

# Il linguaggio SQL: DDL di base

Sistemi Informativi T

Versione elettronica: [04.1.SQL.DDLbase.pdf](#)

# SQL: caratteristiche generali

- SQL (Structured Query Language) è il linguaggio *standard de facto* per DBMS relazionali, che riunisce in sé funzionalità di DDL, DML e DCL
- SQL è un linguaggio **dichiarativo** (non-procedurale), ovvero **non specifica la sequenza di operazioni da compiere per ottenere il risultato**
- SQL è “**relazionalmente completo**”, nel senso che **ogni espressione dell'algebra relazionale può essere tradotta in SQL**
  - ...inoltre SQL fa molte altre cose...
- Il modello dei dati di SQL è basato su **tabelle anziché relazioni**:
  - Possono essere presenti righe (tuple) duplicate
  - In alcuni casi l'ordine delle colonne (attributi) ha rilevanza
- ...il motivo è pragmatico (ossia legato a considerazioni sull'efficienza)
- SQL adotta la **logica a 3 valori** introdotta con l'Algebra Relazionale

# SQL: standard e dialetti

- Il processo di standardizzazione di SQL è iniziato nel 1986
- Nel 1992 è stato definito lo standard **SQL-2** (o SQL-92) da parte dell'**ISO** (International Standards Organization), e dell'**ANSI** (American National Standards Institute), rispettivamente descritti nei documenti ISO/IEC 9075:1992 e ANSI X3.135-1992 (identici!)
- Del 1999 è lo standard **SQL:1999**, che rende SQL un **linguaggio computazionalmente completo** (e quindi con istruzioni di controllo!) per il supporto di oggetti persistenti...
- Allo stato attuale **ogni sistema ha ancora un suo dialetto**:
  - supporta (in larga parte) SQL-2
  - ha già elementi di SQL:1999
  - ha anche costrutti non standard
- Quello che vediamo è la parte più “diffusa”

# Organizzazione del materiale

- La trattazione di SQL viene suddivisa in più parti come segue:
  - DDL di base
  - DML “per gli operatori dell’algebra” e per le operazioni di modifica dei dati
    - Per fare “quello che si fa anche in algebra”
  - DML per il raggruppamento dei dati
    - Per derivare informazioni di sintesi dai dati
  - DML con blocchi innestati
    - Per scrivere richieste complesse
  - DDL per la definizione di viste e vincoli generici
    - Per migliorare la qualità dei dati
  - DCL per la definizione di transazioni
    - Per eseguire modifiche multiple senza rischi di inconsistenze

# Data Definition Language (DDL)

- Il DDL di SQL permette di definire **schemi di relazioni** (o “**table**”, tabelle), modificarli ed eliminarli
- Permette di inoltre di specificare **vincoli**, sia a livello di tupla (o “**riga**”) che a livello di tabella
- Permette di definire nuovi **domini**, oltre a quelli predefiniti
  - Per **vincoli e domini** si può anche fare uso del **DML** (quindi inizialmente non si trattano completamente)
- Inoltre si possono definire **viste** (“view”), ovvero tabelle virtuali
  - Altro ancora (ad es. **schemi** = spazi di nomi) si vedranno in laboratorio

# Creazione ed eliminazione di tabelle

- Mediante l'istruzione **CREATE TABLE** si definisce lo **schema di una tabella** e se ne crea un'**istanza vuota**
- Per ogni **attributo** va specificato il **dominio**
  - E' anche possibile impostare un eventuale **valore di default** e specificare dei **vincoli**
- Infine possono essere espressi **altri vincoli** a livello di tabella
- Mediante l'istruzione **DROP TABLE** è possibile eliminare lo schema di una tabella (e conseguentemente la corrispondente istanza)

**DROP TABLE Imp**

# Definizione di tabelle: es. con solo domini

```
CREATE TABLE Studenti (  
    Matricola char(5),  
    CF char(16),  
    Cognome varchar(30),  
    Nome varchar(30),  
    DataNascita date,  
    Email varchar(100) )
```

```
CREATE TABLE Corsi (  
    CodCorso int,  
    Titolo varchar(50),  
    Docente varchar(30),  
    Anno smallint )
```

```
CREATE TABLE Esami (  
    Matricola char(5),  
    CodCorso int,  
    Voto smallint,  
    Lode char(2) )
```

