

Corso di Laurea in Ingegneria Clinica e Biomedica  
Informatica Medica I

**SQLITE E IL LINGUAGGIO SQL**

*Prof. Aleksandar Miladinović*

# Operazioni su database relazionali: SQL

- SQL (**S**tructured **Q**uery **L**anguage): linguaggio di interrogazione per basi di dati relazionali – **DATA MANIPULATION LANGUAGE**
- SQL è un **ISO/ANSI standard**: indipendente dal DBMS su cui è implementato;
- **SQL è un LINGUAGGIO DICHIARATIVO**: specifica le CARATTERISTICHE DEL RISULTATO e NON la PROCEDURA per ottenerlo;
- SQL utilizza i termini **Tabella, Riga, Colonna** che corrispondono a Relazione, Tupla, Attributo nel modello relazionale.
- SQL definisce le operazioni di
  - Definizioni di dati (schema e istanze)
  - Istruzioni di aggiornamento (schema e istanze)
  - Interrogazioni

# FUNZIONI DI SQL

- CREAZIONE DEL DATABASE
  - CREAZIONE DELLO SCHEMA
  - POPOLAMENTO DEL DATABASE
- MODIFICA DEL DATABASE
  - MODIFICA DELLO SCHEMA
  - MODIFICA DELLE ISTANZE
- INTERROGAZIONE DEL DATABASE
  - ESTRAZIONE DELLE INFORMAZIONI
  - CREAZIONE DI VISTE

# SQLITE

- Database relazionale “embedded” → non richiede un DBMS
- nasce per essere “portabile” →
  - Self embedded database
  - Di facile utilizzo
  - Può essere utilizzato su qualsiasi macchina senza necessità di software aggiuntivo
- Lavora bene con applicazioni piccole o medie
- Lavora con la maggior parte dei linguaggi di programmazione

# CREAZIONE DEL DATABASE - SQL

- Istruzione per la creazione di una nuova base di dati

**CREATE DATABASE** <nomeDataBase>;

- Istruzione per la cancellazione di una base di dati

**DROP DATABASE** <nomeDataBase>;

# CREATE DATABASE SQLite

- È sufficiente aprire il database ('NomeDB.db'):
  - Se non esiste → il DB viene creato
  - Se esiste --> il DB viene aperto
- Per aprire il database da terminale:
  - Alla chiamata dell'applicazione  
`Sqlite3 NomeDB.db`
  - All'interno dell'applicazione  
`sqlite> .open NomeDB.db`

# CREAZIONE TABELLA

- Istruzione di creazione

```
CREATE TABLE <nomeTabella> (  
    <nomeAtt1> <TIPOATTRIBUTO> [DEFAULT] [VINCOLI],  
    <nomeAtt2> <TIPOATTRIBUTO> [DEFAULT] [VINCOLI],  
    ...  
    <nomeAttN> <TIPOATTRIBUTO> [DEFAULT] [VINCOLI],  
  
    [ALTRI VINCOLI – es. Integrità referenziale]  
);
```

# TIPI DI DATO PRINCIPALI - SQL

- **INTERO: INTEGER**
- **DECIMALE: DECIMAL (M,N)**
  - M=numero totale di cifre
  - N= numero di cifre dopo la virgola
  - **FLOAT** – approssimato precisione 16 cifre
- **STRINGA:**
  - **CHARACTER (N)** – stringa di lunghezza fissa N
  - **VARCHAR (N)** – stringa di lunghezza variabile, massimo N caratteri
- **BOOLEANO: BOOLEAN**
- **BINARIO**
  - **BINARY (N)** – binario di lunghezza fissa N
  - **VARBINARY (N)** – binario di lunghezza variabile, massimo N
- **DATA**
  - **DATE - 'YYYY-MM-DD'**
  - **DATETIME 'YYYY-MM-DD HH:MM:SS'**
  - **TIMESTAMP**

# TIPI DI DATO SQLite

- SQLite usa solamente alcuni tipi di dato base:
  - TEXT
  - INTEGER
  - NUMERIC
  - REAL
  - NONE
- Per essere compatibile con altri DB, SQLite permette l'utilizzo dei tipi di dato classici e li mappa sui suoi tipi di dato.
  - Char/varchar/text, ... → TEXT (ignora la lunghezza)
  - INT, SMALLINT, INTEGER, ... → INTEGER
  - DECIMAL, NUMERIC → NUMERIC
  - DOUBLE, FLOAT, ... → REAL
  - DATE, .. → NUMERIC

