I DBMS sono ... efficienti

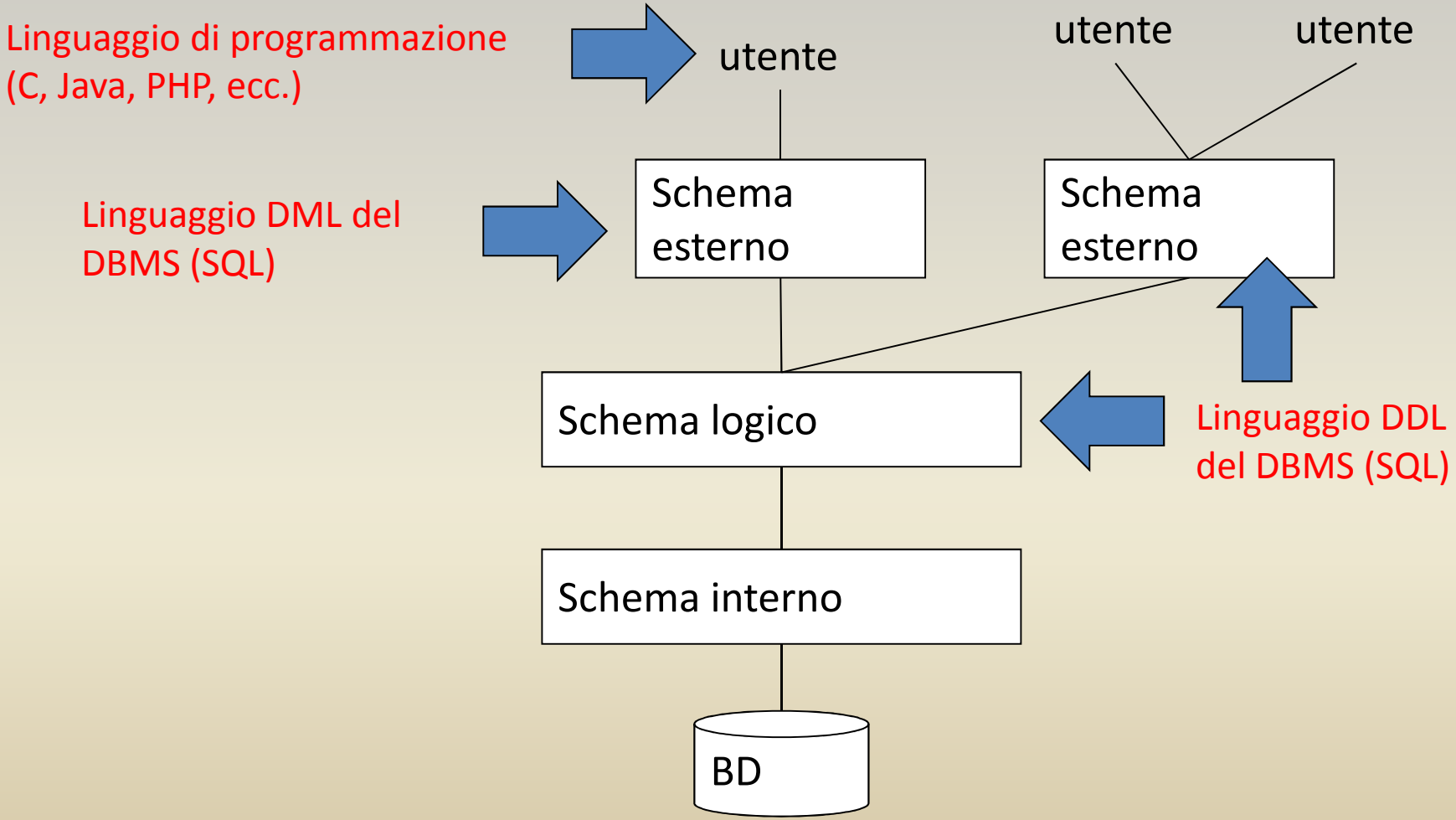
- Cercano di utilizzare al meglio le risorse di spazio di memoria (principale e secondaria) e tempo (**di esecuzione e di risposta**)
- Tecniche specializzate per l'implementazioni dei DBMS con investimenti e competizione

Perché (non) utilizzare MS Access?

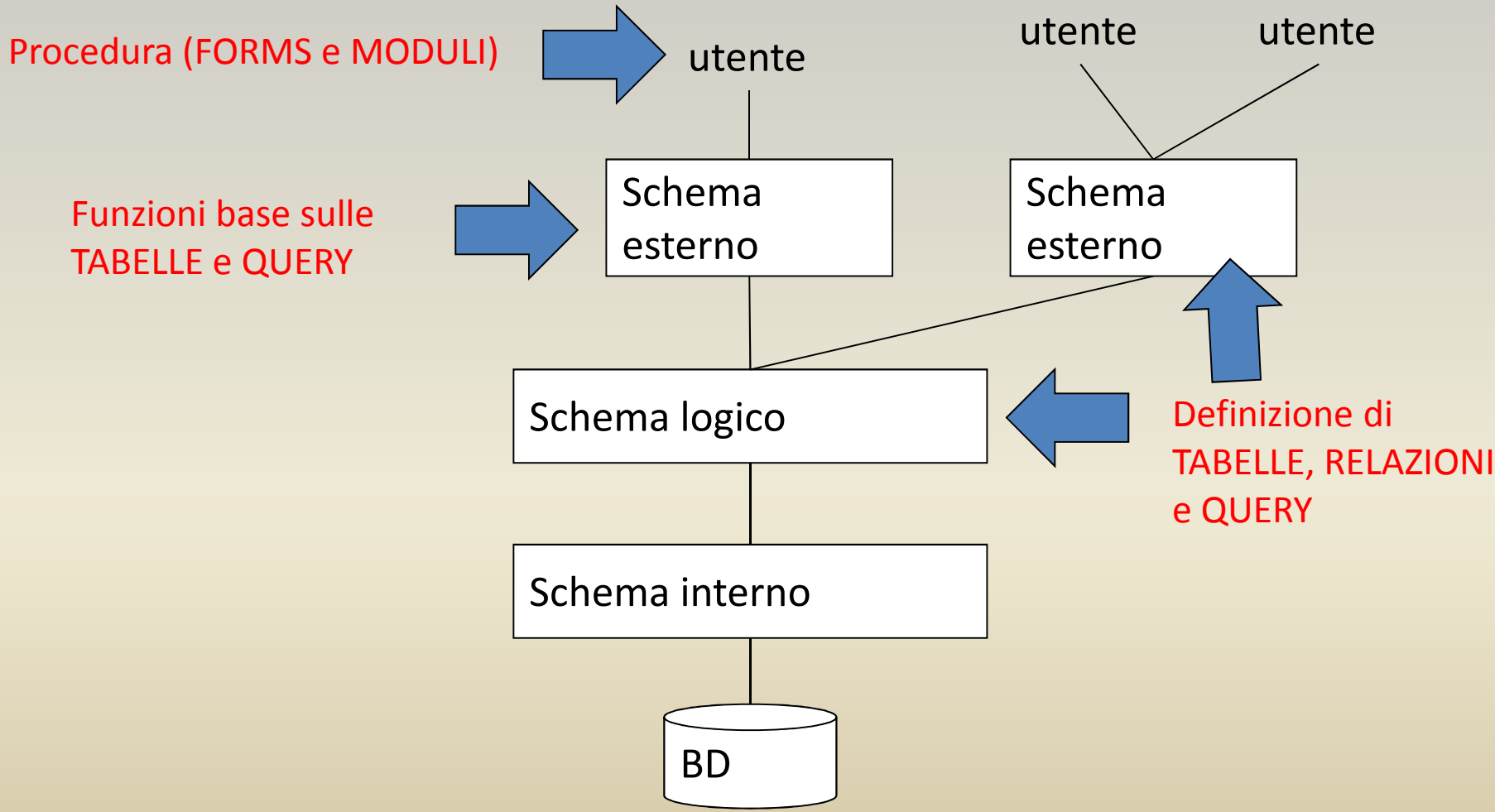
Vantaggi:
• potete creare facilmente il database utilizzando MS Office dal vostro pc, disponendo di una interfaccia facile e intuitiva
• oltre a dover configurare i permessi di scrittura sulla cartella che ospita il db e sul db stesso non avete bisogno di altro
• potete effettuare un back up del database quando desiderate semplicemente copiando il file .MDB

Svantaggi:
• limitazione dichiarata da Microsoft sul numero di connessioni contemporanee al database, che è di 15 (teorico). In pratica con 3 siamo al limite.
• lentezza delle prestazioni in caso di “molti” record oppure multiutenza.

