UNIVERSITA DEGLI STUDI DI FOGGIA

Dipartimento di Agraria

CdI in Ingegneria dei Sistemi Logistici per l'Agroalimentare

Corso integrato di Sistemi di Elaborazione

Modulo I

Prof. Crescenzio Gallo

crescenzio.gallo@unifg.it

Il linguaggio SQL

Linguaggio SQL

- Il **linguaggio SQL** (*Structured Query Language*) è un linguaggio dichiarativo (indica *cosa* vogliamo ottenere) e non procedurale (non *come* ottenerlo; a questo ci pensa il DBMS).
- È da tempo lo standard per le basi di dati relazionali. La versione più diffusa è l'**SQL2** (che noi utilizzeremo, nell'implementazione del DBMS MySql) adottata nel 1992.
- Una versione importante è l'**SQL3** che implementa nuove caratteristiche come la *ricorsività* e funzionalità *object-oriented*.

Linguaggio SQL

- Il **linguaggio SQL** è costituito dai seguenti sottoinsiemi:
  - **DDL** (*Data Definition Language*), che prevede le istruzioni per definire e **DCL** *(Data Control Language)* controllare la struttura della base di dati. Serve quindi a creare tabelle, vincoli, viste e indici e definire il controllo degli accessi e i permessi per gli utenti autorizzati;
  - **DML** (*Data Manipulation Language*), che prevede le istruzioni per manipolare (cioè inserire, modificare, cancellare) i dati contenuti nelle tabelle.

Identificatori e tipi di dati

- Le istruzioni possono essere scritte utilizzando indifferentemente caratteri maiuscoli o minuscoli, e sono generalmente separate con il "**;**" (punto e virgola).
- Gli identificatori utilizzati per i nomi di tabelle e attributi devono:
  - avere *lunghezza massima* pari a 18 caratteri;
  - *iniziare* con una lettera;
  - *avere* come unico carattere speciale l'underscore "_".
- Per riferirsi ad un attributo di una tabella si usa la notazione:

  <NomeTabella>.<NomeAttributo>

Identificatori e tipi di dati

- I **tipi di dato** utilizzabili per gli attributi (DBMS MySql) sono riassunti nella tabella seguente:

Tipo	Descrizione	Range di valori
CHAR(*n*)	Stringa a lunghezza fissa, di esattamente *n* caratteri	$0 \leq n \leq 255$ (con eventuale aggiunta di spazi)
VARCHAR(*n*)	Stringa a lunghezza variabile, di massimo *n* caratteri	$0 \leq n \leq 65535$
BOOLEAN	Valore logico "falso" (0) o "vero" ($\neq 0$)	{TRUE, FALSE}
TINYINT	Numero intero di 8 bit	[-128, 127] signed [0, 255] unsigned
SMALLINT	Numero intero di 16 bit	[-32768, 32767] signed [0, 65535] unsigned

Identificatori e tipi di dati

Tipo	Descrizione	Range di valori
INT	Numero intero di 32 bit	$[-2^{31}, 2^{31}-1]$ signed $[0, 2^{32}-1]$ unsigned
BIGINT	Numero intero di 64 bit	$[-2^{63}, 2^{63}-1]$ signed $[0, 2^{64}-1]$ unsigned
DECIMAL(m,d)	Numero con m cifre totali, di cui d dopo la virgola	Vengono aggiunti eventuali zeri dopo la virgola
FLOAT	Numero a virgola mobile di 32 bit	-3.4E38 ÷ 3.4E38 (precisione di circa 7 cifre)
DOUBLE	Numero a virgola mobile di 64 bit	-1.8E308 ÷ 1.8E308 (precisione di circa 15 cifre)
DATE	Data nel formato YYYY-MM-DD	1000-01-01 ÷ 9999-12-31

Identificatori e tipi di dati

- Le costanti stringa e le date sono delimitate con gli apici (‘ ’). Ad esempio:
  - ‘Dante Alighieri’
  - ‘2016-09-04’
- Nelle espressioni possono essere usati i seguenti operatori:
  - aritmetici {+, -, /, * }
  - relazionali {<, ≤, >, ≥, =, <>, BETWEEN…AND, IN(…)}
  - logici {AND, OR, NOT}