# SYSTEM TABLES

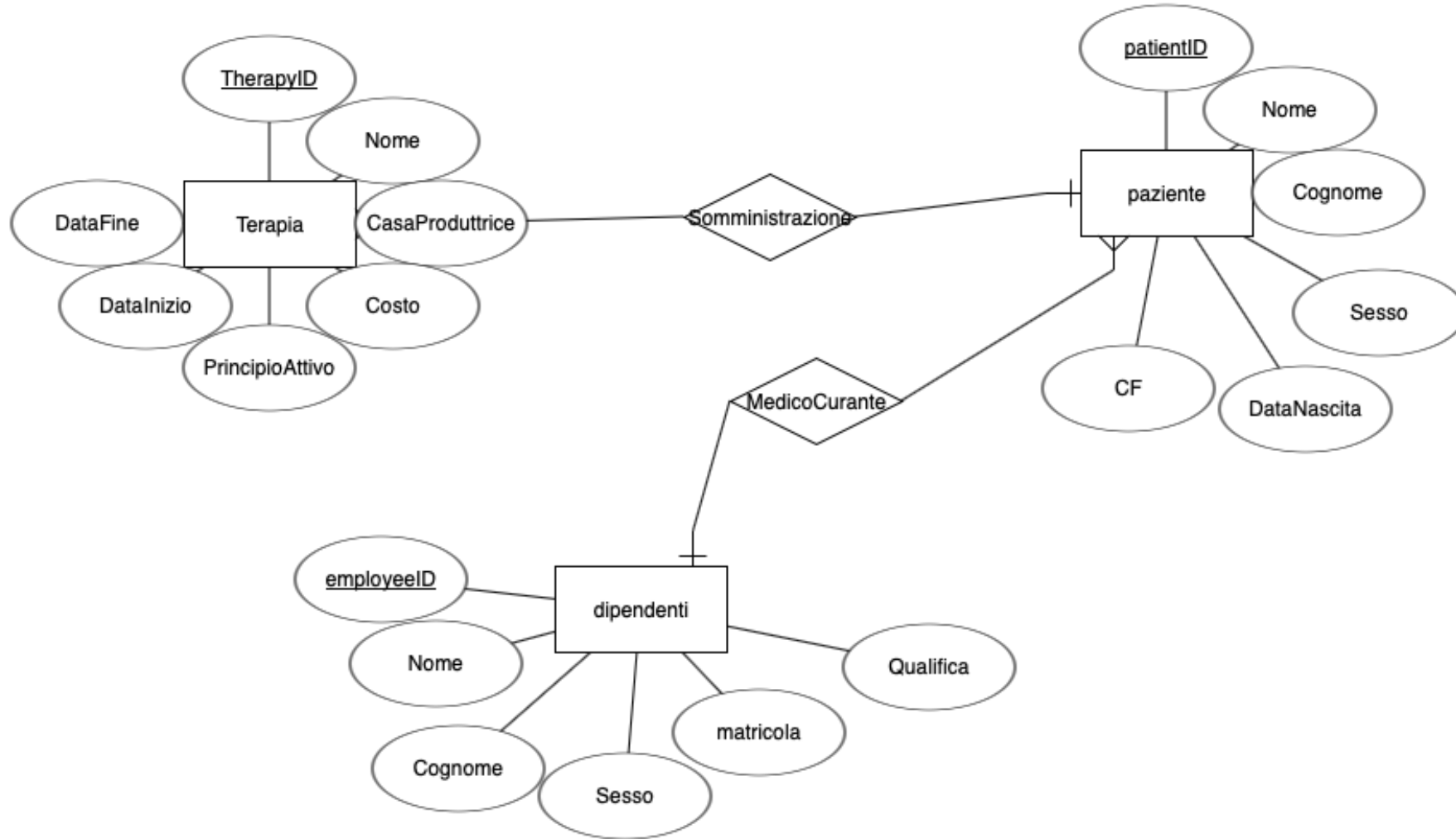
SQLite genera delle tabelle di sistema

System Table	Description
sqlite_master	Master listing of all database objects in the database and the SQL used to create each object.
sqlite_sequence	<p>Lists the last sequence number used for the AUTOINCREMENT column in a table.</p> <p>The <i>sqlite_sequence</i> table will only be created once an AUTOINCREMENT column has been defined in the database and at least one sequence number value has been generated and used in the database.</p>
sqlite_stat1	<p>This table is created by the <a href="#">ANALYZE command</a> to store statistical information about the tables and indexes analyzed.</p> <p>This information will be later used by the query optimizer.</p>

