Giampiero Granatella

Introduzione a SQL

SQL è il linguaggio standard dell'ANSI (American National Standards Institute) per accedere a database.

Cos'è SQL?

- SQL è l'acronimo di **S**tructured **Q**uery **L**anguage
- SQL permette di accedere a un database
- SQL è un linguaggio standard ANSI
- SQL serve per formulare domande (query) ad un DB
- SQL serve per ricevere dati da un DB
- SQL serve per inserire dati in un DB
- SQL serve per cancellare dati in un DB
- SQL serve per modificare aggiornare dati in un DB
- SQL è molto semplice da imparare

SQL è uno Standard

SQL è uno standard ANSI (American National Standards Institute).

SQL serve per accedere ai dati di tutti I più famosi DBMS come Access, DB2, Informix, Microsoft SQL Server, Oracle, Sybase e altri (ma sfortunatamente molti di questi hanno estensioni al linguaggio proprietarie).

Tabelle

I Database contengono oggetti chiamati tabelle (**Tables)**.

Records di dati sono memorizzati in queste tabelle. Le tabelle sono identificate univocamente con un nome (like "Persons", "Orders", "Suppliers").

Le tabelle contengono **Colonne e Righe** con dati. Le righe contengono records. Le colonne contengono dati (come First Name, Last Name, Address, e City).

Di seguito viene mostrata una tabella chiamata "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

LastName, FirstName, Address, e City sono le colonne della tabella. Le righe contengono tre record con I dati di tre persone.

SQL Queries

Con SQL, si può interrogare (**Query)** un DB e avere I risultati (**Result)** in forma tabellare.

Una query come questa:

```
SELECT LastName FROM Persons
```

Da questo risultato:

LastName
Hansen

Svendson
Pettersen

Nota: Alcuni DBMS richiedono il punto e virgola alla fine dell'istruzione (**statement**).

SQL Data Manipulation

Come lo stesso nome suggerisce, SQL serve per eseguire queries. Ma il linguaggio SQL permette anche di aggiornare, inserire e cancellare record.

Tuute queste istruzioni formano il Data Manipulation Language (DML), parte di SQL:

- SELECT – estrae dati
- UPDATE – aggiorna dati
- DELETE – cancella dati
- INSERT – inserisce nuovi dati

SQL Data Definition

Il Data Definition Language (DDL), anch'esso parte di SQL permette di creare o cancellare le tabelle. Si possono inoltre definire indici, collegamenti fra le tabelle, e definire vincoli sui dati..

Le più importanti istruzioni DDL sono:

- CREATE TABLE – crea una nuova tabella
- ALTER TABLE - alters (modifica) una tabella
- DROP TABLE - cancella una tabella
- CREATE INDEX – crea un indice (search key)
- DROP INDEX – cancella un indice

SQL e la programmazione

SQL è utilizzato embedded in linguaggi di programmazione come PHP, ASP, Java, VB e C++…

SQL Select Statement

L'istruzione SELECT seleziona colonne di dati da un DB.

Il risultato è fornito in una tabella (chiamato the result set).

La sintassi dell'istruzione è la seguente

```
SELECT column_name(s) FROM table_name
```

Esempio: Selezionare colonne da una tabella

Per selezionare le colonne "LastName" e "FirstName" si usa l'istruzione SELECT nel seguente modo:

```
SELECT LastName,FirstName FROM Persons
```

La tabella "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes

Pettersen	Kari	Storgt 20	Stavanger

Il risultato (Result Set):

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Esempio: Selezionare tutte le colonne

Per selezionare tutte le colonne della tabella "Persons" si usa un * invece del nome delle colonne:

```
SELECT * FROM Persons
```

Risultato:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

La tabella risultato

Il risultato di un istruzione SQL è memorizzato in un result set. Il result set può essere visto come una tabella risultato. La maggioranza dei DBMS permettono la navigazione del result set con funzioni come Move-To-First-Record, Get-Record-Content, Move-To-Next-Record.....

