ESERCIZI LABORATORIO SQL

- Creazione tabelle

Creazione delle tabelle Impiegati e Dipartimenti.

```
CREATE TABLE Impiegati(
ID              smallint primary key,
Nome            char(20) not null,
Cognome         char(30) not null,
Residenza       char(20) default '*** Manca Residenza',
Stipendio       decimal(9,2),
Dipartimento    char(5) references Dipartimenti(Codice),
UNIQUE(Cognome, Nome, Dipartimento)
);
```

ID è chiave primaria della tabella.

Campi obbligatori: non sono ammessi valori nulli in *Nome* e *Cognome*.

Il valore da attribuire a *Residenza* in caso di valore nullo.

Ogni comando SQL termina con il punto e virgola.

Vieta la presenza di valori duplicati in una colonna o in un insieme di colonne: non ci possono essere due dipendenti con identico nome e cognome nello stesso dipartimento.

Dipartimento è chiave esterna associata a *Codice*. Crea un vincolo di *integrità referenziale* con *Dipartimenti*.

```
CREATE TABLE Dipartimenti(
Codice      char(5),
Descrizione char(20) not null,
Sede        char(20),
Manager     smallint,
Primary Key (Codice),
Unique (Descrizione),
Foreign Key (Manager) references Impiegati(ID)
   On Delete set null
   On Update cascade
);
```

Codice è chiave primaria della tabella.

Nella colonna *Descrizione* non ci possono essere valori duplicati. *Descrizione* identifica univocamente una riga della tabella.

Manager è chiave esterna associata al campo *ID* della tabella *Impiegati*.

L'aggiornamento di un *ID* associato a *Manager* si riflette a catena sui valori di *Manager*.

La cancellazione di una riga di *Impiegati* implica che i valori di *Manager* associati all'*ID* di quella riga assumano valore nullo.

- Inserimento dati in tabella

```
INSERT INTO Impiegati
(ID, Nome, Cognome, Residenza, Stipendio, Dipartimento)
VALUES(20, 'Mario', 'Rossini', 'Caserta', 31500, 'Mag');
```

Inserimento di una riga nella tabella *Impiegati*.

- Modifiche tabella

```
ALTER TABLE Impiegati
ADD Nascita date;

ALTER TABLE Impiegati
DROP Residenza;
```

Inserisce in *Impiegati* una nuova colonna di nome *Nascita* per registrare la data di nascita del dipendente.

Elimina da *Impiegati* la colonna di nome *Residenza*: il comando elimina anche i dati contenuti nella colonna.

- Modifiche campi tabella

```
UPDATE Impiegati
SET Dipartimento = 'Prod'
WHERE ID = 20;
```

Assegnamento del dipendente con *ID=20* al dipartimento *Produzione*.

```
DELETE FROM Impiegati
WHERE ID = 20;
```

Cancellazione del dipendente con *ID=20* dalla tabella *Impiegati*.

```
UPDATE Impiegati
SET Stipendio = Stipendio * 1.05
WHERE Dipartimento = 'Prod';
```

Aumenta del 5% lo stipendio ai dipendenti del dipartimento *Produzione*.

- Cancellazioni campi tabella

```
DELETE FROM Impiegati
WHERE Dipartimento = 'R&S';
```

Cancella i dipendenti del dipartimento *Ricerca & Sviluppo*.

```
DELETE FROM Impiegati;
```

Manca la clausola *WHERE*: il comando svuota la tabella *Impiegati* dai dati, ma rimane la sua struttura.

- Cancellazione struttura tabella

```
DROP TABLE Impiegati;
```

Elimina struttura e dati della tabella *Impiegati*.

- Selezione dei dati in tabella

```
SELECT Elenco di espressioni
FROM Tabelle
WHERE Condizioni ;
```

Un elenco di espressioni, che coinvolgono le colonne, da mostrare. * per indicare tutte le colonne.

La tabella, o le tabelle, usate nell'interrogazione.

Una espressione logica che specifica quali righe considerare. Espressione booleana ottenuta componendo confronti con gli operatori AND OR e NOT.

Cognome, nome e residenza dei dipendenti del dipartimento di codice *Prod*

```
SELECT Cognome, Nome, Residenza
FROM Impiegati
WHERE Dipartimento = 'Prod';
```

Query1

Cognome	Nome	Residenza
Rossi	Mario	Torino
Colombi	Margherita	Roma
Magenta	Fabrizio	Torino

Record: 1 di 3

```
SELECT ID, Cognome, Nome
FROM Impiegati
WHERE Dipartimento = 'Prod' AND Residenza = 'Torino';
```

ID, *Cognome*, *Nome* dei dipendenti che lavorano alla produzione e risiedono a *Torino*.

```
SELECT ID AS Matricola, Nome, Cognome
FROM Impiegati;
```

