

SQL



Antonia Sacchitella

Analyst@icubedsrl

Antonia.sacchitella@icubed.it



Database (Base di Dati)

Dato → informazione.

Database → struttura di **dati** organizzati secondo un modello.

Un DB ha le seguenti caratteristiche:

- Usato per rappresentare/raccogliere dati d'interesse
- Condiviso tra diverse applicazioni software e più utenti
- Ogni dato è rappresentato solo una volta nella collezione

DBMS

Per accedere a uno o più database di usa il DBMS:

Database **M**anagement **S**ystem

Un set di software che permettono l'accesso, l'aggiornamento e eventuale recupero di dati.



Modelli di database

1. Relational

Struttura tramite tabelle composte da campi e record.

Relazioni: interne alla tabella e tra diverse tabelle

->Gestiti da RDBMS

2. Object Oriented

Struttura tramite oggetti, usata soprattutto in ambito documentale (Json, XML..)

->Gestiti da ODBMS / OODBMS

3. Object-Relational

Struttura mista

Database Relational Model

Un archivio è solitamente composto da dati **non omogenei** (ad esempio pensando ad un DB che raccoglie le info di una scuola, i dati –non omogenei- potrebbero essere Libri, Alunni, Professori, Voti, Assenze, ...).

- Ogni gruppo di dati **omogenei** viene registrato all'interno di uno stesso contenitore/struttura detta **tabella**.
- Il singolo elemento inserito in una tabella è detto **record**.
- Le proprietà che caratterizzano ogni singolo elemento della stessa tabella vengono definite **attributi/campi**.

In una rappresentazione tabellare:

- le **righe** rappresentano i **record**
- le **colonne** rappresentano i **campi**.

L'insieme delle descrizioni dei campi (nome, dimensione, tipo ...) prende il nome di struttura della tabella.



Database Relational Model

Altro esempio. Un archivio di dati anagrafici contiene le informazioni sulle persone, con Cognome, Nome, Data di nascita, Città, Telefono.

Cognome	Nome	Data di nascita	Città	Telefono
---------	------	-----------------	-------	----------

Ognuno di questi dati si chiama **campo** e l'insieme dei campi di una stessa riga forma il **record**, che si riferisce ad una singola persona.
L'archivio è quindi un insieme di record.

Database Relational Model

Altro esempio. Un archivio di dati anagrafici contiene le informazioni sulle persone, con Cognome, Nome, Data di nascita, Città, Telefono.

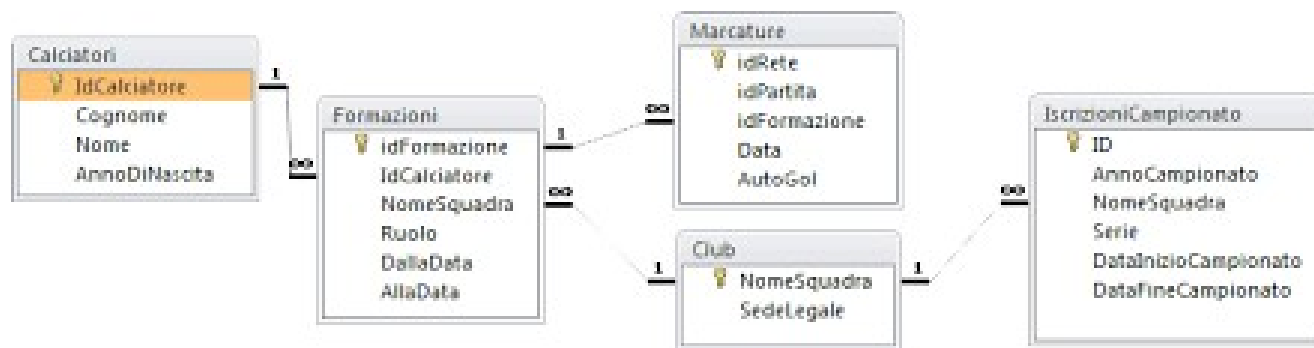
Cognome	Nome	Data di nascita	Città	Telefono
---------	------	-----------------	-------	----------

Ognuno di questi dati si chiama **campo** e l'insieme dei campi di una stessa riga forma il **record**, che si riferisce ad una singola persona.
L'archivio è quindi un insieme di record.