Caratteri jolly % e _

- Ai dati di tipo stringa è applicabile l'operatore **LIKE**, che consente di selezionare i valori stringa che soddisfano la condizione specificata.
- Ad es., se vogliamo i soli dipendenti il cui cognome inizia per *Ros* scriveremo la query:

```
SELECT * FROM Dipendenti WHERE Cognome LIKE 'Ros%';
```

- Il simbolo **%** sta per *nessuno, uno o più caratteri.*
- Se invece vogliamo i dipendenti il cui cognome inizia per *Ros* ed è seguito da un solo carattere (qualunque), scriveremo:

```
SELECT * FROM Dipendenti WHERE Cognome LIKE 'Ros_';
```

- Il simbolo _ sta per *un carattere qualunque*. Saranno quindi selezionati i cognomi *Rosi*, *Rosa* ma non *Rossi*.

Le istruzioni DDL

- Le istruzioni **DDL** dell'SQL creano o modificano lo schema di una base dati relazionale.
- Esaminiamo le più importanti.

CREATE TABLE <*NomeTabella*>
(<*Attributo*_1> <*Tipo*_1> [**NOT NULL**],
…
<*Attributo*_N> <*Tipo*_N> [**NOT NULL**] [,<*Vincolo*>]);

➡ NOT NULL indica che l'attributo è obbligatorio
➡ <*Vincolo*> può essere **PRIMARY KEY**, **UNIQUE**, **FOREIGN KEY**, **CHECK**.

Creare una tabella

- Data la tabella Dipendenti(CodDip, Cognome, Nome, DataAssunzione, Livello, Stipendio) un'istruzione SQL per la sua creazione potrebbe essere la seguente:

```
CREATE TABLE Dipendenti(CodDip CHAR(6) NOT NULL,
  COGNOME VARCHAR(30) NOT NULL, NOME VARCHAR(30) NOT NULL,
  DataAssunzione DATE, Livello TINYINT NOT NULL,
  Stipendio DECIMAL(8,3) NOT NULL,
  PRIMARY KEY(CodDip), CHECK(Stipendio>0),
  CHECK(Livello BETWEEN 0 AND 9));
```

Modificare la struttura di una tabella

Una volta creata la struttura di una tabella, la si può successivamente modificare con l'istruzione ALTER TABLE.

- **ALTER TABLE** <*NomeTabella*>
  ADD <*Attributo*_1> <*Tipo*_1> [**BEFORE** <*Attributo*_2>];
- **ALTER TABLE** <*NomeTabella*> **DROP COLUMN** <*Attributo*>;
- **ALTER TABLE** <*NomeTabella*> **MODIFY**(<*Attributo*> <*Tipo*>);

Le modifiche possibili alla struttura sono quindi l'*aggiunta* di una nuova colonna, la *rimozione* di una colonna o la *variazione* del tipo di dati di una colonna esistente nella tabella.

Eliminare una tabella

Per cancellare completamente una tabella dalla base di dati si utilizza l'istruzione DROP TABLE:

- **DROP TABLE** [**IF EXISTS**] <*NomeTabella*>;

L'opzione IF EXISTS evita di avere un errore nel caso la tabella da cancellare non esista.

Gli eventuali vincoli di integrità referenziale in cui è coinvolta la tabella potrebbero comunque impedirne la cancellazione (ad es. vincolo FOREIGN KEY … ON DELETE RESTRICT nella chiave esterna).

Vincoli di integrità

Abbiamo visto che si possono utilizzare i seguenti **vincoli interni** in sede di creazione di una tabella:

- **NOT NULL** per esprimere vincoli sugli attributi;
- **PRIMARY KEY** per esprimere vincoli sulle chiavi;
- **CHECK** per esprimere vincoli sulle righe (in MySql sono ignorati).

Per esprimere vincoli di integrità referenziale utilizziamo le clausole:

- **FOREIGN KEY** … **REFERENCES** per indicare le chiavi esterne;
- **RESTRICT**, **CASCADE** e **SET NULL** per implementare le relative politiche di gestione delle modifiche alla chiave primaria referenziata da una chiave esterna.