Elenco degli impiegati con *ID*, *Nome* e *Cognome*, ma usando *Matricola* come intestazione della colonna *ID*.

```
SELECT Cognome, Nome, Residenza, Stipendio
FROM Impiegati
WHERE Stipendio >= 55000;
```

Elenca i dipendenti con retribuzione di almeno 55000 euro

```
SELECT *
FROM Impiegati, Dipartimenti
WHERE Dipartimento = Codice;
```

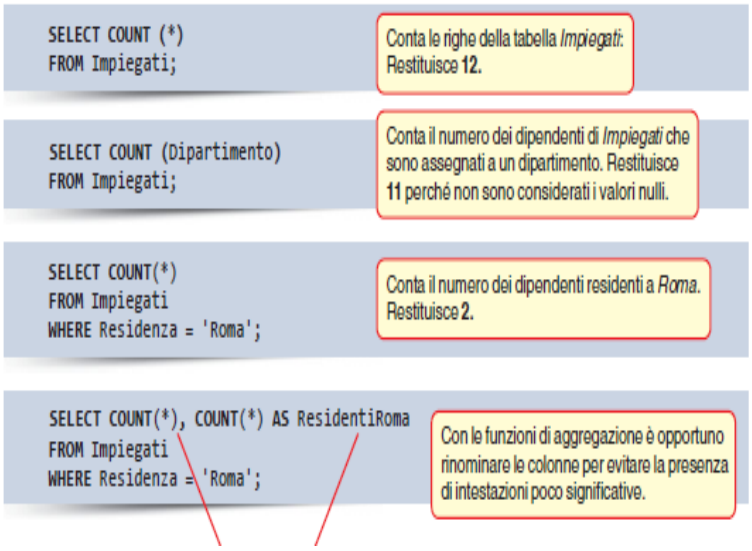
L'elenco di tutti i dipendenti con i dati del dipartimento dove lavorano, si ottiene con la congiunzione delle tabelle *Impiegati* e *Dipartimenti* secondo gli attributi comuni *Dipartimento* e *Codice*.

Esempio dell'utilizzo di AS per ridefinire il nome del campo nel risultato della query

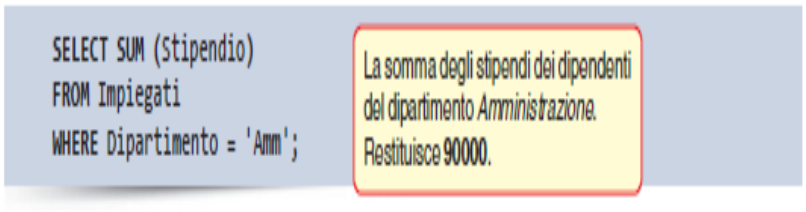
```
SELECT ID AS Matricola, Cognome, Nome
FROM Impiegati
WHERE Dipartimento IS NULL;
```

Per elencare i dipendenti senza dipartimento di assegnazione, si devono riconoscere le righe con *valori nulli* nel campo *Dipartimento*.

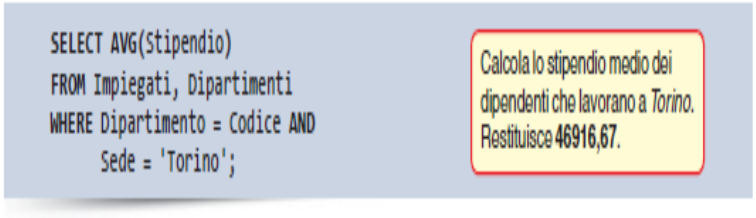
- Operatori di aggregazione COUNT



- Operatori di aggregazione SUM



- Operatori di aggregazione AVG (Media)



- Selezione MIN E MAX

```
SELECT MIN(Stipendio),
       MAX(Stipendio)
FROM Impiegati;
```

```
SELECT MIN(Cognome),
       MAX(Cognome)
FROM Impiegati;
```

- Ordinamento dei risultati

```
SELECT Cognome, Nome, Residenza
FROM Impiegati
ORDER BY Cognome, Nome;
```

```
SELECT Cognome, Stipendio
FROM Impiegati
ORDER BY Stipendio DESC, Cognome;
```

- LE CONDIZIONI DI RICERCA

Le **condizioni di ricerca** sono utilizzate nella clausole *Where* per determinare i criteri di selezione rispettivamente delle righe e dei raggruppamenti. Nella scrittura delle condizioni si usano i segni del confronto =, <, >, <>, >=, <=.

Una condizione di ricerca è costruita anche mettendo insieme più condizioni legate tra loro con gli operatori **AND** e **OR**, precedute eventualmente dall'operatore **NOT**. L'ordine di applicazione degli operatori è il seguente: **NOT** viene applicato prima di **AND** e **AND** prima di **OR**.
Le condizioni di ricerca si possono esprimere utilizzando anche altri operatori o predicati che rendono più potenti ed espressive le interrogazioni alla base di dati: **BETWEEN**, **IN**, **IS NULL**, **LIKE**.