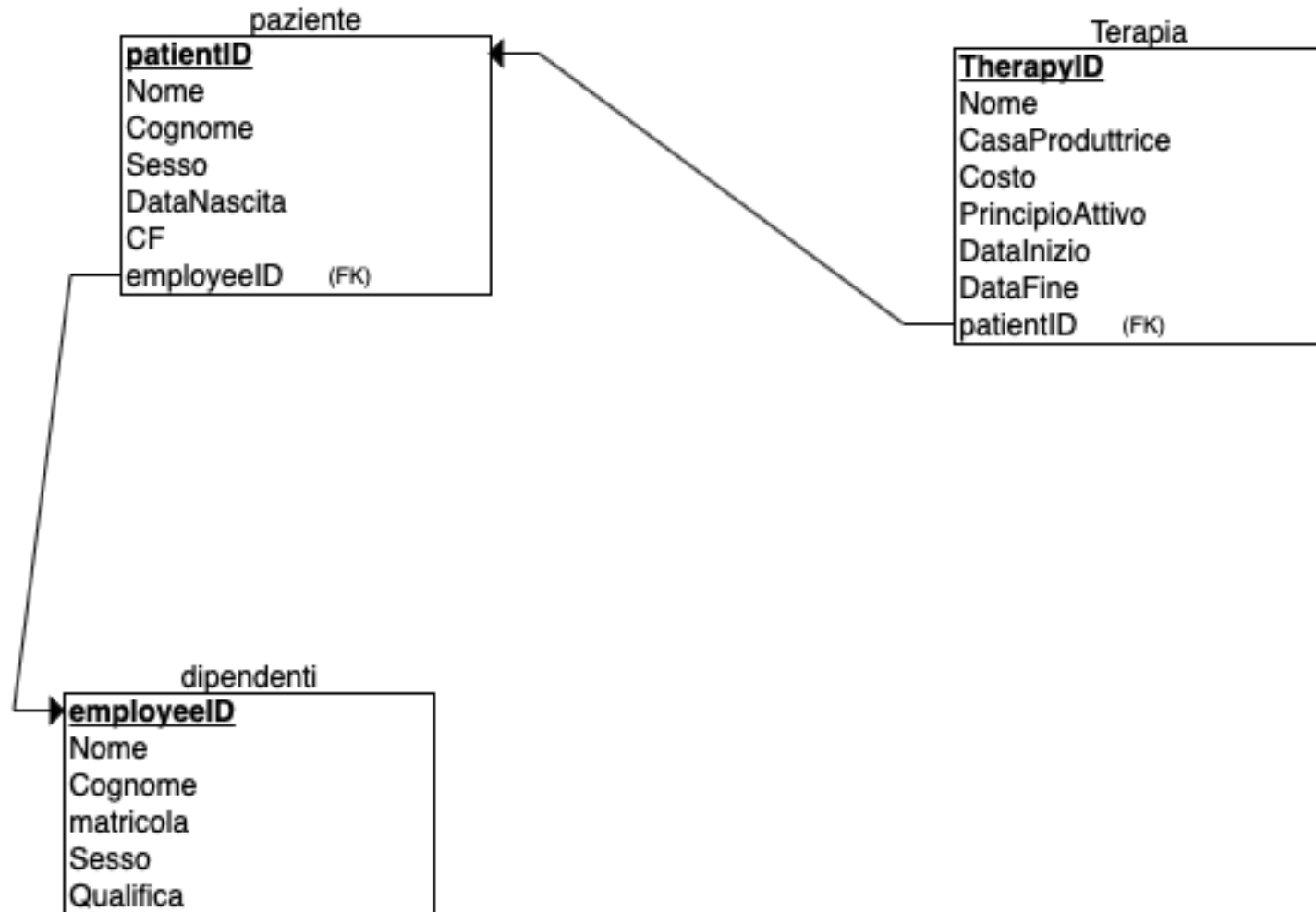
# ISTRUZIONI UTILI SQLite

- `.open` → apertura/creazione database
- `.database` → visualizzazione DB di lavoro
- `.tables` → visualizza le tabelle nel DB (o una specifica)
- `.schema` → visualizza le istruzioni CREATE per lo schema attuale
- `.show` → definisce le opzioni di visualizzazione
- `.mode` → opzioni visualizzazione
  - `column` → visualizzazione a colonne (`.width` definisce le larghezza delle colonne, un valore per colonna)
  - Altri esempi: `list`, `csv`, `html`, `line`, etc
- `. Headers on/off` → visualizza le intestazioni delle colonne