Database Relational Model

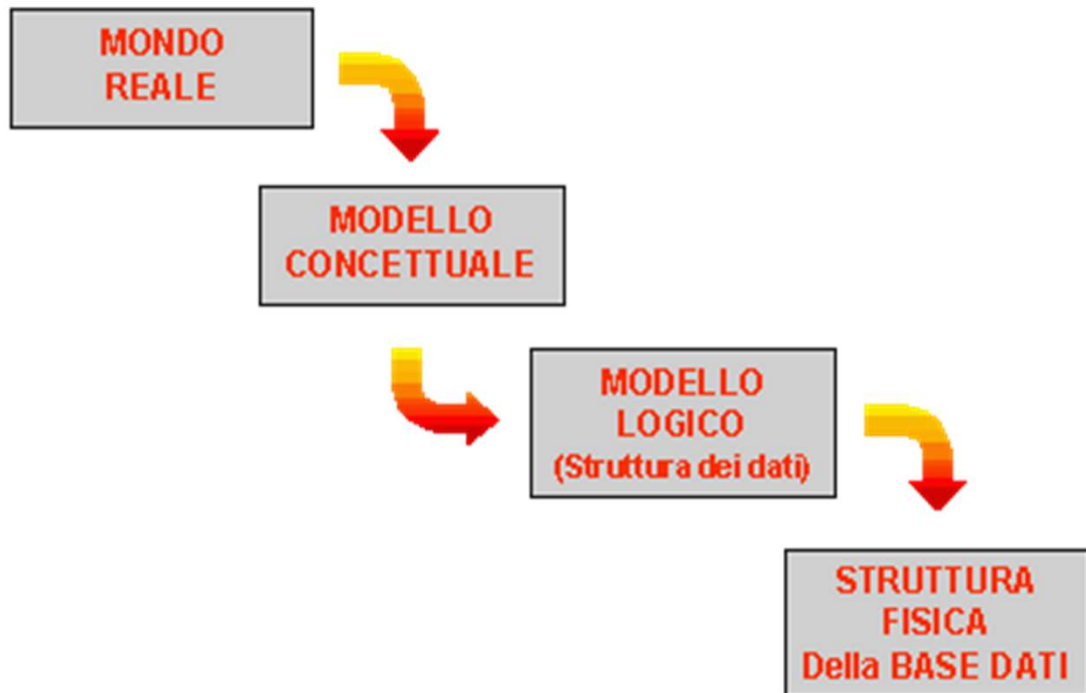
Un DB composto da **diverse tabelle in relazione** tra loro si dice **Relazionale**.

Le relazioni tra le tabelle, permettono di manipolare i dati più facilmente e soprattutto evitano la ridondanza dei dati, ovvero la duplicazione delle informazioni che è inevitabile quando si opera con singole tabelle indipendenti.



Modelli per la fase di Progettazione

- **Concettuale**
- **Logico** (relazionale)
- **Fisico**



Modello Concettuale

Osservando una realtà possiamo coglierne le **entità** utili per rappresentarne la gestione automatizzata.

Ciò si ottiene individuando gli elementi che la caratterizzano: ad esempio in una scuola gli studenti, i docenti, le materie, le prove degli studenti, ecc.

Ciascun'entità possiede degli **attributi**, ovvero le proprietà che la identificano e la caratterizzano.

Modello Concettuale

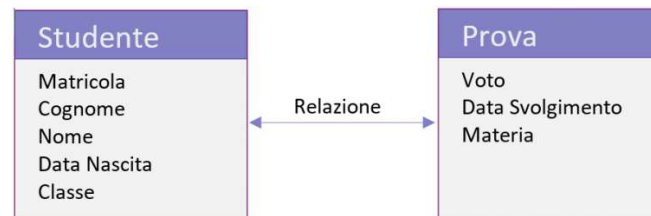
Per esempio, le proprietà (o attributi) dell'entità **Studente** sono la matricola, il cognome, il nome, la data di nascita, la classe.