# I tipi di dato più comuni (1)

**CHAR(n)**: stringhe di lunghezza fissa  $n$ ,  $1 \leq n \leq 254$

- **CHAR** equivale a **CHAR(1)**

**VARCHAR(n)**: stringhe di lunghezza variabile  $\leq n$  ( $n \leq 32672$ )

**INTEGER** o **INT**: interi a 4 byte

**SMALLINT**: interi a 2 byte

**DECIMAL(p,s)** o **DEC(p,s)**: numeri decimali

**p** precisione (numero totale di cifre),  $1 \leq p \leq 31$

**s** scala (numero di cifre a destra del punto decimale),  $0 \leq s \leq p$

**REAL**: numeri reali a 32 bit, singola precisione floating-point

**DOUBLE** o **FLOAT**: numeri reali a 64 bit, doppia precisione floating-point

# I tipi di dato più comuni (2)

**DATE:** 10 byte - DD-MM-YYYY (giorno, mese e anno)  
**TIME:** 8 byte - HH.MM.SS (ore, minuti e secondi)  
**TIMESTAMP:** 26 byte - YYYY-MM-DD-HH-MM-SS-NNNNNNN  
(anno, mese, giorno, ora, minuti, sec. e nanosec.)

**BOOLEAN:** in DB2 il tipo boolean NON può essere usato per definire il dominio di un attributo

- Nei sistemi SQL (incluso DB2) è anche possibile definire nuovi tipi di dato (**User Data Types**, o **UDT**), ma non li trattiamo

# “Opzioni di colonna”: Valori nulli e di default

- Per vietare la presenza di valori nulli, è sufficiente imporre il vincolo **NOT NULL**

**CF**            **char(16)**            **NOT NULL**

- Per ogni attributo è possibile specificare un **valore di default**, che verrà usato se all’atto dell’inserimento di una tupla non viene fornito un valore per quell’attributo

**Anno**            **smallint**            **DEFAULT 1**

**Email varchar(100)**   **DEFAULT 'guest@studio.unibo.it'**

# “Opzioni di colonna”: Chiavi

- La definizione di una chiave avviene esprimendo un vincolo **UNIQUE**

**CF**                      **char(16)**                      **UNIQUE**

- Per specificare un vincolo di chiave primaria la sintassi è

**Matricola**   **char(5)**                      **PRIMARY KEY**

- In DB2 è obbligatorio aggiungere il vincolo NOT NULL, ovvero **non si possono dichiarare chiavi con valori nulli**, quindi:

**CF**                      **char(16)**                      **NOT NULL UNIQUE**

**Matricola**   **char(5)**                      **NOT NULL PRIMARY KEY**

- Si noti che la specifica di una chiave primaria non è obbligatoria, e che si può specificare al massimo una chiave primaria per tabella

# “Opzioni di colonna”: Foreign key

- La definizione di una foreign key avviene specificando un vincolo **FOREIGN KEY**, e indicando quale chiave viene referenziata

```
CodCorso int REFERENCES Corsi(CodCorso)
```

- Se si omettono gli attributi destinazione, vengono assunti quelli della chiave primaria

```
CodCorso int REFERENCES Corsi
```

- Nell'esempio, **Esami** è detta **tabella di riferimento** e **Corsi** **tabella di destinazione** (analogia terminologia per gli attributi coinvolti)
- Le colonne di destinazione devono essere una chiave della tabella destinazione (non necessariamente la chiave primaria)

# “Opzioni di colonna”: Vincoli generici

- Mediante la clausola **CHECK** è possibile esprimere vincoli di tupla arbitrari, sfruttando tutto il potere espressivo di SQL

- La sintassi è: **CHECK (<condizione>)**

**CHECK ((Lode = 'SI') OR (Lode = 'NO'))**

- Il vincolo è **violato** se esiste almeno una tupla che rende **falsa** la <condizione>. Pertanto

**Stipendio dec(7,2) CHECK (Stipendio > 0),**

non permette tuple con stipendio negativo, ma **ammette valori nulli** per l'attributo **Stipendio** (perché non posso essere sicuro che sia negativo!)