# ESEMPIO



# ESEMPIO



# CREAZIONE TABELLA PAZIENTE

PatientID	Nome	Cognome	Sesso	Data Nascita	CF
1	Anna	Rossi	F	11/3/62	RSSNNA62C51L872P
2	Roberto	Marzio	M	4/25/71	MRZRRT71D04F251R
3	Giampiero	Di Nicola	M	7/3/37	DNCGPR37L07H821Q
4	Luciana	Nunziatella	F	5/11/55	NNZLCN55R45F205N
5	Arianna	Lucchini	F	2/2/82	LCCRNN82B42N127C

```
CREATE TABLE paziente (patientID INTEGER PRIMARY KEY  
AUTOINCREMENT, Nome VARCHAR (20), Cognome VARCHAR (20), Sesso  
CHAR, DataNascita DATE DEFAULT NULL);
```

# TIPO “DATA”

- La data di nascita (formato DATE) deve essere inserito come stringa (*timestring*):  
    ‘YYYY-MM-DD’ oppure ‘YYYY-MM-DD HH:MM:SS’
- Esiste anche la timestring ‘now’ → restituisce la stringa corrispondente al momento attuale (GMT)
- SQLite utilizza i seguenti operatori
  - **date**(*timestring, modifier, modifier, ...*) → ritorna ‘YYYY-MM-DD’
  - **time**(*timestring, modifier, modifier, ...*) → ritorna ‘HH:MM:SS’
  - **datetime**(*timestring, modifier, modifier, ...*) → ritorna ‘YYYY-MM-DD HH:MM:SS’
  - **julianday**(*timestring, modifier, modifier, ...*) → ritorna il numero di giorni da 1/1/4713 AC
  - **strftime**(*format, timestring, modifier, modifier, ...*) → ritorna la stringa come formattata in “format”

# ESEMPI

- **sqlite> select date('now');**

date('now')

-----

2020-10-19

- **sqlite> select time('now');**

time('now')

-----

10:46:09

- **sqlite> select datetime('now');**

datetime('now')

-----

2020-10-19 10:46:13

- **sqlite> select julianday('now');**

julianday('now')

-----

2459141.9474229

- **sqlite> select julianday('now')/365;**

julianday('now')/365

-----

6737.37519888702

- **sqlite> select strftime('%Y','now');**

strftime('%Y','now')

-----

2020

# ESEMPI - MODIFICATORI

• **sqlite> select datetime('now','-3 days');**

`datetime('now','-3 days')`

-----

2020-10-16 10:49:08

• **select datetime('now','start of year');**

`datetime('now','start of year')`

-----

2020-01-01 00:00:00

## POSSIBILI MODIFICATORI

- 1.NNN days
- 2.NNN hours
- 3.NNN minutes
- 4.NNN.NNNN seconds
- 5.NNN months
- 6.NNN years
- 7.start of month
- 8.start of year
- 9.start of day
- 10.weekday N
- 11.unixepoch
- 12.localtime
- 13.utc

# MODIFICA TABELLA

- Cancellazione tabella

**DROP TABLE** <NomeTabella>

- Modifica tabella: aggiungere una Colonna

**ALTER TABLE** <NomeTabella> **ADD** <nomeAtt> <Tipo>

- Modifica tabella:eliminare una Colonna

**ALTER TABLE** <NomeColonna> **DROP COLUMN** <nomeAtt>;

ALTER TABLE paziente ADD cf VARCHAR(16);