L'entità **Prova** ha come attributi il voto, la data di svolgimento, la materia a cui si riferisce.



Tra le entità si stabiliscono inoltre delle **relazioni** o associazioni, cioè dei collegamenti. Per conoscere a quale studente si riferiscono le prove fissiamo un collegamento tra l'entità Prova e l'entità Studente.



Definendo le entità, gli attributi e le relazioni si costruisce il modello concettuale della realtà osservata.

Modello Logico

Dal modello precedente si passa al modello logico (o relazionale).

- Ogni **entità** del modello concettuale diventa una **tabella**.
- Gli **attributi** diventano i titoli delle colonne e andranno a formare il tracciato record, cioè l'insieme di tutti gli identificatori dei **campi della tabella**.

Studenti

Matricola	Cognome	Nome	DataNascita	Classe	Tel
0001	Rossi	Laura	15/04/2002	2A	320.5564332
0002	Verdi	Maria	12/08/2001	2A	333.9887001
0003	Bianco	Giorgio	06/01/2002	2A	349.5435672
0004	Neri	Luca	21/12/2001	2A	348.1267887

Prove

ID	Voto	Data	Materia
V001	10	24/03/2018	ITALIANO
V002	9	23/07/2017	MATEMATICA
V003	7	16/01/2018	FRANCESE
V004	9	20/11/2017	INGLESE

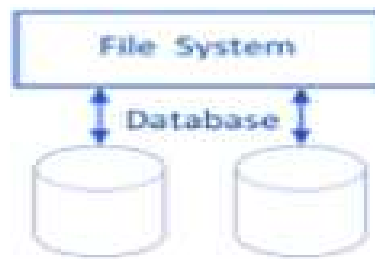
Le righe (o record) contengono i dati che si riferiscono a uno specifico esemplare (o istanza) dell'entità.

Ad esempio, la prima riga della tabella Studenti rappresenta lo studente Laura Rossi.

Modello Fisico

Infine, il modello fisico individua il supporto fisico di memorizzazione da utilizzare per l'archiviazione dei dati (cd-rom, hard-disk,...).

La progettazione fisica coincide con l'associazione della struttura logica ad una struttura fisica per la memorizzazione di massa.



Progettazione del Database

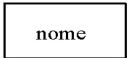


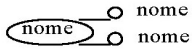




Modello Entità-Relazione

Il modello Entità-Relazione (E-R) è un **modello concettuale** di dati, e come tale fornisce una serie di strutture (costrutti), atte a descrivere la realtà in una maniera facile da comprendere

Rappresentazione
concettuale della struttura
dei dati

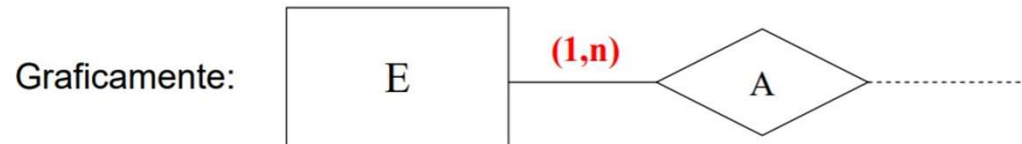
- Costrutti hanno una rappresentazione con diagramma
- Modello più leggibile e comprensibile

STRUTTURA DI CLASSIFICAZIONE	SIMBOLO
ENTITA'	
RELAZIONE	
ATTRIBUTO SEMPLICE	
ATTRIBUTO AGGREGATO	
SOTTOINSIEME	
GENERALIZZAZIONE	

Cardinalità di Relazione

Per ogni entità partecipante ad una relazione viene specificata una **cardinalità di relazione**.

Essa è una **coppia di numeri naturali** che specifica il **numero minimo e massimo** di istanze di relazione → (min-card,max-card)



Ad esempio, se i vincoli di cardinalità per un'entità E relativamente a un'associazione A sono (1,n) questo significa:

- ogni istanza di E partecipa almeno ad una istanza di A → min-card = 1
- ogni istanza di E può partecipare a più istanze di A → max-card = n

N.B. Con la costante n si indica un numero generico maggiore di 1 quando la cardinalità non è nota con precisione.