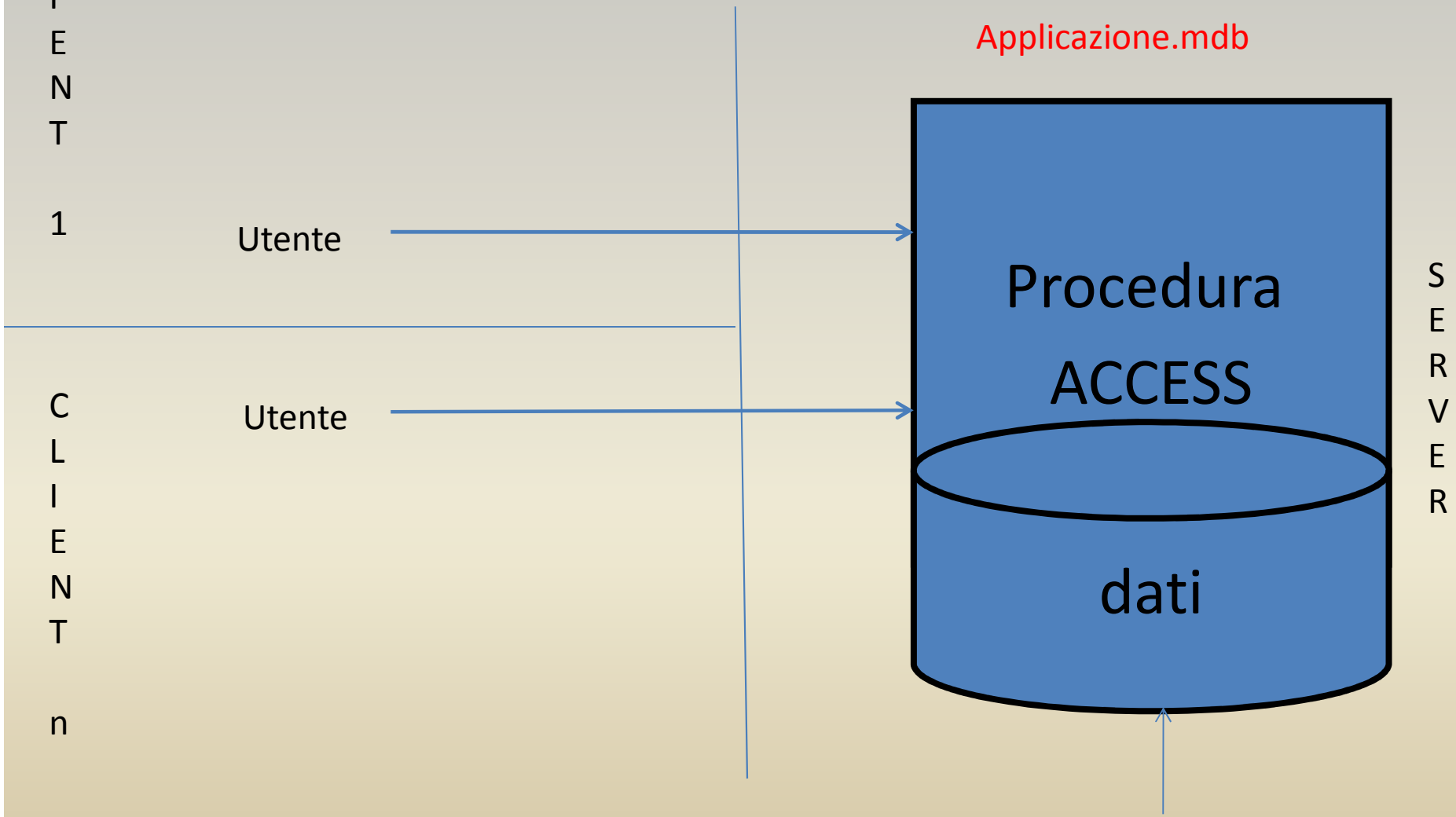
Architettura standard (ANSI/SPARC) a tre livelli per DBMS



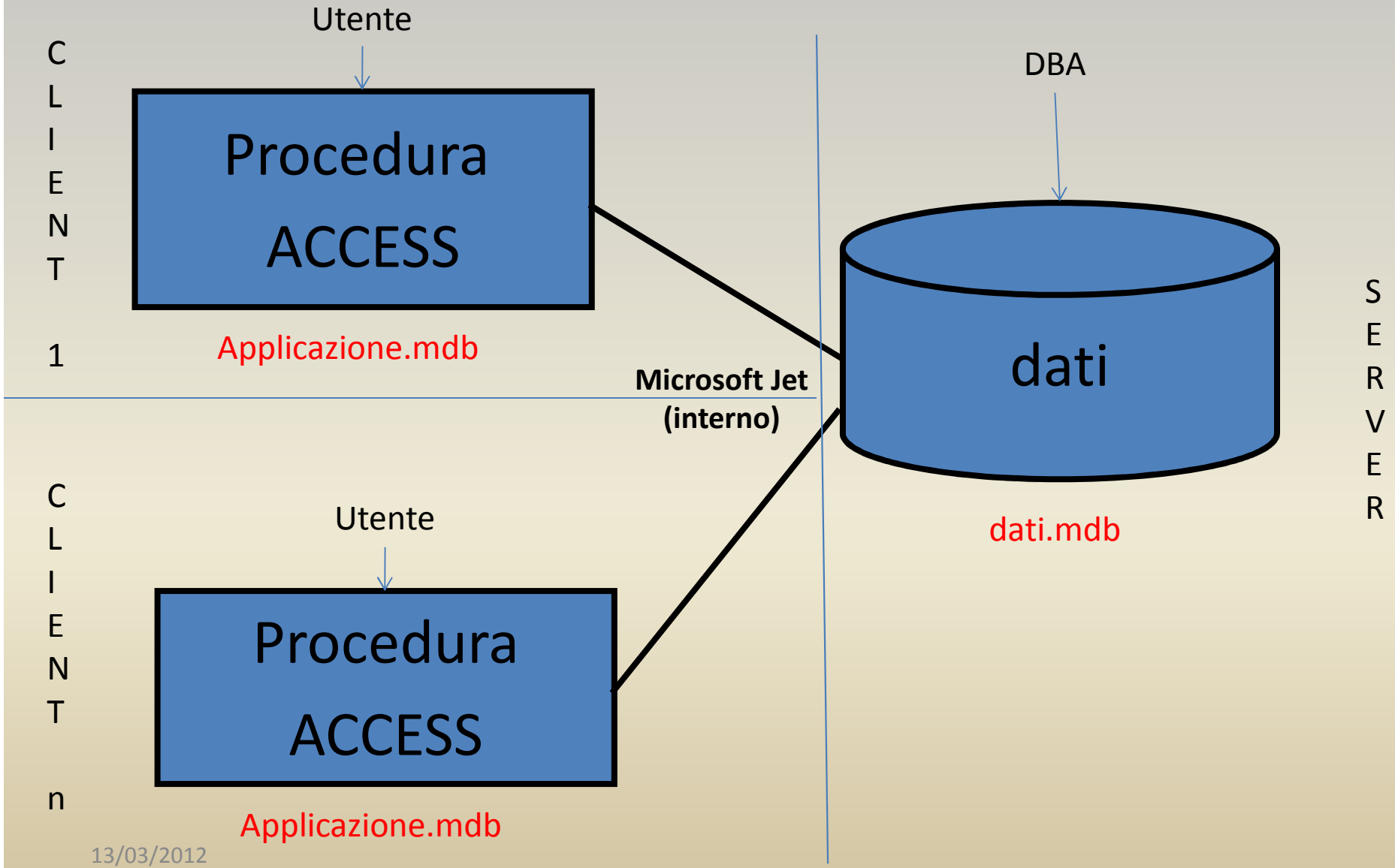
Architettura standard (ANSI/SPARC) MS-ACCESS