# INSERIMENTO DATI

- Istruzione base

**INSERT INTO** <nomeTabella> **VALUES** (Val1, Val2, ..., ValN)

N=numero attributi

- Inserimento selettivo

**INSERT INTO** <nomeTabella> (*Att1, Att2,...*) **VALUES** (*Val1,Val2,...*)

Si inseriscono solo I valori dichiarati nella lista (*Att1,Att2, ...*) nell'ordine dichiarato

```
INSERT INTO paziente VALUES (1,'Anna','Rossi', 'F','1962-03-11','RSSNNA62C51L872P');
```

```
INSERT INTO paziente (Nome, Cognome, Sesso, DataNascita,cf)  
VALUES ('Roberto','Marzio','M','1971-04-25','MRZRRT71D04F251R');
```

# VINCOLI

- **PRIMARY KEY** – l'attributo è la chiave primaria della tabella
- **NOT NULL** – l'attributo deve essere sempre dichiarato (è sottointeso in PRIMARY KEY)
- **UNIQUE** – l'attributo deve essere univoco
- **AUTO\_INCREMENT (o AUTOINCREMENT)** – tipico dell'indice che è Primary key della tabella (fa aumentare l'indice automaticamente senza doverlo inserire)
- **DEFAULT** - valore di default che l'attributo deve assumere

# MODIFICA VALORI

- AGGIORNAMENTO VALORE (Istruzione base)

```
UPDATE <NomeTabella>  
SET <NomeAtt1> = VALORE1 {, <NomeAtt2> = VALORE2,... <NomeAttN> = VALOREN }  
{WHERE [condizione] };
```

In assenza della clausola WHERE la modifica ricade su tutte le righe  
Possono essere modificate più colonne contemporaneamente

- CANCELLAZIONE RIGA

```
DELETE FROM <NomeTabella>  
{WHERE [condizione]};
```

# INTERROGAZIONE: SINTASSI BASE

Modello di Interrogazione generica

**SELECT** <lista di attributi>

**FROM** <lista delle tabelle>

**WHERE** <condizione>

```
SELECT Nome,Cognome  
FROM paziente  
WHERE Nome='Anna';
```

# DAL LINGUAGGIO NATURALE ALLA QUERY

Traduzione di una query di select in linguaggio naturale  
(traduzione linguaggio dichiarativo in procedurale):

1. Tra le righe ottenute dal prodotto cartesiano delle tabelle elencate nella clausola **from**,
2. vengono considerate quelle righe che soddisfano la condizione espressa nella clausola **where**;
3. su tali righe vengono valutate le espressioni sulle colonne indicate nella clausole **select**.

# INTERROGAZIONE: OPZIONI DI VISUALIZZAZIONE

- ORDER BY – presenta i risultati secondo un ordinamento

**ORDER BY** <NomeAtt> <TIPOLOGIA>

TIPOLOGIA = ASC (default), DESC

- LIMIT – presenta un numero limitato di righe

**LIMIT N**

N= numero massimo di righe

- AS – modifica il nome della Colonna (in visualizzazione)

**SELECT** <NomeAtt> **AS** <NuovoNome>

# CREAZIONE TABELLA TERAPIA

```
CREATE TABLE terapia (  
therapyID INTEGER PRIMARY KEY AUTOINCREMENT,  
patientID INTEGER,  
Nome TEXT,  
CasaProduttrice TEXT,  
costo DECIMAL(5,2),  
PrincipioAttivo TEXT,  
DataInizio DATE,  
DataFine DATE,  
FOREIGN KEY(patientID) REFERENCES paziente(patientID) ON DELETE CASCADE  
);
```

```
INSERT INTO terapia(patientID,Nome,CasaProduttrice,costo,PrincipioAttivo,DataInizio,DataFine)  
VALUES (3,'Cumadin','Bayer',45.00,'warfarin','2003-07-15',NULL);
```

# CREAZIONE TABELLA DIPENDENTI E COLLEGAMENTO A PAZIENTE

1

```
CREATE TABLE dipendenti(  
employeeID INTEGER PRIMARY KEY  
AUTOINCREMENT,  
Nome TEXT,  
Cognome TEXT,  
Sesso TEXT,  
matricola INTEGER,  
qualifica TEXT  
);
```

```
INSERT INTO dipendenti (Nome,Cognome,Sesso,matricola,qualifica) VALUES  
( 'Bruno','Mascheroni','M',12345,'dirigente medico');
```

```
INSERT INTO dipendenti (Nome,Cognome,Sesso,matricola,qualifica) VALUES  
( 'Anna','Trilli','F',61549,'dirigente medico');
```

```
INSERT INTO dipendenti (Nome,Cognome,Sesso,matricola,qualifica) VALUES  
( 'Vincenzo','Cinti','M',95346,'infermiere');
```