Cardinalità di Relazione



Cardinalità di relazione (Persone,Proprietà) \rightarrow (0,n)

min-card = 0: esistono persone che non posseggono alcuna automobile

max-card = n: ogni persona può essere proprietaria di molte (n) automobili

Cardinalità di relazione (Automobili,Proprietà) \rightarrow (0,1)

min-card = 0: esistono automobili non possedute da alcuna persona

max-card = 1: ogni automobile può avere al più un proprietario

Tipi di associazione: terminologia

Nel caso di un'associazione binaria A tra due entità $E1$ ed $E2$ (non necessariamente distinte), si dice che:

- A è **uno a uno** se le cardinalità massime di entrambe le entità rispetto ad A sono 1
- A è **uno a molti** se $\max\text{-card}(E1,A) = 1$ e $\max\text{-card}(E2,A) = n$, o viceversa
- A è **molti a molti** se $\max\text{-card}(E1,A) = n$ e $\max\text{-card}(E2,A) = n$

Si dice inoltre che:

La partecipazione di $E1$ in A è *opzionale* se $\min\text{-card}(E1,A) = 0$

La partecipazione di $E1$ in A è *obbligatoria* (o totale) se $\min\text{-card}(E1,A) = 1$

Relazione uno a uno

La relazione **uno a uno** è detta anche **biunivoca** perché ad ogni elemento della prima entità, fa corrispondere un solo, specifico, elemento dell'entità collegata.

Esempi:

- A ciascun marito, corrisponde una sola e specifica moglie.
- A ciascuna persona corrisponde una sola carta di identità.

NB. Ha senso parlare di relazione 1:1 solo se entrambe le entità collegate sono entità a tutti gli effetti.

Altrimenti la relazione «non esiste» ma si traduce nell'inserire un attributo in più nell'entità di partenza.

Ad esempio: ad ogni persona corrisponde un solo Codice Fiscale. Il codice fiscale non è un'entità vera e propria quindi diventa un attributo dell'entità persona.

Relazione uno a molti

La relazione **uno a molti** fa corrispondere:

- a ciascun elemento della prima entità, **uno o più elementi** della seconda entità.
- ad ogni elemento della seconda entità, **un solo e specifico elemento** della prima entità.

Esempio ***Studente - Valutazione***:

- per ogni studente possiamo avere più valutazioni (voto di Storia di novembre; voto di Matematica di ottobre; voto di Italiano di gennaio; voto di Italiano di febbraio...),
- a ciascuna valutazione (personale), corrisponde il solo e specifico Studente che l'ha presa.

Relazione molti a molti

La relazione **molti a molti** invece fa corrispondere:

- ad un elemento della prima entità, tanti elementi della seconda entità;
- a ciascun elemento della seconda entità, fanno capo tanti elementi della prima entità.

Ad esempio: Ogni studente ha più Docenti e ogni Docente ha più Studenti.

NOTA: Poiché la relazione di tipo molti a molti è riconducibile, attraverso un artificio, ad una combinazione di relazioni uno a molti, ci focalizzeremo soprattutto sulle relazioni 1 a molti!

Chiavi e Integrità Referenziale

Tutte le tabelle hanno un campo

Chiave Primaria (PK: Primary Key)



codice alfanumerico o un numero identificativo (ID) per distinguere ciascuna riga all'interno della tabella.

La chiave primaria di una tabella è un campo (obbligatorio) del tracciato record i cui valori identificano **univocamente** ciascun singolo record della tabella, in modo che non possano esistere due o più record della tabella con la stessa chiave primaria.

(Es. Per l'entità Studente la PK è Matricola)

Per stabilire poi i **collegamenti tra le tabelle** occorre aggiungere le **Chiavi Esterne!**

La **Chiave Esterna (FK: Foreign Key)** di una tabella è un campo del tracciato record che può ammettere valori duplicati, ma che invece è chiave primaria di un'altra tabella alla quale ci si vuole relazionare logicamente.

I record di due tabelle si mettono in relazione attraverso la coppia di campi chiave primaria/chiave esterna.