Vincoli di integrità

- La clausola RESTRICT impedisce la cancellazione di un valore della chiave primaria se esistono valori collegati di una chiave esterna.
- La clausola CASCADE aggiorna i valori di chiave esterna corrispondenti alla chiave primaria modificata.
- La clausola SET NULL imposta a NULL i valori di chiave esterna corrispondenti al valore di chiave primaria eliminato.

Ad esempio:

```
CREATE TABLE Azienda (CodAzienda CHAR(5) NOT NULL,
  RagioneSociale VARCHAR(30), CodAttività CHAR(4), CodDip CHAR(6) NOT NULL,
  PRIMARY KEY(CodAzienda),
  FOREIGN KEY(CodDip) REFERENCES Dipendenti(CodDip) ON DELETE CASCADE);
```

Le istruzioni DML

- Una volta creato lo schema relazionale tramite le istruzioni DDL, occorre poter riempire, modificare e cancellare i valori delle righe delle tabella.
- Vediamo le più importanti istruzioni **DML** (*Data Manipulation Language*) di SQL che effettuano tali operazioni.

Inserire righe in una tabella

- I valori delle righe possono essere inseriti utilizzando l'istruzione INSERT INTO la cui sintassi è:

  INSERT INTO <*NomeTabella*>[(<*Attributo1*>, …, <*AttributoN*>)]
  VALUES(<*Valore1*>, …, <*ValoreN*>);

- Se non è presente la lista degli attributi, si intende che i valori specificati devono corrispondere in ordine, tipo e numero a quelli specificati nella dichiarazione di <*NomeTabella*>.
- Se invece si specifica una lista di attributi, l'ordine e il tipo dei valori dovrà rispettare questa lista. Gli attributi omessi vengono assunti come NULL.

Ad esempio: `INSERT INTO Categoria VALUES('C001', 'Servizi');`

Modificare righe in una tabella

Per aggiornare una o più righe di una tabella utilizziamo l'istruzione UPDATE, la cui sintassi è:

UPDATE <*NomeTabella*>
SET <*Attributo*> = <*Espressione*> [, …]
[**WHERE** <*Condizione*>];

dove gli attributi specificati nella clausola SET vengono aggiornati con i valori delle corrispondenti espressioni in tutte le righe che soddisfano la condizione. Ad esempio:

- UPDATE Azienda
  SET RagioneSociale='New Electronics Srl'
  WHERE CodAzienda = 'A001'; ← una sola riga
- UPDATE Dipendenti
  SET Stipendio = Stipendio*1.05; ← tutte le righe della tabella

Cancellare righe in una tabella

Per cancellare una o più righe di una tabella utilizziamo l'istruzione DELETE, la cui sintassi è:

DELETE FROM <*NomeTabella*>
[**WHERE** <*Condizione*>];

In questo modo vengono eliminate tutte le righe che soddisfano la condizione. Ad esempio:

- DELETE FROM Azienda
  WHERE CodAzienda = 'A001'; ← una sola riga
- DELETE FROM Dipendenti; ← tutte le righe della tabella

Query Language: istruzione SELECT

- Per il reperimento dei dati (query), il linguaggio SQL prevede il comando SELECT, la cui potenza ed espressività sono alla base del successo dell'SQL.
- Il risultato di una query è sempre una tabella, che viene normalmente visualizzata, ma che può anche essere stampata oppure assegnata ad una variabile strutturata.
- La sintassi del comando SELECT è molto complessa. Iniziamo esaminando la sua forma semplificata.

Query Language: istruzione SELECT

SELECT [**DISTINCT**] <*Attributo*>, … | *
FROM <*Tabella*>, …
[**WHERE** <*Condizione*>];

- L'istruzione SELECT restituisce una tabella (avente gli attributi indicati; * = tutti) del prodotto cartesiano delle tabelle elencate limitatamente alle righe che soddisfano la <*Condizione*>.
- Se manca la clausola WHERE tutte le righe sono selezionate.
- Se è presente l'opzione DISTINCT, nella tabella risultato vengono rimosse le righe duplicate.

