' Intrarelazionali in SQL:

- NOT NULL: attributo non nullo
- UNIQUE: definisce (super-) chiavi
- PRIMARY KEY: definisce la chiave primaria (una sola, implica NOT NULL)
- CHECK(condizione): vincolo generico

NOT NULL

NOT NULL

- Il valore nullo non e' ammesso come valore dell'attributo.
- Il valore dell'attributo deve essere specificato in fase di inserimento.

Example

```
CREATE TABLE  Ditta.Impiegato (
   cod Ditta.dom_cod_impiegato PRIMARY KEY,
   nome VARCHAR(40) NOT NULL,
   stipendio Ditta.dom_stipendio
);
```

UNIQUE

UNIQUE

- Impone che i valori di un attributo (oppure di un insieme di attributi) siano una superchiave, quindi tuple differenti della stessa tabella non possono avere gli stessi valori.
- Si puo' definire su:
  - un solo attributo

    ```
    Matricola CHAR(6) UNIQUE
    ```
  - un insieme di attributi

    ```
    Nome VARCHAR(20)
    Cognome VARCHAR(20)
    UNIQUE(Nome,Cognome);
    ```

UNIQUE

UNIQUE su insieme di attributi

```
Nome VARCHAR(20) NOT NULL,
Cognome VARCHAR(20) NOT NULL,
UNIQUE(Nome,Cognome)
```

Impone che non ci siano due righe che abbiano uguale sia il nome che il cognome

UNIQUE su singoli attributi

```
Nome VARCHAR(20) NOT NULL UNIQUE,
Cognome VARCHAR(20) NOT NULL UNIQUE
```

Impone che non ci siano due righe che abbiano lo stesso nome o lo stesso cognome

PRIMARY KEY

- Specifica la chiave primaria della relazione
  - Si usa una sola volta per tabella
  - Implica una definizione di **not null**
- Due forme
  1. In definizione di un attibuto, se forma da solo chiave primaria

```
Matricola CHAR(20)PRIMARY KEY
```

  2. Come elemento separato (quando la chiave primaria e' composta da piu' attributi)

```
Nome VARCHAR(20),
Cognome VARCHAR(20),
PRIMARY KEY(Nome, Cognome)
```

CHECK

Example

```
CREATE TABLE  Impiegato (
   Matricola CHAR(6) PRIMARY KEY,
   Nome VARCHAR(20) NOT NULL,
   Cognome VARCHAR(20) NOT NULL,
   Stipendio FLOAT DEFAULT 1000,
   UNIQUE(Cognome,Nome),
   CHECK Stipendio ≥ 1000);
```

Vincoli intrarelazionali di colonna: Sintassi

Sintassi di specifica dei vincoli intrarelazionali definiti su singoli attributi:

[CONSTRAINT <*nome_vincolo*>]
{NOT NULL | UNIQUE | PRIMARY KEY | CHECK(<*condizione*>)}

Vincoli intrarelazionali di tabella: Sintassi

Sintassi di specifica dei vincoli intrarelazionali definiti su insiemi di attributi:

[CONSTRAINT <*nome_vincolo*>]
{ PRIMARY KEY(<*nome_colonna*> [, ...]) |
  UNIQUE(<*nome_colonna*> [, ...]) |
  CHECK(<*condizione*>)}

Vincoli di Integrita' Interelazionali

Vincoli di Integrita' Interelazionali

- Vincoli che coinvolgono piu' relazioni
- I piu' significativi sono i vincoli di integrita' referenziale o vincoli di riferimento
- In SQL i vincoli di integrita' referenziale sono definiti mediante il concetto di chiave esterna o chiave esportata.