2

```
INSERT INTO dipendenti (Nome,Cognome,Sesso,matricola,qualifica) VALUES  
( 'MariaChiara','Bandai','F',87309,'tecnico radiologia');
```

```
INSERT INTO dipendenti (Nome,Cognome,Sesso,matricola,qualifica) VALUES  
( 'Andrea','Giannoni','F',25685,'infermiere');
```

3

```
ALTER TABLE paziente ADD  
COLUMN MedicoID INT; 3a-In SQL  
ALTER TABLE paziente ADD  
FOREIGN KEY (MedicoID) REFERENCES  
dipendenti(employeeID) ON DELETE SET NULL;
```

**Il comando ALTER TABLE non può essere usato in SQLite per aggiungere FOREIGN KEY (nè altri vincoli)**

**3b-In SQLite**

→ Rinominare la tabella in cui va aggiunto il vincolo – creare una nuova tabella che include il vincolo – copiare tutti i valori della tabella rinominata nella nuova tabella - aggiungere l'istanza dei vincoli

# CREAZIONE TABELLA DIPENDENTI E COLLEGAMENTO A PAZIENTE

```
ALTER TABLE paziente RENAME TO paziente_old;  
ALTER TABLE paziente_old ADD Medicoid INTEGER;
```

Rinomino la vecchia tabella e aggiungo la Colonna che ospiterà la foreign key

```
CREATE TABLE paziente  
(patientID INTEGER PRIMARY KEY AUTOINCREMENT,  
Nome VARCHAR (20),  
Cognome VARCHAR (20),  
Sesso CHAR,  
DataNascita DATE DEFAULT NULL,  
cf VARCHAR(16),  
Medicoid integer,  
FOREIGN KEY(Medicoid) REFERENCES dipendenti(employeeID) ON DELETE SET NULL);
```

Creo la nuova tabella che include la foreign key

```
INSERT INTO paziente SELECT* FROM paziente_old;
```

Copio I dati della vecchia tabella nella nuova

```
UPDATE paziente SET Medicoid = 3 WHERE patientID = 1 OR patientID = 5 OR patientID = 7;  
UPDATE paziente SET Medicoid = 1 WHERE patientID = 2 OR patientID = 4;
```

Aggiungo il valore della foreign key

# ON DELETE

**ON DELETE SET NULL** → quando la riga della tabella di riferimento (References) viene cancellata, la corrispondente FOREIGN KEY diventa NULL

```
DELETE FROM dipendenti WHERE employeeID=1;
```

**ON DELETE CASCADE** → quando la riga della tabella di riferimento (References) viene cancellata, la corrispondente riga che contiene la primary key cancellata come FOREIGN KEY viene cancellata

```
DELETE FROM paziente WHERE patientID=5;
```

# COSTRUTTI E OPERATORI

- SELECT DISTINCT – permette di visualizzare solo le tuple non duplicate
- COUNT (NomeAtt) – riporta come risultato il numero di tuple che soddisfano la condizione, contando i valori dell'attributo
- [SUM, AVG, MIN, MAX](NomeAtt) – riportano come risultato il valore dell'operazione algebrica di riferimento sulla Colonna indicata
- CONDIZIONE SU OPERATORI AGGREGATI – HAVING – permette di usare gli operatori aggregati nella clausola WHERE, dopo aver raggruppato appropriatamente mediante GROUP BY

# OPERATORE LIKE

- Definizione di pattern all'interno di stringhe:
  - % = qualsiasi numero di caratteri
  - \_ = un solo carattere
- Parola chiave nella clausola WHERE: LIKE al posto di =

```
SELECT *  
FROM terapia  
WHERE PrincipioAttivo LIKE '%o';
```

# Queries 1.1, 1.2 e 1.3

1.1 Conoscere cognome e nome dei pazienti maschi considerati nella base di dati;

1.2 Conoscere tutti i dati relativi ai pazienti il cui cognome è “Nunziatella”;

1.3 Conoscere cognome, nome, anno di nascita dei pazienti considerati, in ordine di cognome.

1.4 Conoscere cognome, nome e età dei pazienti

1.5 Conoscere principio attivo, ID paziente e data inizio delle terapie finite durante il 2019

# JOIN

## Sintassi SQL

```
SELECT *  
FROM paziente JOIN terapia ON paziente.patientID = terapia.patientID;
```

TABELLE DI  
CALCOLO

CONDIZIONE DI  
JOIN

## Sintassi SQLite alternative

```
SELECT *  
FROM paziente INNER JOIN terapia ON paziente.patientID = terapia.patientID;
```

```
SELECT *  
FROM paziente INNER JOIN terapia USING (patientID);
```