- **BETWEEN** controlla se un valore è compreso in un intervallo di valori, inclusi gli estremi. È possibile specificare, anteponendolo a *Between*, anche l'operatore logico **NOT** per valutare la condizione opposta, cioè per controllare se il valore *non* rientra nell'intervallo specificato.
- **IN** controlla se un valore appartiene a un insieme di valori che viene precisato, tra parentesi, di seguito a *In*. I valori dell'elenco sono separati da una virgola. Anche **IN** può essere preceduto da **NOT** per indicare la condizione opposta, cioè la non appartenenza del valore all'insieme dei valori.
- **IS NULL** è un predicato che serve a controllare la presenza di un valore nullo in una colonna. È importante osservare che l'uso di questo predicato è il solo modo per controllare la presenza di un valore nullo. È possibile inserire l'operatore di negazione **NOT** per controllare che un dato non abbia valore nullo: il controllo si esegue con il predicato **IS NOT NULL**. L'operatore *Is* viene utilizzato solo con la parola *Null*.

Esempi:

```
SELECT Cognome, Nome, Residenza
FROM Impiegati
WHERE Stipendio BETWEEN 30000 AND 45000;

WHERE Stipendio>=30000 AND Stipendio<=45000
```

Cognome, nome e residenza dei dipendenti con retribuzione compresa tra *30000* e *45000*.

Condizione equivalente.

```
SELECT *
FROM Impiegati
WHERE Residenza IN ('Torino','Venezia','Palermo');

WHERE Residenza ='Torino' OR Residenza ='Venezia'
      OR Residenza ='Palermo'
```

I dati dei dipendenti che risiedono a: *Torino* o *Venezia* o *Palermo*.

Condizione equivalente.

```
SELECT Cognome, Nome
FROM Impiegati
WHERE Stipendio IS NOT NULL;
```

Cognome e nome dei dipendenti dei quali è noto lo stipendio.

- CONDIZIONI DI RICERCA (LIKE)

```
SELECT Cognome, Dipartimento
FROM Impiegati
WHERE Cognome LIKE 'R%';
```

Il precedente esempio estrae le colonne Cognome e Dipartimento selezionando dal Campo cognome solo i valori che iniziano per R.

LIKE confronta il valore di un attributo di tipo carattere con un modello di stringa che può contenere **caratteri jolly** (o *metacaratteri*). Anche in questo caso si può usare l'operatore **NOT** prima di *Like* per indicare criteri di ricerca opposti. Per l'uso dell'operatore **LIKE** nelle interrogazioni SQL di Access valgono le convenzioni descritte nel capitolo precedente.

I caratteri jolly sono:
_ (*underline* o *underscore*) per indicare uno e un solo carattere qualsiasi in quella posizione della stringa, Access, come altre implementazioni di SQL, fa uso di ? al posto di _;
% (*percento*) per indicare una sequenza di zero o più caratteri in quella posizione della stringa. Access, come altre implementazioni di SQL, fa uso di * al posto di %;
Esempi
LIKE 'xyz%' Ricerca le stringhe che *iniziano* con 'xyz' e anche i soli caratteri 'xyz';
LIKE '%xyz' Ricerca le stringhe che *terminano* con 'xyz' e anche i soli caratteri 'xyz';
LIKE '%xyz%' Ricerca le stringhe che *contengono* 'xyz' e anche i soli caratteri 'xyz';
LIKE '_xyz' Ricerca le stringhe di 4 caratteri che finiscono con 'xyz'.
Nome LIKE 'xyz' Equivale al confronto: Nome = 'xyz

Per ricercare uno dei caratteri jolly in una stringa si deve usare uno speciale carattere, detto **carattere escape** per qualificare un carattere come carattere da ricercare e non come metacarattere.
Il carattere *escape* viene precisato con la dichiarazione **ESCAPE**.
Per esempio, per ricercare la presenza del carattere _ nel nome di un dipartimento, si usa la seguente condizione:
LIKE '%$_%' **ESCAPE** '$'.
Il carattere '$' è il carattere *escape* e la sequenza '$_' precisa che il carattere _ (*underline*) è il carattere da ricercare nella stringa e non un carattere jolly.