Queste funzioni non fanno parte del linguaggio SQl.

Il punto e virgola alla fine di un istruzione?

Molti tutorial richiedono il punto e virgola dopo ogni istruzione SQL, questo viene richiesto dalla maggioranza dei DB

La clausola WHERE

La clausola WHERE viene utilizzata per specificare un criterio di ricerca.

La clausola WHERE

La sintassi della clausola WHERE è la seguente:

```
SELECT column FROM table WHERE column condition value
```

Con il WHERE possono essere utilizzate le seguenti condizioni:

Operator	Condition
=	Equal
< >	Not equal
>	Greater than
<	Less than
> =	Greater than or equal
< =	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Explained below

Notae: In alcuni “dialetti” SQL il diverso <> può essere scritto come !=

Esempio: Selezionare una persona di una determinate città

Per selezionare solo le persone che vivono in Sandnes, Aggiungere una clausola WHERE come la seguente all’istruzione SELECT:

```
SELECT * FROM Persons WHERE City='Sandnes'
```

La tabella persone:

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980
Pettersen	Kari	Storgt 20	Stavanger	1960

Risultato:

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980

Utilizzo dei Quotes (‘)

SQL utilizza l’apice ‘ per racchiudere costanti testuali. I valori numerici non richiedono l’apice .

Per valori Testo:

```
This is correct:
SELECT * FROM Persons WHERE FirstName='Tove'
This is not correct:
SELECT * FROM Persons WHERE FirstName=Tove
```

Per valori Numerici:

```
This is correct:
SELECT * FROM Persons WHERE Year>1965
This is not correct:
SELECT * FROM Persons WHERE Year>'1965'
```

La condizione LIKE

La condizione LIKE è usata per ricercare un determinato pattern in una colonna.

La sintassi è la seguente:

```
SELECT column FROM table WHERE column LIKE pattern
```

Il simbolo "%" è utilizzato per indicare una sequenza di caratteri non specificata.

Esempio: Selezionare una persona con un determinato Pattern nel nome

Questo staterment trova tutte le persone il cui nome inizia per 'O'.

```
SELECT * FROM Persons WHERE FirstName LIKE 'O%'
```

Questo staterment trova tutte le persone il cui nome finisce per 'a'

```
SELECT * FROM Persons WHERE FirstName LIKE '%a'
```

Questo staterment trova tutte le persone il cui nome contiene 'la'

```
SELECT * FROM Persons WHERE FirstName LIKE '%la%'
```

Tutti gli esmpi forniscono il seguente risultato:

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951

SQL And & Or

AND & OR

AND e OR uniscono due o più condizioni in una clausola WHERE.

L'operatore AND mostra le righe che soddisfano TUTTE le condizioni. L'operatore OR mostra tutte le righe che soddisfano ALMENO UNA condizione.

Tabella originale

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Esempio

Utilizzare AND per avere tutte le persone che abbiamo per nome "Tove", e cognome "Svendson":

```
SELECT * FROM Persons
WHERE FirstName='Tove'
AND LastName='Svendson'
```

Risultato:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes

Esempio

Utilizzare OR per mostrare tutte le persone il cui nome sia "Tove", o il cognome "Svendson":

```
SELECT * FROM Persons
WHERE firstname='Tove'
OR lastname='Svendson'
```

Risultato:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Esempio

Si possono combinare AND e OR (usando parentesi per formulare espressioni complesse):

```
SELECT * FROM Persons WHERE
(FirstName='Tove' OR FirstName='Stephen')
AND LastName='Svendson'
```

Risultato:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

SQL Between...And

BETWEEN ... AND

L'operatore BETWEEN ... AND seleziona un intervallo range di dati fra due valori.

```
SELECT column_name FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

Tabella usata nell'esempio

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Nordmann	Anna	Neset 18	Sandnes
Pettersen	Kari	Storgt 20	Stavanger
Svendson	Tove	Borgvn 23	Sandnes

Esempio 1

Per mostrare le persone alfabeticamente comprese fra "Hansen" (incluso) and "Pettersen" (escluso), l'istruzione SQL è la seguente:

```
SELECT * FROM Persons WHERE LastName
BETWEEN 'Hansen' AND 'Pettersen'
```

Risultato:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Nordmann	Anna	Neset 18	Sandnes

Esempio 2

Per mostrare le persone fuori dall'intervallo si usa l'operatore NOT:

```
SELECT * FROM Persons WHERE LastName
NOT BETWEEN 'Hansen' AND 'Pettersen'
```

Risultato:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Svendson	Tove	Borgvn 23	Sandnes

SQL Select Distinct

La parola chiave DISTINCT viene utilizzata per ricavare solo i valori differenti (distinti).

DISTINCT

L'istruzione SELECT resistuisce le informazioni di una colonna. Ma cosa usare se vogliamo evitare la ripetizione di informazione?

Con SQL, tutto quello di cui abbiamo bisogno è l'aggiunta della parola chiave DISTINCTall'istruzione SELECT con la sintassi seguente:

```
SELECT DISTINCT column-name(s) FROM table-name
```

Esempio: Selezionare I fornitori da una tabella di vendite

Esempio: Tabella delle vendite:

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

L'istruzione SQL:

```
SELECT Company FROM Orders
```

Fornisce il seguente risultato:

Company

Sega
W3Schools
Trio
W3Schools

Nota che la ditta W3Schools compare due volte.

Esempio: Select Distinct Companies from Orders

Questa istruzione:

```
SELECT DISTINCT Company FROM Orders
```

Fornisce il seguente risultato:

Company
Sega
W3Schools
Trio

W3Schools compare una sola volta.

SQL Order By

La parola chiave ORDER BY è usata per ordinare il risultato.

Ordinare le Righe

ORDER BY è usata per ordinare le righe.

Orders:

Company	OrderNumber
Sega	3412
ABC Shop	5678
W3Schools	2312
W3Schools	6798

Esempio

Per ordinare le ditte in ordine alfabetico:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company
```

Risultato:

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	6798
W3Schools	2312

Esempio

Per ordinare le ditte E e il numero di ordine:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company, OrderNumber
```

Result:

Company	OrderNumber
ABC Shop	5678
Sega	3412

W3Schools	2312
W3Schools	6798

Esempio

Per effettuare un ordinam,ento inverso:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company DESC
```

Risultato:

Company	OrderNumber
W3Schools	6798
W3Schools	2312
Sega	3412
ABC Shop	5678

SQL INSERT INTO

Inserire nuove righe

L'istruzione INSERT INTO inserisce nuove righe in una tabella, la sintassi è la seguente:

```
INSERT INTO table_name
VALUES (value1, value2,....)
```

Si possono inoltre specificare le colonne su cui inserire I dati:

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

Inserire una nuova riga

La tabella "Persons":

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger

E l'istruzione SQL:

```
INSERT INTO Persons
VALUES ('Hetland', 'Camilla', 'Hagabakka 24', 'Sandnes')
```

Da i seguenti risultati:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

Inserire Dati in Colunne specificate

Tabella Persone:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

L'istruzione SQL:

```
INSERT INTO Persons (LastName, Address)
VALUES ('Rasmussen', 'Storgt 67')
```

Da il seguente risultato:

LastName	FirstName	Address	City

Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes
Rasmussen		Storgt 67	

SQL Update

Aggiornare righe

L'istruzione UPDATE aggiorna o modifica righe:

```
UPDATE table_name SET column_name = new_value
WHERE column_name = some_value
```

Person:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen		Storgt 67	

Aggiornare un campo in una riga

Vogliamo aggiungere il nome Nina a tutte le persone che hanno per cognome Rasmussen

```
UPDATE Person SET FirstName = 'Nina'
WHERE LastName = 'Rasmussen'
```