Chiave Esterna

Chiave Esterna

- Crea un legame tra:
  - i valori (di uno o piu') attributi della tabella corrente (tabella interna)
  - i valori (di uno o piu') attributi di un'altra tabella (tabella esterna)
- Impone che per ogni riga della tabella interna il valore dell'attributo (l'insieme di valori di attributi) sia presente tra i valori (l'insieme di valori) di un attributo della tabella esterna dell'attributo
- L'attributo della tabella esterna a cui si fa riferimento deve essere soggetto a vincolo `UNIQUE` o `PRIMARY KEY` (vincolo di tabella se sono coinvolti piu' di un attributo)

Chiave esterna

`REFERENCES` e `FOREIGN KEY`

SQL fornisce due costrutti di base per la definizione di chiavi esterne:

1. Costrutto `REFERENCES`, per la definizione di chiavi esterne su un attributo (mediante vincolo di colonna)
2. Costrutto `FOREIGN KEY` per la definizione di chaivi esterne su piu' attributi (mediante vincolo di tabella)

Costrutto REFERENCES

REFERENCES

- Si usa il solo costrutto REFERENCES quando il vincolo e' definito su un singolo attributo
- Con REFERENCES (nella tabella interna) si specificano:
  - la tabella esterna, e
  - l'attributo della tabella esterna con il quale l'attributo della tabella interna deve essere legato

Uso del Costrutto REFERENCES

Example

```
CREATE TABLE dipartimento(
   nome_dip VARCHAR(15) PRIMARY KEY,
   sede VARCHAR(20) NOT NULL);


CREATE TABLE impiegato(
   matricola CHAR(6) PRIMARY KEY,
   nome VARCHAR(20) NOT NULL,
   cognome VARCHAR(20) NOT NULL,
   nome_dpt VARCHAR(15) REFERENCES  dipartimento(nome_dip));
```

Costrutto FOREIGN KEY

FOREIGN KEY

- Si usa il costrutto FOREIGN KEY+REFERENCES quando il vincolo e' definito su un insieme di attributi
  - Con FOREIGN KEY nella tabella interna si elencano gli attributo della tabella interna coinvolti nel legame
  - Con REFERENCES nella tabella tabella interna si specificano la tabella esterna e gli attributi della tabella esterna con i quali gli attributi della tabella interna devono essere legati

Uso del Costrutto FOREIGN KEY

Example

```
CREATE TABLE anagrafica(
   codice_fiscale CHAR(11) PRIMARY KEY,
   nome VARCHAR(20) NOT NULL,
   cognome VARCHAR(20) NOT NULL,
   UNIQUE(nome, cognome));


CREATE TABLE impiegato(
   matricola CHAR(6) PRIMARY KEY,
   nome VARCHAR(20) NOT NULL,
   cognome VARCHAR(20) NOT NULL,
   nome_dpt VARCHAR(15) REFERENCES  dipartimento(nome_dip),
   FOREIGN KEY(nome, cognome) REFERENCES anagrafica(nome, cognome));
```

PostgreSQL

PostgreSQL

- Sistema di gestione di basi di dati relazionali ad oggetti sviluppato in varie fasi sin dal 1977
- E' largamente considerato il sistema di gestione di basi di dati open source piu' avanzato
- L'interazione tra un utente e la base di dati segue il modello client-server

Lavorare in `psql`

Client testuale `psql`

- `psql` e' un client a riga di comando, distribuito insieme a PostgreSQL,
- `psql`, pur essend semplice, e' molto potente e permette l'interazione diretta con il server postgreSQL

Accesso all'archivio

Accesso all'archivio

```
psql -U <nome_utente> -h <hostname> <database>
```

Subito dopo l'avvio di `psql` appare un breve sommario di alcuni comandi slash:

- `\h` per l'aiuto su SQL
- `\?` per l'aiuto sui comandi `psql`
- `\q` per uscire da `psql` una volta terminato

Comandi di slash

- `\a`
  Attiva e disattiva la modalita' di allineamento
- `\c[onnect] [nomedb | - [utente]]`
  Si connette ad un nuovo database
- `\C <titolo>`
  Titolo della tabella
- `\d <table>`
  Descrive la tabella (o la vista)
- `\d{t|i|s|v}`
  Elenca tabelle/indici/sequenze/viste
- `\d{p|S|l}`
  Elenca permessi/tabelle di sistema/large objects
- `\da` Elenca gli aggregati

Comandi di Slash

- `\dd [oggetto]`
  Elenca i commenti per tabella, tipo, funzione o operatore
- `\df`
  Elenca le funzioni
- `\do`
  Elenca gli operatori
- `\dT`
  Elenca i tipi di dati
- `\e [file]`: Permette la modifica del buffer della query attuale o di [file] con un editor esterno
- `\echo <testo>`: scrive il testo sullo stdout
- `\f <sep>`: Cambia il separatore dei campi