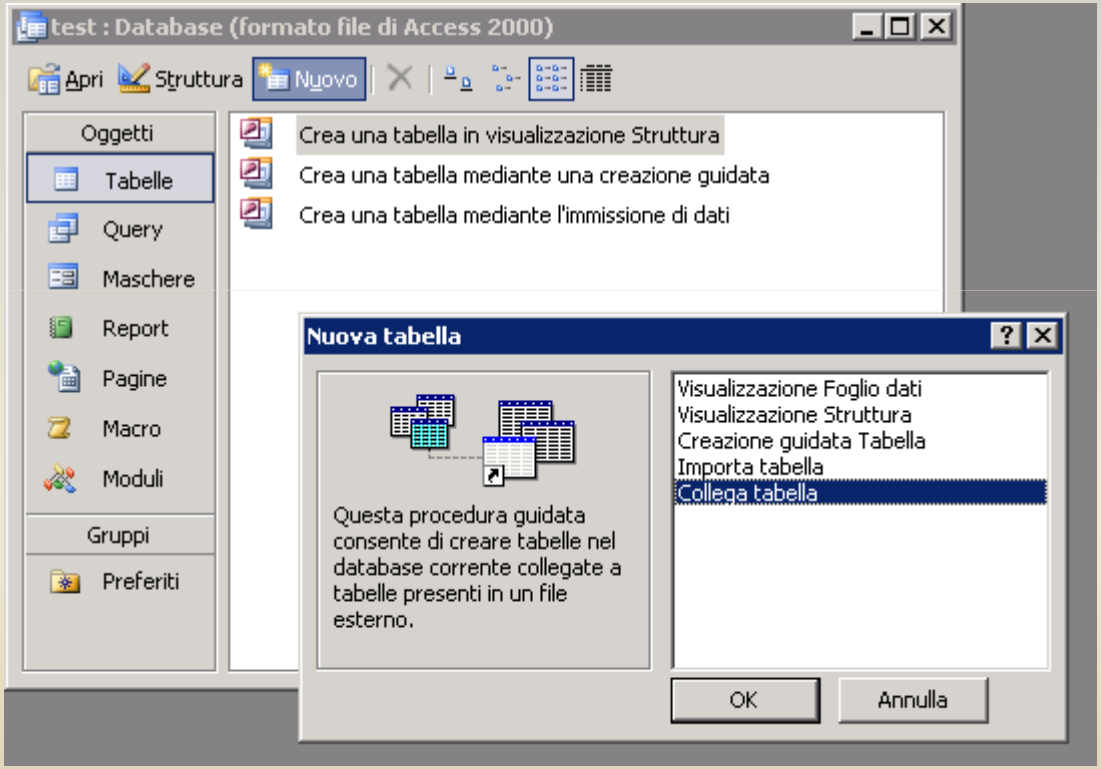
MULTIUTENZA



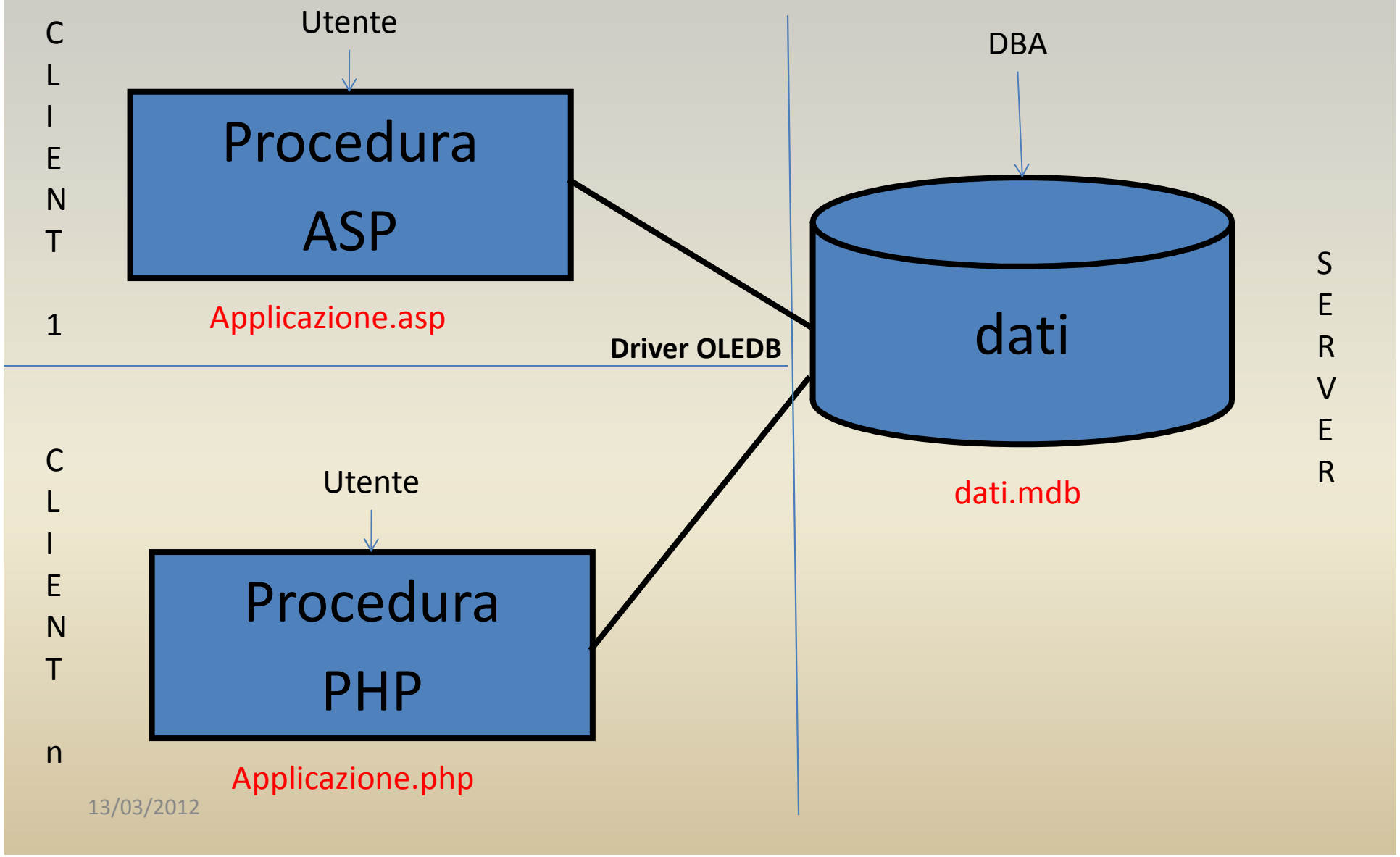
Uso di MS Access (DBMS + Appl.)



Uso di MS Access (DBMS + Appl.)



Uso di MS Access (solo DBMS)



Esempi di connessione ad un DB ACCESS

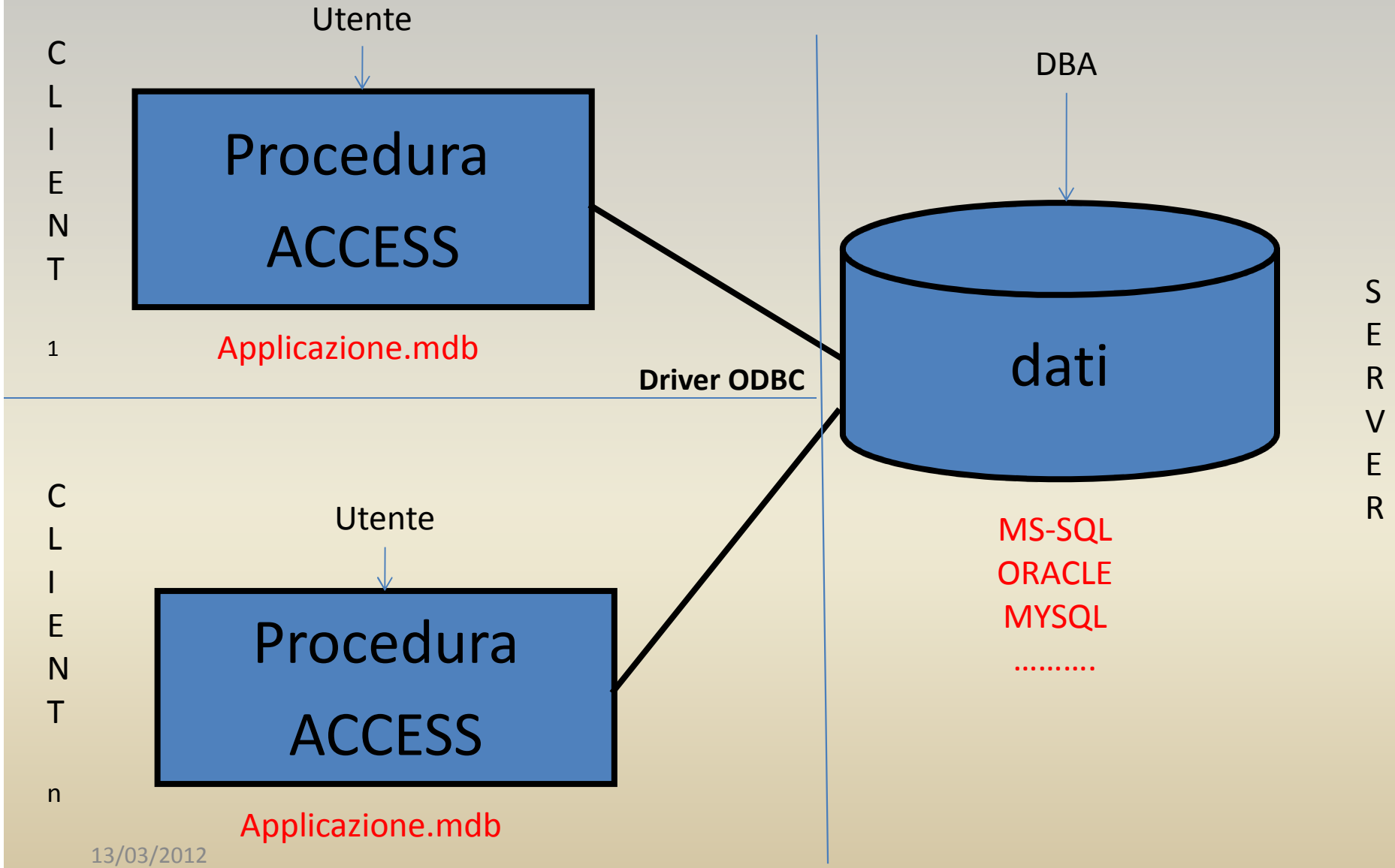
ASP

```
<%
Set conn = Server.CreateObject("ADODB.Connection")
conn.Open "DRIVER={Microsoft Access Driver (*.mdb)};DBQ=" &
server.MapPath("/database/esempio.mdb")
sql = "SELECT * FROM Tabella1"
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn
%>
```