- Se ciò non è il comportamento voluto:

**Stipendio dec(7,2) NOT NULL CHECK (Stipendio > 0),**

# Vincoli con nomi

- A fini diagnostici (e di documentazione) è spesso utile sapere quale vincolo è stato violato a seguito di un'azione sul DB
- A tale scopo è possibile dare dei **nomi ai vincoli**, ad esempio:

```
Voto smallint NOT NULL
```

```
    CONSTRAINT VotoValido
```

```
    CHECK ((Voto >= 18) AND (Voto <=30)),
```

```
CF char(16) NOT NULL CONSTRAINT CFKey UNIQUE,
```

```
Stipendio dec(7,2) CONSTRAINT StipendioPositivo  
    CHECK (Stipendio > 0),
```

# Definizione di tabelle con column options (1)

```
CREATE TABLE Studenti (  
    Matricola char(5) NOT NULL PRIMARY KEY,  
    CF char(16) NOT NULL UNIQUE,  
    Cognome varchar(30) NOT NULL,  
    Nome varchar(30) NOT NULL,  
    DataNascita date NOT NULL,  
    Email varchar(100) DEFAULT 'guest' )
```

```
CREATE TABLE Esami (  
    Matricola char(5) NOT NULL REFERENCES Studenti,  
    CodCorso int NOT NULL REFERENCES Corsi,  
    Voto smallint NOT NULL CONSTRAINT VotoValido  
        CHECK ((Voto >= 18) AND (Voto <=30)),  
    Lode char(2) NOT NULL  
        CHECK ((Lode = 'SI') OR (Lode = 'NO')) )
```

# Definizione di tabelle con column options (2)

```
CREATE TABLE Corsi (  
    CodCorso    int        NOT NULL PRIMARY KEY,  
    Titolo      varchar(50) NOT NULL,  
    Docente      varchar(30),  
    Anno        smallint    DEFAULT 1  
                CHECK ((Anno >= 1) AND (Anno <= 3)) )
```

E' evidente che con le sole column options non si riescono a specificare tutti i vincoli necessari...

# Vincoli di tabella

- Tutti i vincoli sinora visti (UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK) si possono anche dichiarare separatamente dagli attributi, ad es:

```
PRIMARY KEY (Matricola,CodCorso)
```

```
UNIQUE (Cognome,Nome)
```

```
CHECK ((Voto = 30) OR (Lode = 'NO'))
```

con la sintassi già vista

- La sola eccezione è data dai vincoli di foreign key:

```
FOREIGN KEY (CodCorso) REFERENCES Corsi(CodCorso)
```

- Resta sempre possibile dare un nome ai vincoli:

```
CONSTRAINT StranoVincolo UNIQUE (Cognome,Nome)
```

- Quando il vincolo riguarda un singolo attributo lo si può esprimere a livello di tabella o di riga, indifferentemente

# Definizione con table constraints

```
CREATE TABLE Esami (  
    Matricola char(5)    NOT NULL REFERENCES Studenti,  
    CodCorso  int        NOT NULL REFERENCES Corsi,  
    Voto      smallint    NOT NULL CONSTRAINT VotoValido  
                CHECK ((Voto >= 18) AND (Voto <=30)),  
    Lode      char(2)     NOT NULL  
                CHECK ((Lode = 'SI') OR (Lode = 'NO')),  
    PRIMARY KEY (Matricola,CodCorso),  
    CONSTRAINT LodeValida CHECK  
                ((Voto = 30) OR (Lode = 'NO'))    )
```

# Modifica di tabelle

- Mediante l'istruzione **ALTER TABLE** è possibile modificare lo schema di una tabella, in particolare:
  - Aggiungendo o rimuovendo attributi
  - Aggiungendo o rimuovendo vincoli

```
ALTER TABLE Imp
```

```
ADD COLUMN Sesso char
```

```
ADD CONSTRAINT StipendioMax CHECK (Stipendio < 4000)
```

```
DROP CONSTRAINT StipendioPositivo
```

```
DROP UNIQUE(Cognome, Nome) ;
```

- Se si aggiunge un attributo con vincolo NOT NULL, bisogna prevedere un valore di default, che il sistema assegnerà automaticamente a tutte le tuple già presenti

```
ADD COLUMN Istruzione char(10) NOT NULL DEFAULT 'Laurea'
```

# Riassumiamo:

- Il linguaggio SQL è lo standard de facto per interagire con DB relazionali
- Si discosta dal modello relazionale in quanto permette la presenza di tuple duplicate (**tabelle anziché relazioni**)
- La **definizione delle tabelle** permette di esprimere **vincoli** di vario tipo (UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK), di specificare se un attributo può o meno assumere valori nulli e di assegnargli un eventuale valore di default
- Altri elementi del DDL verranno introdotti in seguito, in quanto legati ad aspetti del DML (politiche di reazione, viste, ecc.)