*Condizione di JOIN In caso le due colonne abbiano lo stesso nome*

# TIPI DI JOIN

- **inner** (join interno fra le due tabelle) corrisponde al theta-join del modello relazionale, dove la condizione viene espressa in `Condizione_di_join`;
- **left (outer)** (join esterno sinistro) viene valutato, con la condizione espressa in `Condizione_di_join`, il join interno sulle due tabelle, e tale risultato è arricchito con le righe della tabella di sinistra che non hanno righe nella tabella di destra (aggiunte righe a destra);
- **right (outer)** (join esterno destro) viene valutato, con la condizione espressa in `Condizione_di_join`, il join interno sulle due tabelle, e tale risultato è arricchito con le righe della tabella di destra che non hanno righe nella tabella di sinistra (aggiunte righe a sinistra);
- **full (outer)** (join esterno completo) viene valutato, con la condizione espressa in `Condizione_di_join`, il join interno sulle due tabelle, e tale risultato è arricchito con le righe di entrambe le tabelle che non hanno righe corrispondenti nell'altra (non supportato in Sqlite);

## Queries 2.4, 2.5 e 2.6

2.4 Conoscere tutte le terapie associate ad ogni paziente;

2.5 Conoscere cognome e nome dei pazienti trattati con warfarin;

2.6 Conoscere cognome, nome, data di nascita, terapia per quei pazienti la cui terapia è finita prima dell'anno 2015.

## Queries 2.10, 2.11, e 2.12

2.10 Conoscere cognome e nome dei pazienti che hanno terapie iniziate prima del 2015, ma senza duplicati;

2.11 Conoscere la media del costo di tutte le terapie associate ad ogni paziente in corso nel 2013;

2.12 Conoscere nome, cognome e numero di terapie associate ad ogni paziente;

2.12bis Conoscere nome, cognome e numero di terapie associate ad ogni paziente considerando solo i pazienti con più di una terapia

# OPERATORE UNION

- Operazione di unione insiemistica
- Vincoli:
  - Si possono unire due tabelle con lo STESSO NUMERO DI COLONNE
  - Le colonne selezionate devono AVERE LO STESSO DOMINIO

```
SELECT  
Nome,Cognome,Sesso  
FROM paziente
```

**UNION**

```
SELECT Nome, Cognome,  
Sesso  
FROM dipendenti;
```

# QUERY ANNIDATE

- È possibile usare il risultato di una query all'interno di un'altra query
- Il DBMS eseguirà prima la query più interna e poi quella più esterna
- Le query annidate vanno inserite tra parentesi

```
SELECT paziente.Nome, paziente.Cognome
FROM paziente
WHERE paziente.patientID IN
(
  SELECT terapia.patientID
  FROM terapia
  WHERE PrincipioAttivo LIKE '%lolo');
```

# ESEMPIO

- Ottenere il Codice Fiscale di quei pazienti che hanno speso più di 25 euro per la terapia.

STEP 1 – ottenere gli ID dei pazienti che hanno speso più di 25 euro

```
select patientID,sum(costo) as CostoTot from terapia group by  
patientID having CostoTot>25;
```

STEP 2 – visualizzare il CF a partire da questa selezione → uso  
l'operatore "IN"

# ESEMPIO

STEP 2a – seleziono solo il patientID a partire dalla query precedente

```
select patientID from  
(select patientID,sum(costo) as CostoTot from terapia group by  
patientID having CostoTot>25)as NewTer;
```

STEP 2b – seleziono il CF dei pazienti con quell ID

```
select cf  
from paziente  
where patientID in  
(select patientID from (select patientID,sum(costo) as CostoTot from terapia  
group by patientID having CostoTot>25)as NewTer);
```

## Query 2.13 e 2.14

2.13 - Conoscere il nome e il cognome dei medici associati ai pazienti che prendono carvedilolo.

2.14 - Conoscere il nome e il cognome dei medici associati ai pazienti che prendono carvedilolo solo se hanno almeno due terapie.