Query Language: istruzione SELECT

Esempi:

- SELECT Cognome, Nome FROM Dipendente;

  (visualizza il cognome e nome di tutti i dipendenti)

- SELECT * FROM Dipendente;

  (visualizza tutti gli attributi dei dipendenti)

- SELECT Cognome, Nome, Stipendio FROM Dipendenti
  WHERE Stipendio > 2000;

  (visualizza cognome, nome e stipendio dei dipendenti che guadagnano più di 2000 Euro)

Le operazioni relazionali in SQL

- Le operazioni di SELEZIONE (σ), PROIEZIONE (π) e JOIN (⋈) dell'algebra relazionale vengono realizzate in un database relazionale attraverso l'istruzione SELECT.
- Abbiamo già visto alcuni esempi di di proiezione (indicazione di alcuni attributi o * per tutti gli attributi) e selezione (clausola WHERE) con l'istruzione SELECT nelle diapositive precedenti: esaminiamone altri più in dettaglio.

L'operazione di selezione

- L'operazione di SELEZIONE (σ), che consente di ricavare da una tabella un'altra contenente solo le righe che soddisfano una certa condizione, viene realizzata in SQL mediante la clausola WHERE del comando SELECT.
- Ad esempio, l'operazione dell'algebra relazionale:

  $\sigma_{Stipendio>1000}(Dipendente)$

  viene tradotta in SQL nel seguente comando:

```
SELECT *
FROM Dipendente
WHERE Stipendio > 1000;
```

L'operazione di proiezione

- L'operazione di PROIEZIONE (π), che permette di ottenere solo alcuni attributi da una tabella, si realizza in SQL indicando accanto alla keyword SELECT l'elenco degli attributi richiesti.
- Ad esempio, l'operazione dell'algebra relazionale:

  $\pi_{Cognome,Nome,Livello}(Dipendente)$

  viene tradotta in SQL nel seguente comando:

```
SELECT Cognome, Nome, Livello
FROM Dipendente;
```

L'operazione di join

- Il comando SELECT può operare sul prodotto (cartesiano) di più tabelle, indicandone i nomi dopo la parola chiave FROM.
- Per praticità, ad una tabella si può associare un alias. Ad es.:
  SELECT * FROM Dipendente AS d;
  da utilizzare sia per qualificare gli attributi che nella WHERE.
- Indicando dopo WHERE i nomi degli attributi corrispondenti nelle due tabelle collegate (legati tra loro col segno "=") si realizza in pratica l'operazione di **join naturale** (chiamata anche **inner-join** o **equi-join**).

L'operazione di join

Ad esempio, l'operazione dell'algebra relazionale:

Dipendente ⋈ Azienda

può essere tradotta in SQL nel seguente comando:

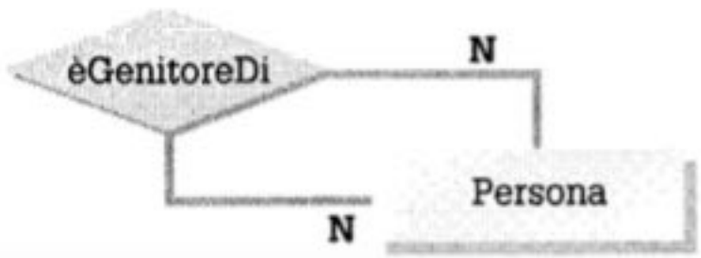
```
SELECT d.Cognome, d.Nome, d.Livello, a.RagioneSociale
FROM Dipendente AS d, Azienda AS a
WHERE d.CodAz = a.Id;
```

oppure, equivalentemente:

```
SELECT d.Cognome, d.Nome, d.Livello, a.RagioneSociale
FROM Dipendente AS d INNER JOIN Azienda AS a ON d.CodAz = a.Id;
```

L'operazione di join

- Consideriamo ora il seguente esempio, in cui nel diagramma E/R abbiamo l'associazione *èGenitoreDi* sulla stessa entità *Persona*.