Comandi di Slash

- `\g [file]`
  Invia la query al backend (e i risultati in [file])
- `\h [cmd]`
  aiuto sulla sintassi dei comandi SQL
- `\H`
  Attiva o disattiva la modalita' HTML (attualmente disinserita)
- `\i <file>`
  Legge ed esegue la query da file
- `\l`: Elenca tutti i databases
- `\o [file]`: Invia tutti i risultati della query nel [file]
- `\p`: mostra il contenuto del buffer della query attuale
- `\q`: Esce da `psql`

Comandi di Slash

- \r Cancella il buffer della query
- \s [file] Stampa lo storico dei comandi e lo salva in [file]
- \set <var> <valore> Imposta una variabile interna
- \t Visualizza solo le righe
- \T <tags> Imposta gli attributi dei tag HTML per le tabelle
- \unset <var> Elimina una variabile interna
- \w <file> scrive il buffer della query attuale su ¡file¿
- \x Attiva e disattiva l'output esteso
- \z Elenca i permessi di accesso alle tabelle
- \d![cmd] Esce sulla shell o esegue un comando

Domini in PostgreSQL

Caratteri

Oltre ai domini predefiniti per le stringhe previsti dallo standard SQL, PostgreSQL accetta:

- `character varying`, o `varchar` (senza specificare una dimensione massima) per stringhe di lunghezza arbitraria.
- `text` per stringhe di arbitraria lunghezza

Domini in PostgreSQL

Numeri Interi

Oltre ai domini predefiniti per i valori numerici interi previsti dallo standard SQL, PostgreSQL:

- definisce un tipo `bigint` che usa 8 byte.

Domini in PostgreSQL

Numeri decimali con precisione arbitraria

Estensioni allo standard SQL PostgreSQL:

- `numeric` memorizza numeri decimali fino alla precisione massima consentita, senza forzare una scala
- `numeric` e `decimal` sono equivalenti in PostgreSQL

Domini in PostgreSQL

Numeri in virgola mobile

Estensioni allo standard SQL in PostgreSQL:

- `float(1) -- foat(24)` equivale a `real`
- `float(25) -- foat(53)` equivale a `double precision`
- altri valori danno errore

Riepilogo e Sommario

A riepilogo dei concetti introdotti, consideriamo la definizione in PostgreSQL della BD per la memorizzazione di un gruppo di persone e dei corrispondenti rapporti genitore/figlio, costituita dalle tabelle:

- `persone(id, nome, reddito,eta,sesso)` soggetta ai vincoli:
  - `id`: (stringa di due caratteri) e' chiave primaria
  - `nome`: (stringa di 20 caratteri) non puo' essere `NULL`
  - `reddito`: (intero), in migliaia di euro, 0 per default
  - `eta`: intero < 200, `sesso`: 'M' oppure 'F'
- `genitori(figlio,genitore,)` soggetta ai vincoli:
  - `figlio` (stringa di 2 caratteri) chiave esterna su `persone`
  - `genitore` (stringa di 2 caratteri) chiave esterna su `persone`
  - la chiave primaria e' costituita da (figlio,genitore).

Riepilogo e Sommario

Example

```
CREATE TABLE persone(
  id CHAR(2) PRIMARY KEY,
  nome VARCHAR(20) NOT NULL,
  reddito INT DEFAULT 0,
  eta SMALLINT,
  sesso CHAR CHECK(sesso='M' OR sesso'F'));


CREATE TABLE genitori(
  figlio CHAR(2) REFERENCES  persone(id),
  genitore CHAR(2) REFERENCES  persone(id)
  PRIMARY KEY(figlio,genitore));
```

Bibliografia ed Approfondimenti

Bibliografia ed Approfondimenti

- R.A.Elmasri, S.B. Navathe. Sistemi di Basi di Dati – Fondamenti: Capitolo 8 (8.1).
- Capitolo 8 (Data Types) e capitolo 5 (Data Definition) del manuale di PostgreSQL (http://www.postgresql.org/docs/manuals/)