Risultato:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Storgt 67	

Aggiornare diversi campi in una riga

Vogliamop cambiare l'indirizzo e il nome della città:

```
UPDATE Person
SET Address = 'Stien 12', City = 'Stavanger'
WHERE LastName = 'Rasmussen'
```

Risultato:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

SQL Delete

Cancellare Righe

L'istruzione DELETE è usata per cancellare le righe di una tabella

```
DELETE FROM table_name WHERE column_name = some_value
```

Person:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

Cancellare una riga

"Nina Rasmussen" deve essere cancellata:

```
DELETE FROM Person WHERE LastName = 'Rasmussen'
```

Risultato

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger

SQL Count

SQL hauna funzione per contare le righe..

Count Function Syntax

La sintassi di count è la seguente:

```
SELECT COUNT(column) FROM table
```

Function COUNT(*)

COUNT(*) conta le righe ritornate da una select.

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

Per avere il numero di righe della tabella:

```
SELECT COUNT(*) FROM Persons
```

Risultato:

```
3
```

Questo esempio conta tutte le persone che hanno un'età maggiore di 20 anni:

```
SELECT COUNT(*) FROM Persons WHERE Age>20
```

Risultato:

```
2
```

Function COUNT(column)

COUNT(column) conta tutte le righe che non hanno NULL nella colonna specificata.

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	

Questo esempio conta tutte le persone che hanno un'età definite nella tabella:

```
SELECT COUNT(Age) FROM Persons
```

Risultato:

```
2
```

COUNT DISTINCT

Note: Il seguente esempio funziona con Oracle e SQl Server non con Access

DISTINCT e COUNT possono essere utilizzate insieme.

La sintassi è:

```
SELECT COUNT(DISTINCT column(s)) FROM table
```

Tabella Orders:

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

L'istruzione :

```
SELECT COUNT(Company) FROM Orders
```

Fornisce:

```
4
```

L'istruzione

```
SELECT COUNT(DISTINCT Company) FROM Orders
```

Ritorna invece

```
3
```

SQL Functions

SQL ha una serie di funzioni predefinite per effettuare calcoli.

Function Syntax

La sintassi per le funzioni è la seguente:

```
SELECT function(column) FROM table
```

Tabella usata negli esempi

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

Function AVG(column)

AVG fornisce la media fra un insieme di valori escludendo NULL.

Esempio

Questo esempio fornisce la media dell'età delle persone nella tabella

```
SELECT AVG(Age) FROM Persons
```

Risultato

```
32.67
```

Function MAX(column)

MAX ritorna ilvalore più alto di un campo (NULL esclusi).

Example

```
SELECT MAX(Age) FROM Persons
```

Result:

```
45
```

Function MIN(column)

MIN ritorna il valore minimo (NULL escluso).

Example

```
SELECT MIN(Age) FROM Persons
```

Result:

```
19
```

Nota: Min e Max Possono essere usati anche per campi testo

Function SUM(column)

Sum ritorna la soma dei valori di un campo.

Esempio

Somma delle età nella tabella Persons

```
SELECT SUM(Age) FROM Persons
```

Risultato:

```
98
```

SQL Group By e Having

Le funzioni aggregate (tipo SUM) spesso necessitano della funzione di raggruppamento GROUP BY.

GROUP BY

La funzione Group BY è stato aggiunto a SQL perché le funzioni aggregate (come SUM) restituiscono il complesso di tutti i valori della colonna ogni volta che sono chiamate.

Senza il GROUP BY, trovare la somma per ogni gruppo specifico dei valori della colonna non era possibile.