Persona(<u>CodPers</u>, Cognome, Nome)

ÈGenitoreDi(<u>CodPers1</u>, <u>CodPers2</u>)

- Se in SQL vogliamo avere una tabella risultato con *il cognome e il nome delle persone accanto al cognome e nome dei genitori*, dobbiamo scrivere:

```
SELECT t1.Cognome, t1.Nome, t2.Cognome, t2.Nome
FROM èGenitoreDi AS r
    INNER JOIN Persona AS t1 ON r.CodPers1=t1.CodPers
    INNER JOIN Persona AS t2 ON r.CodPers2=t2.CodPers;
```

L'operazione di unione

- Per tradurre l'operazione di unione dell'algebra relazionale il linguaggio SQL utilizza l'operatore UNION.
- Ad esempio, consideriamo le due tabelle (omogenee):

  Regista(CodRegista, Cognome, Nome)
  Attore(CodAttore, Cognome, Nome)

- Per ottenere *tutti i registi e tutti gli attori* scriveremo:

```
(SELECT Cognome, Nome FROM Regista)
UNION
(SELECT Cognome, Nome FROM Attore);
```

L'operazione di intersezione

- L'operazione di intersezione restituisce le righe comuni di due tabelle.
- Sebbene non vi sia una vera e propria istruzione INTERSECT in MySql, è facilmente simulabile utilizzando le clausole IN, EXISTS o JOIN a seconda della complessità della query.
- Per ottenere ad es. *tutti i registi che sono stati anche attori*:

```
SELECT Cognome, Nome FROM Regista
WHERE CONCAT(Cognome, Nome) IN
        (SELECT CONCAT(Cognome, Nome) FROM Attore);
```

oppure

```
SELECT Cognome, Nome FROM Regista AS r INNER JOIN Attore AS a
    ON r.Cognome=a.Cognome AND r.Nome=a.Nome;
```

L'operazione di intersezione

- Come esempio di utilizzo della clausola EXISTS si considerino le tabelle: Rubrica(Id, Cognome, Nome) e Clienti(Id, Cognome, Nome).
- Vogliamo estrarre *i contatti della rubrica con Id < 100 che siano anche clienti non aventi però il cognome* Rossi :

```
SELECT r.Id, r.Cognome, r.Nome FROM Rubrica AS r
WHERE r.Id<100 AND EXISTS
(SELECT * FROM Clienti AS c
WHERE c.Cognome <> 'Rossi'
      AND c.Id = r.Id
      AND c.Cognome = r.Cognome
      AND c.Nome = r.Nome);
```

L'operazione di differenza

- L'operazione di differenza restituisce le righe della prima tabella che non sono presenti nella seconda.
- Sebbene non vi sia una vera e propria istruzione in MySql, la differenza è facilmente simulabile utilizzando le clausole NOT IN o NOT EXISTS a seconda della query.
- Per ottenere ad es. *tutti i registi che non sono mai stati attori*:

```
SELECT Cognome, Nome FROM Regista
WHERE CONCAT(Cognome, Nome) NOT IN
        (SELECT CONCAT(Cognome, Nome) FROM Attore);
```

oppure

```
SELECT r.Cognome, r.Nome FROM Regista AS r WHERE NOT EXISTS
    (SELECT * FROM Attore AS a
        WHERE r.Cognome=a.Cognome AND r.Nome=a.Nome);
```

Le funzioni di aggregazione

- SQL possiede alcune funzioni predefinite, utilissime in molte circostanze in cui occorre effettuare conteggi, somme, calcoli di medie o altro ancora.
- Tali funzioni si applicano a una colonna di una tabella; la loro sintassi è la seguente:

  <*Funzione di aggregazione*>([**DISTINCT**] <*Attributo*>)

  dove <*Funzione di aggregazione*> può essere:

Le funzioni di aggregazione

- **COUNT**, che conteggia il *numero di elementi* della colonna <*Attributo*>
- **MIN**, che restituisce il *valore minimo* della colonna <*Attributo*>
- **MAX**, che restituisce il *valore massimo* della colonna <*Attributo*>
- **SUM**, che restituisce la *somma degli elementi* della colonna <*Attributo*>
- **AVG**, che restituisce la *media aritmetica* della colonna <*Attributo*>

Le funzioni di aggregazione

- Consideriamo ad es. la tabella:
  Dipendente(CodDip, Cognome, Nome, Livello, DataStipendio, Stipendio).
- Per conoscere *il numero di dipendenti con stipendio maggiore di 2000 Euro* scriveremo:

```
SELECT COUNT(Stipendio)
FROM Dipendente WHERE Stipendio > 2000;
```

- Per conoscere *l'esborso totale per gli stipendi dei dipendenti* scriveremo:

```
SELECT SUM(Stipendio) FROM Dipendente;
```