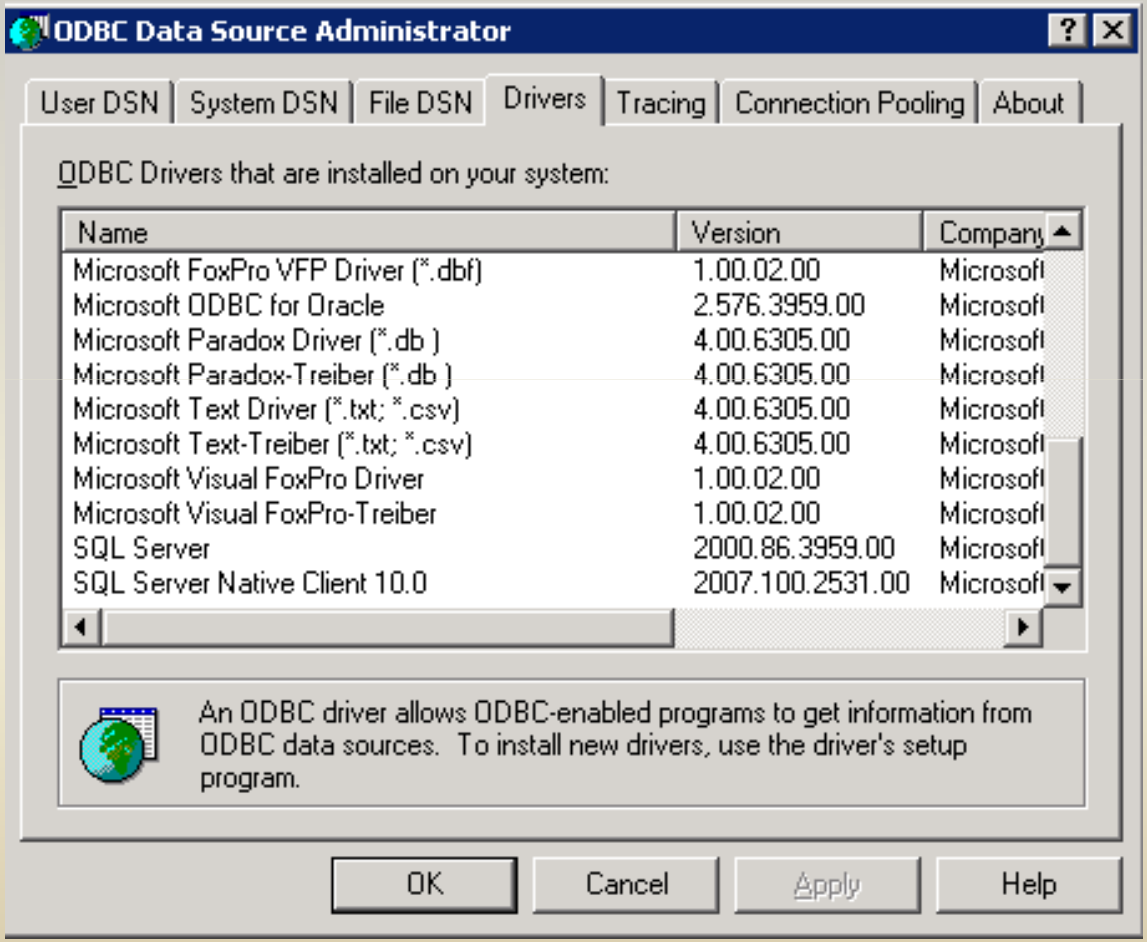
PHP

```
$connessione = new COM("ADODB.Connection");
$stringa_di_connessione = "DRIVER={Microsoft Access Driver (*.mdb)};DBQ=".
realpath("testdb/testdb.mdb");
$connessione->Open($stringa_di_connessione);
$interrogazione = "select * from categorie";
$recordset = new COM("ADODB.Recordset");
```

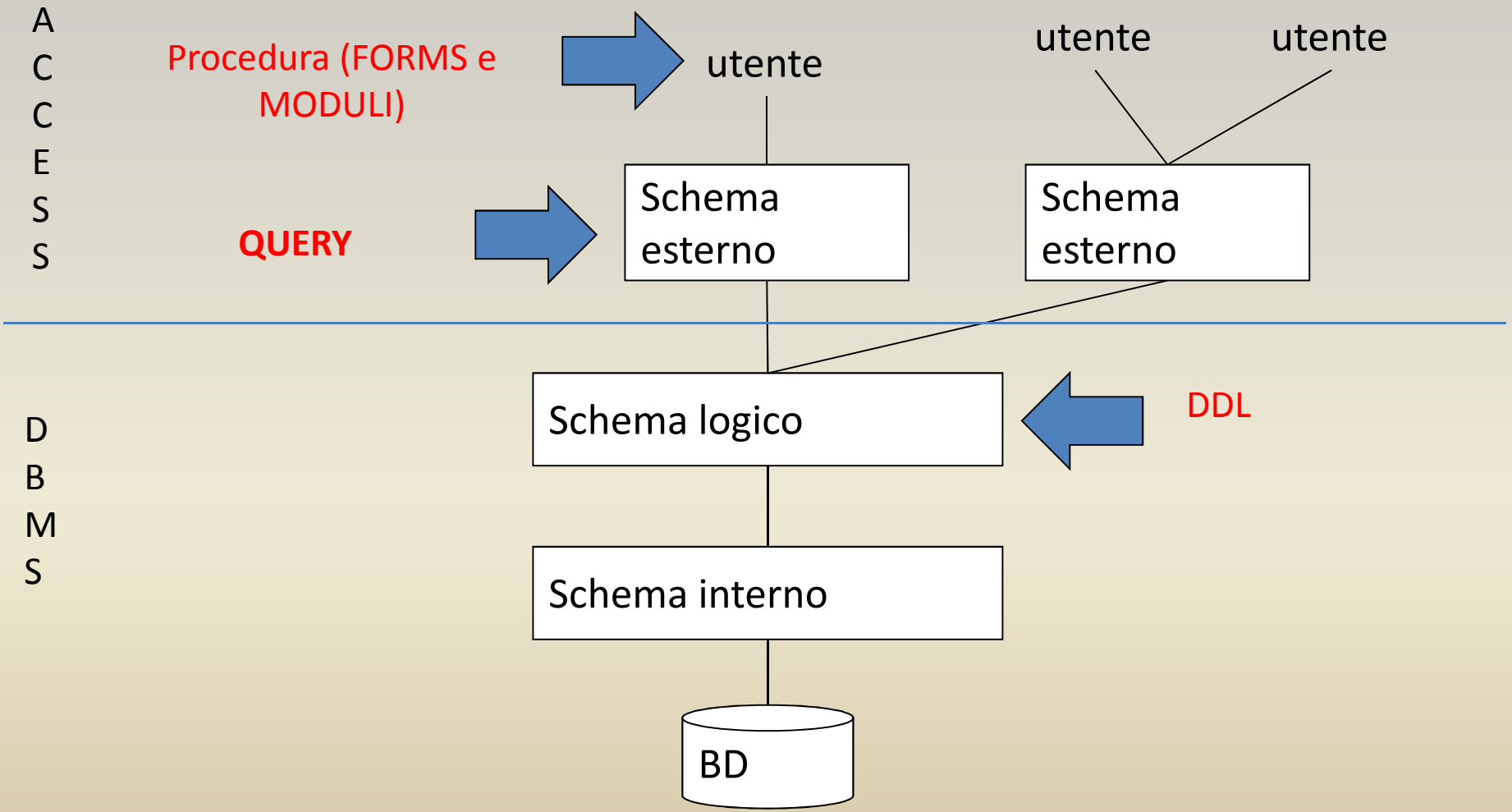
Uso di MS Access (Appl.)