La sintassi per GROUP BY è:

```
SELECT column,SUM(column) FROM table GROUP BY column
```

GROUP BY Esempio

La tabella "Sales":

Company	Amount
W3Schools	5500
IBM	4500

W3Schools	7100

E questa istruzione SQL:

```
SELECT Company, SUM(Amount) FROM Sales
```

Resistuisce come risultato:

Company	SUM(Amount)
W3Schools	17100
IBM	17100
W3Schools	17100

Il suddetto codice non è valido perché la colonna restituita non fa parte di una aggragazione. La clausola GROUP BY correggerà questo errore nel seguente modo:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company
```

Resistuisce questo risultato:

Company	SUM(Amount)
W3Schools	12600
IBM	4500

La clausola HAVING

HAVING è stato aggiunto a SQL perché la clausola WHERE chiave non può essere usata in funzioni aggreganti (GROUP BY).

Senza HAVING non sarebbe possible mettere condizioni con il GROUP BY.

La sintassi è:

```
SELECT column,SUM(column) FROM table
GROUP BY column
HAVING SUM(column) condition value
```

Questa tabella:

Company	Amount
W3Schools	5500
IBM	4500
W3Schools	7100

E questa istruzione SQL:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company HAVING SUM(Amount)>10000
```

Producono il seguente risultato

Company	SUM(Amount)
W3Schools	12600

SQL Aliases

Con SQL, possono essere usati aliases per nomi di colonna o tabella.

Column Name Alias

La sintassi è:

```
SELECT column AS column_alias FROM table
```

Table Name Alias

La sintassi è:

```
SELECT column FROM table AS table_alias
```

Esempio

La seguente tabella:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

E questa istruzione SQL:

```
SELECT LastName AS Family, FirstName AS Name
FROM Persons
```

Danno il seguente risultato:

Family	Name
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Esempio

La seguente tabella::

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

E questa istruzione SQL:

```
SELECT LastName, FirstName
FROM Persons AS Employees
```

Danno questo risultato

Tabella Employees:

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

SQL Join

Joins e chiavi

Spesso bisogna selezionare I dati da più tabelle per avere l'informazione che ci interessa.

Le tabelle sono legate fra loro attraverso le chiavi. Una chiave primaria è una colonna con un valore unico per ogni ruga. Lo scopo è quello di legare i dati tra loro, attraverso le tabelle senza dover ripetere i dati.

Nella tabella "Employees", la colonna "Employee_ID" è la chiave primaria, questo significa che non ci sono due righe con lo stesso Employee_ID. Employee_ID distinguee due persone anche se queste hanno nomi uguali.

Nell'esempio sottostante si può notare che:

- La colonna "Employee_ID" è la chiave primaria della tabella "Employees"
- La colonna "Prod_ID" è la chiave primaria della tabella "Orders"
- "Employee_ID" nella tabella "Orders" è usato per fare riferimento alle persone nella tabella "Employees" senza usare i loro nomi

Employees:

Employee_ID	Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

Orders:

Prod_ID	Product	Employee_ID
234	Printer	01
657	Table	03
865	Chair	03

Riferimenti a due tabelle

Si possono ottenere le informazioni da due tabelle nel seguente modo:

Esempio

Chi ha ordinato un prodotto e cosa hanno ordinato?

```
SELECT Employees.Name, Orders.Product
FROM Employees, Orders
WHERE Employees.Employee_ID=Orders.Employee_ID
```

Result

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Example

Chi ha ordinato una stampante?

```
SELECT Employees.Name
FROM Employees, Orders
WHERE Employees.Employee_ID=Orders.Employee_ID
AND Orders.Product='Printer'
```

Result

Name
Hansen, Ola

Usare le join

La join serve per unire le informazioni fra due tabelle

Esempio INNER JOIN

Sintassi

```
SELECT field1, field2, field3
```

```
FROM first_table
INNER JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

Chi ha ordinato prodotti e cosa hanno ordinato?

```
SELECT Employees.Name, Orders.Product
FROM Employees
INNER JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID
```

L'INNER JOIN restituisce tutte le righe di entrambe le tabelle aventi lo stesso Employee_ID, non sono considerati i campi in cui non c'è il matching dell'Employee_id.

Risultato

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Esempio LEFT JOIN

Sintassi

```
SELECT field1, field2, field3
FROM first_table
LEFT JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

Mostra tutti gli impiegati e i relativi ordini (se ci sono).

```
SELECT Employees.Name, Orders.Product
FROM Employees
LEFT JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID
```

Il LEFT JOIN resistituisce tutte le righe della prima tabella anche se nessuna ha un correspettivo nella seconda. Se ci sono impiegati senza ordini questi vengono restituiti ugualmente.