- Per conoscere *lo stipendio medio dei dipendenti* scriveremo:

```
SELECT AVG(Stipendio) FROM Dipendente;
```

- Per conoscere *quanti livelli di dipendenti* esistono scriveremo:

```
SELECT COUNT(DISTINCT Livello) FROM Dipendente;
```

Ordinamenti

- Non esiste alcuna assunzione sull'ordine in cui possono apparire le righe di una *tabella risultato* di una query.
- In SQL è possibile ordinare tali righe utilizzando alcune clausole, che vediamo di seguito, dell'istruzione SELECT:

  ORDER BY <*Attributo1*> [**ASC**|**DESC**], …, <*AttributoN*> [**ASC**|**DESC**]

  dove ASC e DESC stanno per ordine *crescente* (quello default) e *decrescente*.
- L'ordinamento viene eseguito prima su <*Attributo1*> poi, a parità di ordinamento su questo, si ordina in base ad <*Attributo2*> e così via.

Ordinamenti

Esempi.

- Per ordinare i Dipendenti in *ordine alfabetico* scriveremo:

```
SELECT * FROM Dipendente ORDER BY Cognome, Nome;
```

- Per ordinare i Dipendenti con stipendi maggiori di 3000 Euro in ordine decrescente (cioè dallo stipendio più alto a quello più basso) scriveremo:

```
SELECT Cognome, Nome, Stipendio
FROM Dipendente
WHERE Stipendio > 3000
ORDER BY STIPENDIO DESC;
```

Raggruppamenti

- Le funzioni di aggregazione sono generalmente abbinate alle *clausole di raggruppamento*, la cui sintassi è:

  GROUP BY <*Attributo1*>, …,<*AttributoN*> [**HAVING** <*Condizione*>]

- Con questa clausola il significato della SELECT è il seguente:
  - viene eseguito il prodotto delle tabelle elencate dopo FROM;
  - su tale prodotto si effettua la selezione indicata dalla clausola WHERE;
  - la *tabella risultante* viene logicamente partizionata in gruppi di righe. Due righe appartengono allo stesso gruppo se hanno gli stessi valori per gli attributi elencati nella clausola GROUP BY;
  - tutti i gruppi che *non soddisfano* la clausola HAVING vengono scartati.

Raggruppamenti

- Ad es., *per raggruppare i dipendenti in base al loro livello e conoscere lo stipendio medio per livello* possiamo scrivere:

```
SELECT Livello, AVG(Stipendio) FROM Dipendente GROUP BY Livello;
```

Il risultato potrebbe essere:

Livello	AVG(Stipendio)
5	1600.00
6	1700.00
7	2000.00

Raggruppamenti

- *Per raggrupparli per livello e in più conoscere stipendio medio e numero di dipendenti per livello* scriveremo:

```
SELECT Livello, AVG(Stipendio) AS 'Stipendio medio',
        COUNT(Livello) AS 'N.ro dipendenti'
FROM Dipendente GROUP BY Livello;
```

Il risultato potrebbe essere:

Livello	Stipendio medio	N.ro dipendenti
5	1600.00	10
6	1700.00	8
7	2000.00	5

Raggruppamenti

- *Per raggrupparli in livelli solo per quelli maggiori del sesto* scriveremo:

```
SELECT Livello, AVG(Stipendio) AS 'Stipendio medio',
      COUNT(Livello) AS 'N.ro dipendenti'
FROM Dipendente GROUP BY Livello HAVING LIVELLO>6;
```

Il risultato sarà:

Livello	Stipendio medio	N.ro dipendenti
7	2000.00	5

Query nidificate e subquery

- Per rispondere a query complesse è possibile strutturare più SELECT. Ciò consente di costruire un'interrogazione al cui interno sono presenti altre interrogazioni, dette **subquery**.
- Consideriamo ad es. le seguenti tabelle relative all'utilizzo di laboratori da parte di una classe di studenti:

Laboratorio(<u>CodLab</u>, NumPosti, NomeLab)
Classe(<u>CodClasse</u>, NumPosti)
Utilizza(<u>CodLab</u>, <u>CodClasse</u>)

Query nidificate e subquery

- Se vogliamo conoscere il nome dei laboratori utilizzati dalla classe "A45" scriveremo:

```
SELECT a.NomeLab FROM Laboratorio AS a INNER JOIN
    (SELECT CodLab FROM Utilizza WHERE CodClasse='A45') AS b
    USING(CodLab);
```

- La SELECT interna è una **subquery**.
- L'espressione USING(CodLab) è un'abbreviazione che in MySql equivale a ON a.CodLab=b.CodLab e si può utilizzare solo quando le due colonne hanno lo stesso nome.