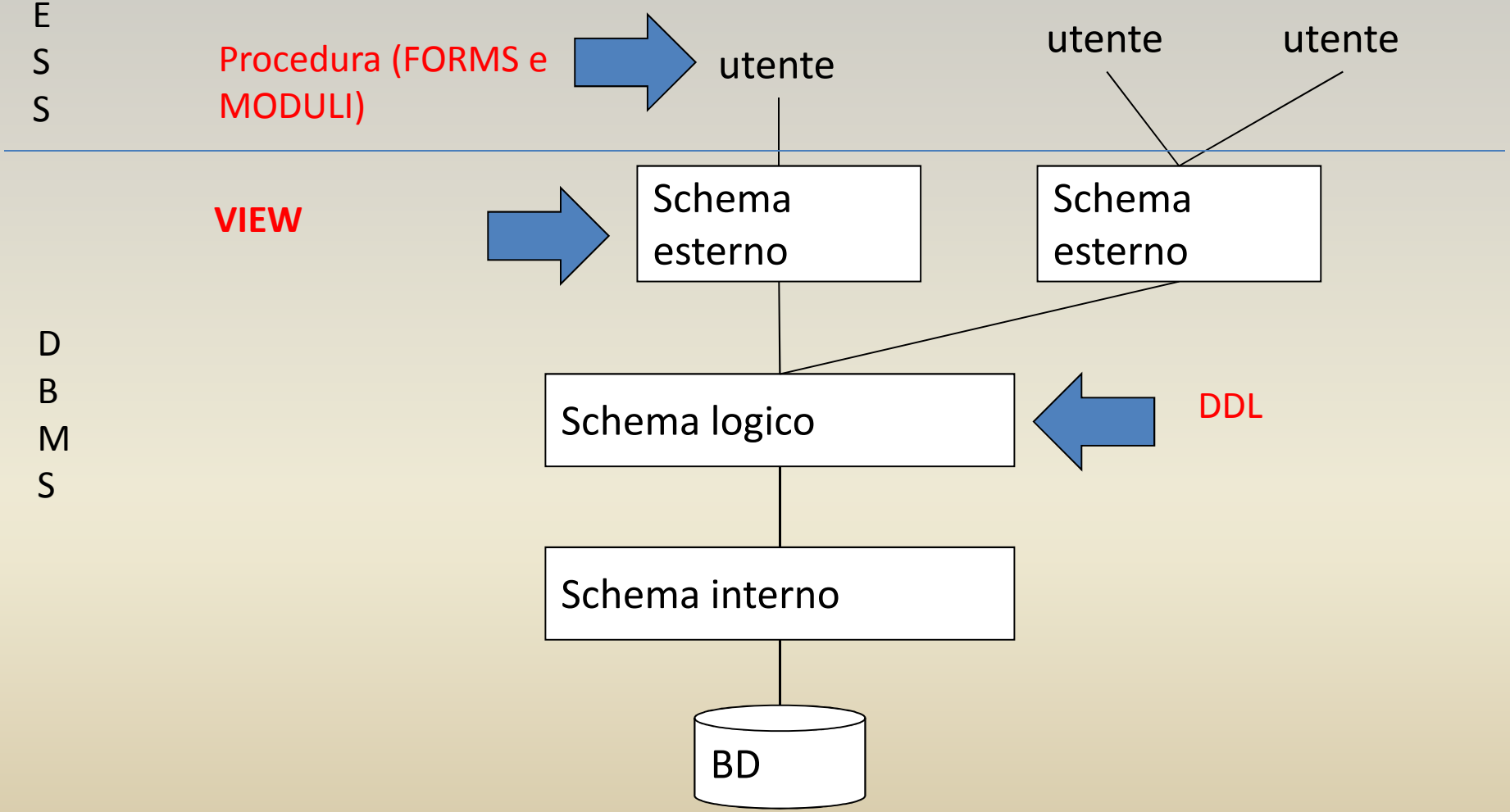
Uso di MS Access (Appl.)

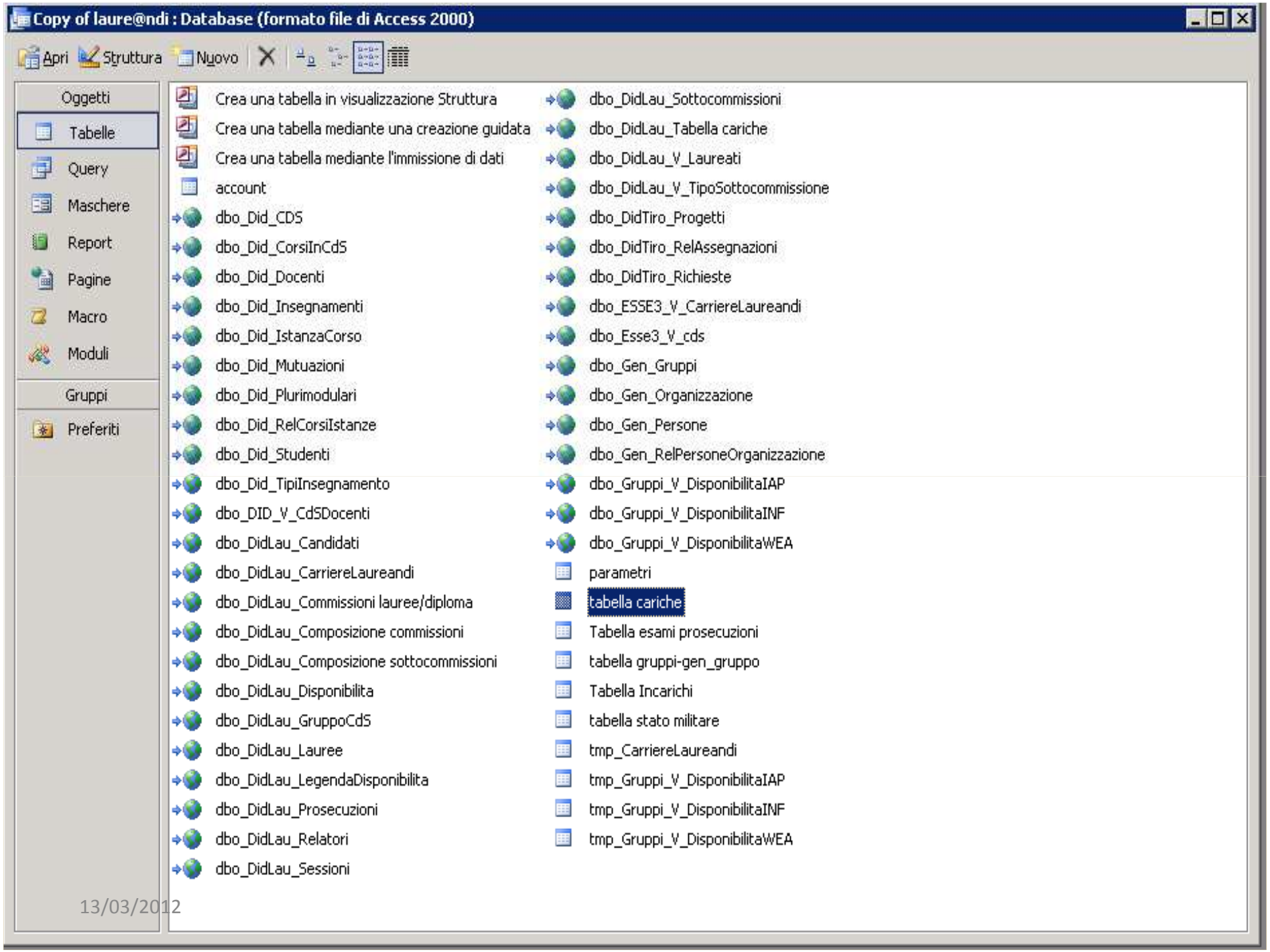


MS-ACCESS Appl.
(soluzione 1 – meno efficiente)



MS-ACCESS Appl. (soluzione 2 – più efficiente)




Copy of laure@ndi : Database (formato file di Access 2000)
Apri
Struttura
Nuovo
Oggetti
Tabelle
Query
Maschere
Report
Pagine
Macro
Moduli
Gruppi
Preferiti
Crea una tabella in visualizzazione Struttura
Crea una tabella mediante una creazione guidata
Crea una tabella mediante l'immissione di dati
account
dbo_Did_CDS
dbo_Did_CorsiInCdS
dbo_Did_Docenti
dbo_Did_Insegnamenti
dbo_Did_IstanzaCorso
dbo_Did_Mutuazioni
dbo_Did_Plurimodulari
dbo_Did_RelCorsiIstanze
dbo_Did_Studenti
dbo_Did_TipiInsegnamento
dbo_DID_V_CdSDocenti
dbo_DidLau_Candidati
dbo_DidLau_CarriereLaureandi
dbo_DidLau_Commissioni lauree/diploma
dbo_DidLau_Composizione commissioni
dbo_DidLau_Composizione sottocommissioni
dbo_DidLau_Disponibilita
dbo_DidLau_GruppoCdS
dbo_DidLau_Lauree
dbo_DidLau_LegendaDisponibilita
dbo_DidLau_Prosecuzioni
dbo_DidLau_Relatori
dbo_DidLau_Sessioni
dbo_DidLau_Sottocommissioni
dbo_DidLau_Tabella cariche
dbo_DidLau_V_Laureati
dbo_DidLau_V_TipoSottocommissione
dbo_DidTiro_Progetti
dbo_DidTiro_RelAssegnazioni
dbo_DidTiro_Richieste
dbo_ESSE3_V_CarriereLaureandi
dbo_Esse3_V_cds
dbo_Gen_Gruppi
dbo_Gen_Organizzazione
dbo_Gen_Persone
dbo_Gen_RelPersoneOrganizzazione
dbo_Gruppi_V_DisponibilitaIAP
dbo_Gruppi_V_DisponibilitaINF
dbo_Gruppi_V_DisponibilitaWEA
parametri
tabella cariche
Tabella esami prosecuzioni
tabella gruppi-gen_gruppo
Tabella Incarichi
tabella stato militare
tmp_CarriereLaureandi
tmp_Gruppi_V_DisponibilitaIAP
tmp_Gruppi_V_DisponibilitaINF
tmp_Gruppi_V_DisponibilitaWEA

Operazioni standard sui dati

- **Inserimento**
- **Ricerca**
- **Modifica**
- **Cancellazione**
- **Selezione**

USO DEGLI INDICI

Esempio: supponiamo di cercare un nome fra i seguenti

Giuseppe
Irene
Fabio
Carlo
Lola
Elena
Fabio

la procedura da fare è:

1. leggi il primo nome della lista
2. se nome trovato fermati
3. leggi prossimo nome torna a 2

Nel caso migliore leggo 1 volta. Nel caso peggiore leggo 7 volte.
Mediamente leggo $\frac{1+7}{2}$ = 4 volte

Se il dato non è presente si casca sempre nel caso peggiore.