- **Riepilogo dei comandi**

Linguaggio SQL

Comando / Clausola	Sintassi
ALTER TABLE (aggiunta colonna)	ALTER TABLE *NomeTabella* ADD *NomeColonna TipoDato*
ALTER TABLE (elimina colonna)	ALTER TABLE *NomeTabella* DROP COLUMN *NomeColonna*
AS (alias per colonna)	SELECT *NomeColonna* AS *NuovoNome* FROM *NomeTabella*
AS (alias per tabella)	SELECT *NomeColonna* FROM *NomeTabella* AS *NuovoNome*
BETWEEN	SELECT *NomeColonna1, NomeColonna2, ...* FROM *NomeTabella* WHERE *NomeColonna* BETWEEN *valore1* AND *valore2*
CREATE DATABASE	CREATE DATABASE *NomeDatabase*
CREATE INDEX	CREATE INDEX *NomeIndice* ON *NomeTabella* (*NomeColonna*)
CREATE TABLE	CREATE TABLE *NomeTabella* (*NomeColonna1 TipoDato,* *NomeColonna2 TipoDato,* .....)
CREATE UNIQUE INDEX	CREATE UNIQUE INDEX *NomeIndice* ON *NomeTabella* (*NomeColonna*)
CREATE VIEW	CREATE VIEW NomeVista AS SELECT *NomeColonna1, NomeColonna2, ...* FROM *NomeTabella* WHERE *condizione*
DELETE FROM	DELETE FROM *NomeTabella* (cancella l'intera tabella) DELETE FROM *NomeTabella* WHERE *condizione*
DROP DATABASE	DROP DATABASE *NomeDatabase*
DROP INDEX	DROP INDEX *NomeTabella.NomeIndice*
DROP TABLE	DROP TABLE *NomeTabella*

Comando / Clausola	Sintassi
GROUP BY	SELECT *NomeColonna1*,SUM(*NomeColonna2*) FROM *NomeTabella* GROUP BY *NomeColonna1*
HAVING	SELECT *NomeColonna1*,SUM(*NomeColonna2*) FROM *NomeTabella* GROUP BY *NomeColonna1* HAVING SUM(*NomeColonna2*) *condizione*
IN	SELECT *NomeColonna1, NomeColonna2, ...* FROM *NomeTabella* WHERE *NomeColonna* IN (*valore1,valore2,..*)
INSERT INTO	INSERT INTO *NomeTabella* VALUES (*valore1,valore2,..*) INSERT INTO *NomeTabella* (*NomeColonna1, NomeColonna2,...*) VALUES (*valore1,valore2,..*)
LIKE	SELECT *NomeColonna1, NomeColonna2, ...* FROM *NomeTabella* WHERE *NomeColonna* LIKE *ValoreCampione*
ORDER BY	SELECT *NomeColonna1, NomeColonna2, ...* FROM *NomeTabella* ORDER BY *NomeColonna* [ASC | DESC]
SELECT	SELECT *NomeColonna1, NomeColonna2, ...* FROM NomeTabella
SELECT *	SELECT* FROM *NomeTabella*
SELECT DISTINCT	SELECT DISTINCT *NomeColonna1, NomeColonna2, ...* FROM *NomeTabella*
SELECT INTO	SELECT *NomeColonna1, NomeColonna2, ...* INTO *NuovaTabella* FROM *NomeTabella*
UPDATE	UPDATE *NomeTabella* SET *NomeColonna =NuovoValore* WHERE *NomeColonna = Valore*
WHERE	SELECT *NomeColonna1, NomeColonna2, ...* FROM *NomeTabella* WHERE *condizione*

Operatori per l'impostazione dei criteri nelle query	
=	uguale a
<	minore di
<=	minore o uguale a
>	maggiore di
>=	maggiore o uguale a
<>	diverso da
Like	confronto con una sequenza di caratteri
And	per combinare due condizioni che devono essere entrambe verificate
Or	almeno una delle due condizioni combinate deve essere verificata
Not	negazione della condizione
Between...And	intervallo di valori
In	valore compreso all'interno di un insieme di valori specificato
Is Null	valore nullo

- Variabili utilizzabili

BOOLEAN	Valore logico	True, False
CHARACTER(n)	Stringa di lunghezza n	n da 1 a 15000
DATE	Data nella forma MM/GG/AA	
TIME	Ora nella forma HH:MM	
INTEGER(p)	Numero intero con precisione *p*	*p* da 1 a 45
SMALLINT	Numero intero con precisione 5	da -32768 a 32767
INTEGER	Numero intero con precisione 10	da -2.147.483.648 a 2.147.483.647
DECIMAL(p,s)	Numero decimale con precisione *p* e *s* cifre decimali	*p* da 1 a 45 e *s* da 0 a *p*
REAL	Numero reale con mantissa di precisione 7	valore 0 oppure valore assoluto da 1E-38 a 1E+38
FLOAT (o DOUBLE PRECISION)	Numero reale con mantissa di precisione 15	valore 0 oppure valore assoluto da 1E-308 a 1E+308
FLOAT(p)	Numero reale con mantissa di precisione *p*	*p* da 1 a 45

- Sito esercitazioni: https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in