Conservazione dei risultati

- In un processo di query nidificate, spesso è utile conservare le tabelle risultato di alcune subquery.
- Per fare questo, occorre creare una *tabella temporanea* che conservi il risultato della subquery.
- Una tabella temporanea viene automaticamente cancellata alla fine della sessione di lavoro, senza la necessità di utilizzare il comando DROP TABLE.

Conservazione dei risultati

- Riprendendo l'esempio precedente, creiamo una tabella temporanea per memorizzare il risultato della subquery:

  ```
  CREATE TEMPORARY TABLE Temp1(CodLab VARCHAR(100));
  ```

  che conterrà il risultato della subquery relativa alla selezione dei codici dei laboratori della classe "A45".

- La precedente subquery diventa ora:

  ```
  INSERT INTO Temp1
      SELECT CodLab FROM UTILIZZA WHERE CodClasse = 'A45';
  ```

- E infine la query nidificata viene così riscritta:

  ```
  SELECT t.CodLab, t.NumPosti
  FROM Temp1 INNER JOIN Laboratorio AS t USING(CodLab);
  ```

Conservazione dei risultati

- Sia data la seguente tabella, che si riferisce ai dipendenti di un'azienda:

  Dipendente(CodDip, Cognome, Nome, DataStipendio, Stipendio)

- Se si vuole creare una nuova tabella *Settembre*, relativa agli stipendi del mese di settembre dei dipendenti di quella azienda, scriveremo:

```
CREATE TABLE Settembre LIKE Dipendente;
INSERT INTO Settembre
    SELECT * FROM Dipendente
    WHERE DataStipendio BETWEEN '2016-09-01' AND '2016-09-30';
```

- La clausola WHERE può anche essere scritta così:

```
WHERE Year(DataStipendio)=2016 AND Month(DataStipendio)=9;
```

Subquery con valore singolo

- Quando si è sicuri che una subquery produca un singolo valore come risultato (cioè una tabella formata da una sola riga e da una sola colonna), è possibile utilizzare tale subquery nelle espressioni delle query.
- Consideriamo le seguenti tabelle:

  Regista(CodRegista, Cognome, Nome, CompensoMinimo)

  Film(CodFilm, Regista, Titolo, Budget)
- Se vogliamo visualizzare il regista del film *Ghost*, scriveremo:

```
SELECT Cognome, Nome FROM Regista
WHERE CodRegista = (SELECT Regista FROM Film WHERE Titolo = 'Ghost');
```

Subquery con righe multiple

- Nelle subquery è possibile utilizzare i seguenti predicati (già visti) che semplificano la scrittura di query complesse:
  - <*Espressione*> **IN** | **NOT IN** <*Insieme*>
    (la condizione è vera se il valore di <*Espressione*> appartiene a <*Insieme*>)
  - **EXISTS** | **NOT EXISTS** <*Query*>
    (la condizione è vera se la <*Query*> restituisce almeno una riga)
- Consideriamo le seguenti tabelle:

  Automobile(CodAuto, Marca, Modello, Targa, Prezzo)

  DotataDi(CodAuto, CodAcc)

  Accessorio(CodAcc, Descrizione, PrezzoAcquisto, PrezzoVendita, Quantità)

Subquery con righe multiple

- Per rispondere alla query "*Quali sono le automobili sui cui accessori si ha un ricarico superiore a 100 Euro?*" scriveremo:

```
SELECT CodAuto FROM DotataDi WHERE CodAcc IN
    (SELECT CodAcc FROM Accessorio
     WHERE PrezzoVendita-PrezzoAcquisto > 100);
```

- Per rispondere alla query "*Quali sono le automobili per le quali esiste almeno un accessorio con un prezzo maggiore di 3000 Euro?*" scriveremo:

```
SELECT CodAuto FROM DotataDi WHERE EXISTS
(SELECT CodAcc FROM Accessorio AS a INNER JOIN DotataDi AS d ON(CodAcc)
 WHERE a.PrezzoVendita>3000);
```