USO DEGLI INDICI

Supponiamo ora di organizzare l'elenco precedente in ordine alfabetico:

Carlo
Elena
Fabio
Fabio
Giuseppe
Irene
Lola

Se la procedura è la solita:

1. leggi il primo nome della lista
2. se nome trovato fermati
3. leggi prossimo nome torna a 2

Si ha lo stesso numero di letture. Cambia però il caso del dato non presente che rientra nelle letture medie.

USO DEGLI INDICI

Supponiamo ora cambiare la procedura

Carlo
Elena
Fabio
Fabio
Giuseppe
Irene
Lola

La nuova procedura è:
1 leggi nome centrale
2. se nome trovato fermati
3. se nome > cercato considera la metà superiore del file
4. se nome < cercato considera la metà inferiore del file
5. vai a 1

Nel caso migliore leggo 1 volta. Nel caso peggiore leggo 3 volte.
Mediamente leggo $\frac{1+3}{2}$ = 2 volte

USO DEGLI INDICI

PROBLEMA:

Il caso si presenta più complicato quando aumenta il numero dei campi e il numero delle ricerche possibili.

1)	Giuseppe	Rossi
2)	Irene	Bianchi
3)	Fabio	Verdi
4)	Carlo	Bianchi
5)	Lola	Neri
6)	Elena	Bianchi
7)	Fabio	Rossi

Se si vuol ricercare sia per nome che per cognome non si può dare un'organizzazione unica ai dati e non si può neanche riordinarli prima di ciascuna ricerca.

USO DEGLI INDICI

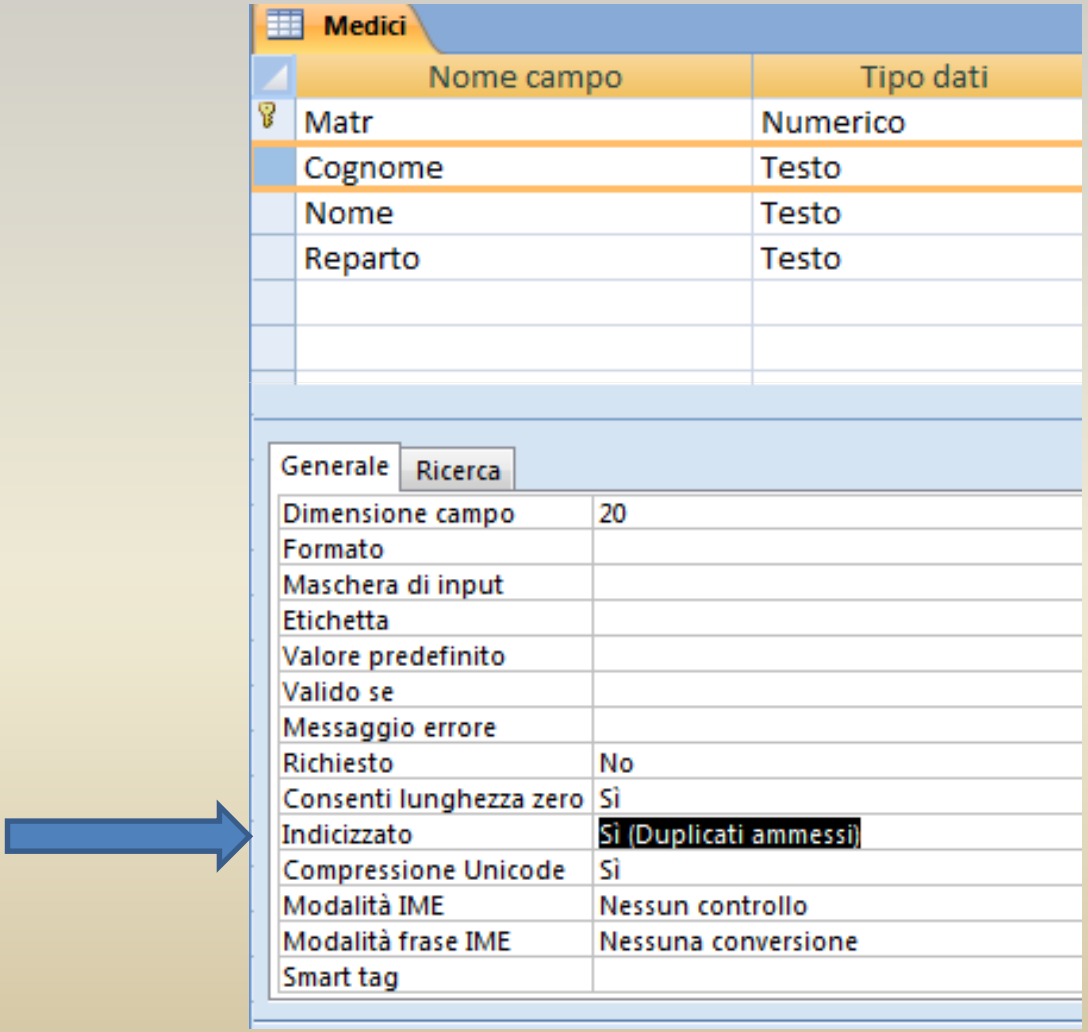
SOLUZIONE:

Si usano degli *indici*: archivi a parte nei quali si ha il riepilogo dei dati organizzati e la maniera di recuperare l'intero record nell'archivio principale che rimane inalterato

Nome	
Carlo	4
Elena	6
Fabio	3, 7
Giuseppe	1
Irene	2
Lola	5

Cognome	
Bianchi	2, 4, 6
Neri	5
Rossi	1, 7
Verdi	3

In ACCESS (struttura tabella)



SQL

- **DDL**
  - "Data Definition Language"
  - definizione degli oggetti dello schema
  - CREATE DATABASE
  - DROP DATABASE
  - CREATE TABLE
  - DROP TABLE
- **DCL**
  - "Data Control Language"
  - utenti e autorizzazioni
- **DML**
  - "Data Manipulation Language"
  - interrogazioni e aggiornamenti
  - INSERT, DELETE, UPDATE
  - SELECT

SQL - DDL

Esercizio 4.3

Dare le definizioni SQL delle tre tabelle

FONDISTA(Nome, Nazione, Età)
GAREGGIA(NomeFondista, NomeGara, Piazzamento)
GARA(Nome, Luogo, Nazione, Lunghezza)

rappresentando in particolare i vincoli di foreign key della tabella GAREGGIA.

Soluzione:

```
Create Table FONDISTA
(
Nome character(20) primary key,
Nazione character(30),
Età smallint
)
Create table GARA
(
Nome character(20) primary key,
Luogo character(20),
Nazione character(20),
Lunghezza integer
)
Create table GAREGGIA
(
NomeFondista character(20) references FONDISTA(Nome),
NomeGara character(20),
Piazzamento smallint,
primary key (NomeFondista, NomeGara),
foreign key (NomeGara) references GARA(Nome)
)
```

SQL - DDL

Esercizio 4.4

Dare le definizioni SQL delle tabelle

AUTORE (Nome, Cognome, DataNascita, Nazionalità)

LIBRO (TitoloLibro, NomeAutore, CognomeAutore, Lingua)
Per il vincolo *foreign key* specificare una politica di *cascade* sulla cancellazione e di *set null* sulle modifiche.

Soluzione:

```
Create table AUTORE
(
Nome character(20),
Cognome character(20),
DataNascita date,
Nazionalità character(20),
primary key(Nome, Cognome)
)
Create table LIBRO
(
TitoloLibro character(30) primary key,
NomeAutore character(20),
CognomeAutore character(20),
Lingua character(20),
foreign key (NomeAutore, CognomeAutore)
references AUTORE(Nome, Cognome)
on delete cascade
on update set NULL
)
```

Esercizio 4.5

Dato lo schema dell'esercizio precedente, spiegare cosa può capitare con l'esecuzione dei seguenti comandi di aggiornamento:

```
delete from AUTORE where Cognome = 'Rossi'

update LIBRO set Nome= 'Umberto'  where Cognome = 'Eco'

insert into AUTORE(Nome,Cognome) values('Antonio','Bianchi')

update AUTORE set Nome = 'Italo' where Cognome = 'Calvino'
```

Soluzione:

1. Il comando cancella dalla tabella AUTORE tutte le tuple con Cognome = 'Rossi'. A causa della politica cascade anche le tuple di LIBRO con CognomeAutore = 'Rossi' verranno eliminate.
2. Il comando non è corretto: Nome e Cognome sono attributi della tabella AUTORE e non della tabella LIBRO.
3. Il comando aggiunge una nuova tupla alla tabella AUTORE. Non ha alcun effetto sulla tabella LIBRO
4. Le tuple di AUTORE con Cognome = Calvino vengono aggiornate a Nome = Italo. A causa della politica set null gli attributi NomeAutore e CognomeAutore delle tuple di Libro con CognomeAutore = Calvino vengono posti a NULL.