Risultato

Name	Product
Hansen, Ola	Printer
Svendson, Tove	
Svendson, Stephen	Table
Svendson, Stephen	Chair
Pettersen, Kari	

Esempio RIGHT JOIN

Sintassi

```
SELECT field1, field2, field3
FROM first_table
RIGHT JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

Mostra tutti gli ordini e chi li ha ordinati (se possible).

```
SELECT Employees.Name, Orders.Product
FROM Employees
RIGHT JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID
```

Il RIGHT JOIN è l'operazione speculare al LEFT JOIN.

Risultato

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Esempio.

Chi ha ordinato una stampante?

Who ordered a printer?

```
SELECT Employees.Name
FROM Employees
INNER JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID
WHERE Orders.Product = 'Printer'
```

Risultato

Name
Hansen, Ola

Creare Database e Tabelle

Creare un DB

Per creare un DB:

```
CREATE DATABASE database_name
```

Creare una Table

Per creare una tabella in un DB:

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
.......
)
```

Esempio

```
CREATE TABLE Person
(
LastName varchar,
FirstName varchar,
Address varchar,
Age int
)
```

Questo esempio mostra come imporre una lunghezza massima ad una stringa.:

```
CREATE TABLE Person
(
LastName varchar(30),
FirstName varchar,
Address varchar,
Age int(3)
)
```

I Data type in SQL

Data Type	Description
integer(size) int(size) smallint(size) tinyint(size)	Hold integers only. The maximum number of digits are specified in parenthesis.

decimal(size,d) numeric(size,d)	Hold numbers with fractions. The maximum number of digits are specified in "size". The maximum number of digits to the right of the decimal is specified in "d".
char(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis.
varchar(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis.
date(yyyymmdd)	Holds a date

In aggiunta in molti DBMS sono presenti i tipi BLOB (Binary Large Object) per memorizzare file binari.

Creare Indici

Gli indici sonop creati in una tabella per recuperare I dati in maniera più rapida ed efficace. Gli utenti non vedono gli indici.

Nota: Aggiornare una tabella con indici necessita di più tempo, perchè oltre ad aggiornare la tabella vanno aggiornati anche gli indici..

Indice Univoco

Un indice univoco signbifica che due righe non possono avere lo stesso valore nell'indice.

```
CREATE UNIQUE INDEX index_name
ON table_name (column_name)
```

"Column_name" specifica la colonna da indicizzare.

Un indice semplice

Senza UNIQUE I valori possono essere duplicati-

```
CREATE INDEX index_name
ON table_name (column_name)
```

Esempi

```
CREATE INDEX PersonIndex
ON Person (LastName)
```

```
CREATE INDEX PersonIndex
ON Person (LastName DESC)
```

```
CREATE INDEX PersonIndex
ON Person (LastName, FirstName)
```

Cancellare un Index

```
DROP INDEX table_name.index_name
```

Cancellare un Database o una Tabella

Per cancellare un database:

```
DROP DATABASE database_name
```

Per cancellare una tabella:

```
DROP TABLE table_name
```

SQL Alter

Alter Table

ALTER TABLE è usata per aggiungere, modificare o cancellare colonne in una tabella esistente.

```
ALTER TABLE table_name
ADD column_name datatype
ALTER TABLE table_name
DROP COLUMN column_name
```

Person:

LastName	FirstName	Address
Pettersen	Kari	Storgt 20

Example

```
ALTER TABLE Person ADD City varchar(30)
```

Result:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	

Example

```
ALTER TABLE Person DROP COLUMN Address
```

Result:

LastName	FirstName	City
Pettersen	Kari	