Le viste (view)

- Le tabelle definite mediante l'istruzione CREATE TABLE sono presenti fisicamente nella base di dati.
- In SQL è possibile definire un'altra classe di tabelle "virtuali" chiamate **viste**, che non sono fisicamente memorizzate nella base di dati.
- Una vista è ottenuta tramite una query su tabelle fisiche e/o altre viste; è sempre possibile interrogare una vista e — sotto certe condizioni — anche modificarla (INSERT/UPDATE/DELETE).
- Per creare una vista si utilizza la seguente sintassi:

CREATE VIEW <*NomeVista*> **AS** <*Query*>;

Le viste (view): esempio

- Consideriamo le seguenti tabelle:

  Automobile(CodAuto, Marca, Modello, Targa, Prezzo, CodProp)

  Proprietario(CodFiscale, Cognome, Nome)

  Accessorio(CodAcc, Descrizione, PrezzoAcquisto, PrezzoVendita, Quantità)

- Una vista molto utile potrebbe essere quella che consente di visualizzare gli accessori che devono essere ordinati, poiché sono terminati in magazzino:

```
CREATE VIEW DaOrdinare AS
SELECT * FROM Accessorio WHERE Quantità=0;
```

Le viste (view): esempio

La tabella virtuale *DaOrdinare* è definita logicamente; essa è fisicamente rappresentata dalla parte della tabella *Accessorio* per cui *Quantità* è pari a 0.

Accessorio	CodAcc	Descrizione	PrezzoAcquisto	PrezzoVendita	Quantità
	A01	Batteria	100,00	120,0	3
	A02	Tergicristalli	80,00	100,00	0
	A03	Specchietto Dx	50,00	75,00	1
	A04	Specchietto Sin	50,00	75,00	0

DaOrdinare	CodAcc	Descrizione	PrezzoAcquisto	PrezzoVendita	Quantità
	A02	Tergicristalli	80,00	100,0	0
	A04	Specchietto Sin	50,00	75,00	0

Il resto della tabella è "oscurato": questo consente di realizzare *sottoschemi logici* dallo schema concettuale globale (per comodità o sicurezza espositiva).

Le viste (view): considerazioni

- Il motivo per cui si creano le viste è quindi quello di fornire a un gruppo di utenti una versione semplificata o parziale di una realtà che può essere anche molto complessa.
- Categorie diverse di utenti possono interagire con la base di dati utilizzando il loro punto di vista e trascurandone altri. Ad es., una view che esponga i dati precedenti per il venditore sarà:

```
CREATE VIEW Vendita AS
SELECT CodAcc, Descrizione, PrezzoVendita, Quantità FROM Accessorio;
```

- Ogni modifica apportata apportata sulla viste *DaOrdinare* e *Vendita* si ripercuote sulla tabella fisica *Accessorio*.

Le viste (view): considerazioni

- N.B. *Le viste che coinvolgono operazioni (complesse) di join e/o che non espongono la chiave primaria potrebbero non consentire la modifica delle tabelle fisiche sottostanti.*
- Le viste sono quindi un vero e proprio **strumento di protezione dei dati**; ad esse è possibile ad es. applicare l'istruzione GRANT:

  GRANT SELECT ON Vendita TO Utente1;

  che in questo caso concede i diritti di accesso in lettura a *Utente1* sulla vista *Vendita* (REVOKE toglie invece i diritti).
- Per eliminare una vista si utilizza l'istruzione:

  DROP VIEW <*NoteVista*>;

Quesiti riassuntivi

1. Che tipo di linguaggio è SQL?
2. A cosa corrispondono le parti DDL e DML dell'SQL?
3. Come è possibile in SQL specificare un vincolo di chiave primaria?
4. Come è possibile in SQL indicare colonne che fanno parte di una chiave candidata?
5. Come si specifica un vincolo di integrità referenziale in SQL?
6. Come si modifica la struttura di una tabella per:
   - aggiungere una nuova colonna?
   - cancellare una colonna?
   - modificare il tipo di una colonna?
   - modificare il nome di una colonna?