Tipi di Dati

All'interno di SQL Server i tipi di dati sono suddivisi nelle seguenti categorie:

- Dati numerici esatti
- Stringhe di testo Unicode
- Numerici approssimati
- Stringhe binarie
- Data e ora
- Altri tipi di dati
- Stringhe di caratteri

Dati numerici esatti

Tipi di dati numerici esatti che utilizzano dati integer:

bigint: dati di dimensioni a 8 byte. Da -2^63 a 2^63-1
int: dati di dimensioni a 4 byte. Da -2^31 a 2^31-1
smallint: dati di dimensioni a 2 byte. Da -2^15 a 2^15-1
tinyint: dati di dimensioni a 1 byte. Da 0 a 255
bit: può assumere il valore 0 o 1. Il valore true viene convertito in 1 e false in 0
decimal: numeri con precisione e scala fisse. Se viene utilizzata la precisione massima, i valori validi sono compresi nell'intervallo da - 10^38 +1 a 10^38 - 1
numeric: funzionalmente è come decimal
money: dati di dimensioni a 8 byte.
smallmoney: dati di dimensioni a 4 byte.

Numerici approssimati

Tipi di dati numerici approssimati da utilizzare con dati numerici a virgola mobile:

float: da - 1,79E+308 a -2,23E-308, 0 e da 2,23E-308 a 1,79E+308
real: da - 3,40E + 38 a -1,18E - 38, 0 e da 1,18E - 38 a 3,40E + 38

Data e ora

Tipi di dati che vengono utilizzati per rappresentare la data e l'ora del giorno:

datetime: 1 gennaio 1753 - 31 dicembre 9999
smalldatetime: 1 gennaio 1900 - 6 giugno 2079

Stringhe e caratteri

Tipo di dati character a lunghezza fissa o variabile:

char: dati di tipo carattere a lunghezza fissa non Unicode con una lunghezza di n byte. n deve essere un valore compreso tra 1 e 8.000. Le dimensioni di archiviazione sono di n byte.
varchar: dati di tipo carattere a lunghezza variabile non Unicode. Le dimensioni di archiviazione sono pari all'effettiva lunghezza dei dati immessi + 2 byte. La lunghezza dei dati immessi può essere uguale a 0 caratteri.
text: dati non Unicode a lunghezza variabile nella tabella codici del server con lunghezza massima di 2^31-1 caratteri.

Stringhe di caratteri UNICODE

http://www.linkas.it/docs/unicode_e_UTF-8_v1_5Uni.html

Tipi di dati carattere che rappresentano dati UNICODE a lunghezza fissa (nchar) o variabile (nvarchar) e utilizzano il set di caratteri UNICODE UCS-2.

nchar: dati Unicode di tipo carattere a lunghezza fissa contenenti n caratteri, dove n deve essere un valore compreso tra 1 e 4.000. Le dimensioni di archiviazione sono pari al doppio di n byte.
nvarchar: dati Unicode di tipo carattere a lunghezza variabile. Le dimensioni di archiviazione, espresse in byte, sono pari al doppio del numero di caratteri immessi + 2 byte. La lunghezza dei dati immessi può essere uguale a 0 caratteri.
ntext: dati Unicode a lunghezza variabile con lunghezza massima di 2^30 - 1 caratteri. Le dimensioni dello spazio di archiviazione, espresse in byte, sono pari al doppio del numero di caratteri immessi.

Stringhe binarie

Tipi di dati binary a lunghezza fissa o variabile.

binary: dati binari a lunghezza fissa con lunghezza di n byte, dove n rappresenta un valore compreso tra 1 e 8.000. Le dimensioni dello spazio di archiviazione corrispondono a n byte.
varbinary: dati binari a lunghezza variabile
image: dati binari a lunghezza variabile da 0 a 2^31-1 byte

Altri tipi di dati

Tutti gli altri tipi di dati che non possono essere catalogati nelle categorie precedenti:

cursor: tipo di dati per variabili o parametri di OUTPUT di stored procedure che contengono un riferimento a un cursore
sql variant: tipo di dati per l'archiviazione di valori per vari tipi di dati supportati da SQL Server 2005
table: tipo di dati speciale utilizzabile per archiviare un set di risultati per l'elaborazione successiva
timestamp: tipo di dati che espone i numeri binari univoci generati automaticamente all'interno di un database
uniqueidentifier: è possibile inizializzare una colonna o variabile locale di tipo uniqueidentifier su un valore specifico
xml: tipo di dati in cui vengono archiviati i dati XML. È possibile archiviare istanze xml in una colonna oppure una variabile di tipo xml.

CREATE TABLE

ESEMPIO

```
CREATE TABLE [dbo].[ANAGRAFICA](
  [ANA_ID] [int] IDENTITY(1,1) NOT NULL,
  [ANA_NOME] [varchar](50),
  [ANA_COGNOME] [varchar](50),
  [ANA_ANNO_NASCITA] [datetime],
  [ANA_INDIRIZZO] [varchar](150),
  [ANA_CIVICO] [smallint],
  [ANA_CAP] [int] NULL,
  [ANA_PROVINCIA] [varchar](2),
  [ANA_NAZIONE] [varchar](2),
  [ANA_TELEFONO] [varchar](20),
  [ANA_CELLULARE] [varchar](20),
  [ANA_EMAIL] [varchar](50),
  CONSTRAINT [PK_ANAGRAFICA] PRIMARY KEY
  )
```

ALTER TABLE

L'operazione di modifica può essere eseguita sia su tabelle prive di record che su tabelle con dati. Attenzione perché se dovessimo modificare il tipo di dati di una determinata colonna, SQL non garantisce l'integrità dei dati.

ESEMPIO

```
ALTER TABLE ANAGRAFICA
  ALTER COLUMN ANA_NOME VARCHAR(100)
```

È possibile modificare o eliminare un campo, tramite il comando DROP, oppure un indice o una chiave.

SQL

- **DDL**
  - "Data Definition Language"
  - definizione degli oggetti dello schema
  - CREATE DATABASE
  - DROP DATABASE
  - CREATE TABLE
  - DROP TABLE
- **DCL**
  - "Data Control Language"
  - utenti e autorizzazioni
- **DML**
  - "Data Manipulation Language"
  - interrogazioni e aggiornamenti
  - INSERT, DELETE, UPDATE
  - SELECT

SELECT

In questa clausola specifichiamo quali sono i campi della tabella, o delle tabelle, che vogliamo visualizzare.

SELECT * indica che vogliamo visualizzare tutti i campi. Se nella clausola SELECT sono presenti più tabelle, si userà la clausola NOMETABELLA.* per visualizzare tutti i campi.
Se si vuole assegnare un nuovo nome ad un campo o ad una tabella, si possono usare gli *Alias*, ovvero i sinonimi. SELECT CAMPO AS ALIAS, il campo si chiamerà ALIAS.

ESEMPIO

```
--Selezione di tutti i campi
SELECT *

--Selezione di campi mirata
SELECT [CAMPO01], [CAMPO02]

--Selezione con alias dei campi
SELECT [CAMPO01] AS NOME, [CAMPO02] AS COGNOME
```

FROM

Nella clausola FROM, vengono indicati i nomi delle tabelle o viste che si vuole interrogare.

ESEMPIO

```
--Selezione di tutti i campi
SELECT * FROM [TABELLA]

--Selezione di campi mirata da una o più tabelle
SELECT [TABELLA].[CAMPO01], [TABELLA02].[CAMPO02] FROM [TABELLA],
[TABELLA02]

--Selezione con alias dei campi
SELECT ANAGRAFICA.[CAMPO01] AS NOME, ANAGRAFICA.[CAMPO02] AS COGNOME
FROM [TABELLA] AS ANAGRAFICA
```

WHERE

La clausola WHERE viene usata per effettuare un vero e proprio *filtraggio* delle informazioni. Nella clausola WHERE possiamo indicare dei possibili valori di un determinato campo.
Se le condizioni da valutare sono più di una, la prima clausola where sarà seguita da un *AND* o da un *OR*.
Con la parola *IN* possiamo indicare diversi valori per i quali la clausola WHERE viene soddisfatta.

ESEMPIO

```
--Selezione di tutti i campi con condizione semplice
SELECT * FROM [TABELLA] WHERE [CAMPO] = 'VALORE‘

--Selezione con più condizioni
SELECT * FROM [TABELLA] WHERE [CAMPO01] = 'VALORE' AND [CAMPO02] = 123

--Selezione con clausola IN
SELECT * FROM [TABELLA] WHERE [PROVINCIA] IN ('VA', 'BG', 'CO', 'MI')
```

ORDER BY

Una volta che abbiamo filtrato i dati dobbiamo ordinarli. Nella clausola Order by, dobbiamo indicare quali sono i campi da ordinare, seguiti da virgola e dalla condizione di ordinamento (*ASC* ascendente o *DESC* discendente).

ESEMPIO

```
--Selezione di tutti i campi con ordine
SELECT * FROM [TABELLA] ORDER BY [CAMPO01] ASC

--Selezione con più condizioni
SELECT * FROM [TABELLA] ORDER BY [CAMPO01] ASC,[CAMPO01] DESC
```

DISTINCT

Può capitare di avere a disposizione una tabella con molti dati ridondanti. Per esempio la tabella ordini denormalizzata, potrebbe contenere parecchie volte il nome Mario Rossi, perchè questa persona ha effettuato diversi ordini. Con la clausola DISTINCT, chiediamo al database di mostrarci solamente 1 riga dove quel campo contiene quel valore. Quindi la parola DISTINCT restituisce l'unicità delle righe per quelle colonne specificate nella SELECT.

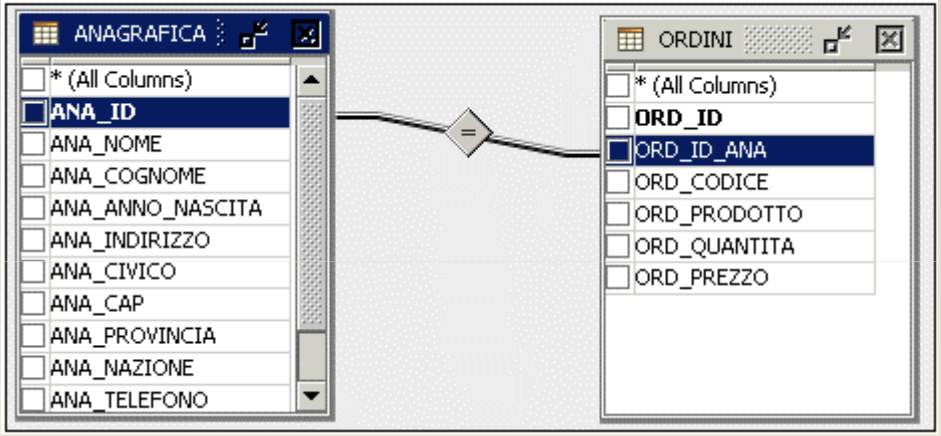
ESEMPIO

```
--Selezione valore unico
SELECT DISTINCT [CAMPO01], [CAMPO02] FROM [TABELLA]
```

I Join

I Join sono particolari istruzioni SQL che riescono a legare tra loro i dati di diverse tabelle, per estrapolare le informazioni legate tra di loro.

Immagine 1. Legame tra tabelle.



Inner Join

Outer Join

LEFT OUTER JOIN

RIGHT OUTER JOIN

FULL OUTER JOIN

Inner Join

Se, nell'esempio di figura 1, vogliamo individuare solamente le righe di Anagrafica, per le quali esistono record nella tabella Ordini, usiamo un Inner Join.

```
SELECT
   [ANAGRAFICA].*, [ORDINI].*
FROM
   [dbo].[ANAGRAFICA]
INNER JOIN
   [dbo].[ORDINI]
ON
   [ANAGRAFICA].[ANA_ID] = [ORDINI].[ORD_ID_ANA]
```

Occorre indicare quali campi vogliamo visualizzare. Poi bisogna elencare la prima tabella seguita da inner join e la seconda tabella, che partecipano al legame. Infine va indicato nell'ordine uno a molti, il meccanismo di paragone per formare il legame

LEFT OUTER JOIN

Nel caso in cui volessimo visualizzare tutti gli utenti, compresi quelli che non hanno mai effettuato ordini.

```
SELECT
   [ANAGRAFICA].*, [ORDINI].*
FROM
   [dbo].[ANAGRAFICA]
LEFT OUTER JOIN
   [dbo].[ORDINI]
ON
   [ANAGRAFICA].[ANA_ID] = [ORDINI].[ORD_ID_ANA]
```

Bisogna tener conto che in questo caso, se un utente non ha effettuato ordini, troveremo una sola riga come risultato, ma i campi relativi l'ordine torneranno un valore di tipo *dbNull*.

RIGHT OUTER JOIN

In questo caso, invece, vogliamo vedere tutti gli ordini.

ESEMPIO

```
SELECT
   [ANAGRAFICA].*, [ORDINI].*
FROM
   [dbo].[ANAGRAFICA]
RIGHT OUTER JOIN
   [dbo].[ORDINI]
ON
   [ANAGRAFICA].[ANA_ID] = [ORDINI].[ORD_ID_ANA]
```

FULL OUTER JOIN

Nell'ultimo esempio vediamo tutti i record della tabella Anagrafica e tutti quelli della tabella Ordini, con e senza legami.

```
SELECT
   [ANAGRAFICA].*, [ORDINI].*
FROM
   [dbo].[ANAGRAFICA]
FULL OUTER JOIN
   [dbo].[ORDINI]
ON
   [ANAGRAFICA].[ANA_ID] = [ORDINI].[ORD_ID_ANA]
```

CREATE VIEW

Il comando per la creazione di una vista, richiede la presenza di una istruzione SELECT nella quale bisogna specificare i campi che vogliamo visualizzare.

ESEMPIO

```
CREATE VIEW [dbo].[ANA_ORDINI]
AS
SELECT
  dbo_ANAGRAFICA.ANA_NOME,
  dbo_ANAGRAFICA.ANA_COGNOME,
  dbo_ORDINI.ORD_PRODOTTO,
  dbo_ORDINI.ORD_QUANTITA,
  dbo_ORDINI.ORD_PREZZO
FROM
  dbo_ANAGRAFICA INNER JOIN
  dbo_ORDINI ON dbo_ANAGRAFICA.ANA_ID = dbo_ORDINI.ORD_ID_ANA
```

ALTER VIEW

Per modificare una vista, si usa l'istruzione *ALTER VIEW*. Dopo il comando ALTER dovremo inserire le opzioni o i campi da modificare. Possiamo, ad esempio, modificare completamente l'istruzione di SELECT oppure cambiare semplicemente la clausola WHERE.

ESEMPIO

```
ALTER VIEW [dbo].[ANA_ORDINI]
AS
SELECT   dbo_ANAGRAFICA.ANA_NOME,
  dbo_ANAGRAFICA.ANA_COGNOME,
  dbo_ORDINI.ORD_PRODOTTO,
  dbo_ORDINI.ORD_QUANTITA,
  dbo_ORDINI.ORD_PREZZO
FROM dbo_ANAGRAFICA INNER JOIN dbo_ORDINI
  ON dbo_ANAGRAFICA.ANA_ID = dbo_ORDINI.ORD_ID_ANA
WHERE dbo_ORDINI.ORD_PREZZO > 100.00
```

Esercizi

Consideriamo lo schema:

1. Selezionare il NOME e il CAP tutte le località della provincia di Pisa (Sigla='PI')
2. Selezionare il NOME, la SIGLA della provincia ed il CAP tutte le località della 'Toscana'
3. Selezionare il NOME e la SIGLA della provincia tutte le località il cui CAP inizia per '56'
4. Selezionare tutte le località il cui CAP inizia per '1' e non sono in "Piemonte" (3 soluzioni)

Soluzioni

1. **SELECT** località.cap, località.NOME **FROM** località **INNER JOIN**
   Province **ON** località.provincia = Province.CODICE
   WHERE (Province.SIGLA = 'PI')

2. **SELECT** località.cap, località.NOME, Province.SIGLA
   FROM località **INNER JOIN**
   Province **ON** località.provincia = Province.CODICE
   INNER JOIN Regioni **ON** Province.regione = Regioni.codice
   WHERE (Regioni.nome = 'Toscana')

3. **SELECT** località.cap, località.NOME **FROM** località **INNER JOIN**
   Province **ON** località.provincia = Province.CODICE
   WHERE (località.cap LIKE '56%')

(Verificare che 1 e 3 sono equivalenti: quale sarà più efficiente ?)

Soluzioni (segue)

4.1 **SELECT** località.NOME, località.cap **FROM** località **INNER JOIN**
Province **ON** dbo_località.provincia = dbo_Province.CODICE **INNER JOIN**
dbo_Regioni **ON** dbo_Province.regione = dbo_Regioni.codice
WHERE (dbo_località.cap LIKE '1%') AND (dbo_Regioni.nome <> 'Piemonte')

4.2 **SELECT** località.NOME, località.cap **FROM** località **INNER JOIN**
Province **ON** località.provincia = Province.CODICE
WHERE (località.cap LIKE '1%') **AND**
(Province.regione **NOT IN**
(**SELECT** codice **FROM** Regioni **WHERE** (nome = 'piemonte')))

4.3 **SELECT** località.codice, località.nome **FROM** (località **INNER JOIN**
Province **ON** località.provincia = Province.CODICE **INNER JOIN**
Regioni **ON** Province.regione = Regioni.codice)
WHERE cap LIKE '1%'
EXCEPT (SELECT codice, nome FROM [*località del piemonte*])

Verificare che le 3 soluzioni sono equivalenti: qual è la più efficiente ?