Chiavi e Integrità Referenziale

Quando si mettono in relazione le tabelle, è possibile applicare all'associazione una particolare proprietà detta **integrità referenziale** che permette di rendere più forte il legame tra i record delle tabelle collegate.

L'**integrità referenziale** è una regola applicata ai valori che può assumere la chiave esterna, in modo da assicurare che i valori che questa assumerà siano sempre riferiti a quelli del campo chiave primaria in relazione.

In altre parole, l'integrità referenziale impone che **ogni inserimento di un valore della chiave esterna debba avere un valore corrispondente della chiave primaria associata** nella relazione.

Esempio Pk e Fk

La tabella dei Prodotti ha due campi aggiuntivi che rappresentano i collegamenti al codice della categoria e al codice del fornitore.

Le **tabelle** saranno così definite:

Categorie: (ID, NomeCategoria, Descrizione)

Fornitori: (CodForn, NomeSocietà, Città, Telefono)

Prodotti: (CodProdotto, NomeProdotto, Prezzo, *CodFornitore*, *IDCategoria*)

dove le chiavi primarie vengono sottolineate e le chiavi esterne sono indicate in corsivo.



 ID	NomeCategoria	Descrizione
1	Bevande	Bibite analcoliche, tè, caffè, birra
2	Dolci	Pasticceria fresca, Biscotti
3	Salumeria	Affettati, Salami, Wrustel
4	Latticini	Formaggi



 CodProdotto	NomeProdotto	Prezzo	<i>CodFornitore</i>	<i>IDCategoria</i>
100	Tè verde	€ 5	3	1
220	Tiramisù	€ 6	2	2
314	Fontina	€ 12	1	4
514	Toma	€ 7	1	4



 CodForn	NomeSocietà	Città	Telefono
1	La Pastora	NA	320 5564332
2	Dolcezze	RM	333 9887001
3	Drinking	PA	349 5435672
4	Altissima	TO	348 1267887

Riassumendo:

Quali sono gli step per creare un modello concettuale?

1. Identificare tutte le entità del sistema. Un'entità dovrebbe apparire una sola volta in un particolare diagramma.
2. Aggiungere gli attributi per le entità.
3. Identificare le relazioni tra le entità. Collegarli utilizzando una linea e aggiungere un diamante al centro che descriva il rapporto.
4. Specificare le cardinalità di relazione

Riassumendo:

Il modello E/R è un **modello concettuale** molto utilizzato per la progettazione di basi di dati.

- Esistono molti **dialetti** E/R, che spesso si differenziano solo per la notazione grafica adottata
- I principali costrutti del modello sono **l'entità, l'associazione e l'attributo**, a cui si aggiungono **identificatori, gerarchie e vincoli di cardinalità**

N.B. L'espressività del modello E/R non è normalmente sufficiente in fase di progettazione, il che comporta la necessità di documentazione di supporto

Modello Entità-Relazione

- Leggere con attenzione i requisiti
- Disegnare all'inizio solo le entità individuate nei requisiti
- Aggiungere relazioni ed attributi tra le entità
- Specificare entità per entità le cardinalità delle relazioni
- Passaggio al modello logico con traduzione delle relazioni

Demo

Progettare una base dati per gestire Attori, Film, Sale, Prenotazioni e Programmazione.

I Film sono identificati da un codice e di essi interessano anche il titolo, il genere, la durata e gli attori che vi recitano.

Gli Attori sono identificati da un codice e di essi interessano anche il nome, nazionalità, la data di nascita ed il cachet percepito.

La Sala è identificata dal nome, un numero univoco e dal numero di posti a sedere. La Programmazione tiene traccia del Film trasmesso in una determinata Sala e viene caratterizzata dall'ora e dal numero di posti disponibili.

Infine della Prenotazione si vuole tenere traccia dei posti da prenotare, della programmazione e dell'email di chi prenota.



Modello ER

Classe

Cinema



Esercitazione

Realizzare il modello concettuale e di Entità - Relazione di un database di Romanzi.

I romanzi sono caratterizzati da:

Titolo (MAX 20 Caratteri) e Anno di pubblicazione

In ogni romanzo ci può essere più di un Personaggio caratterizzato da:

Nome, Sesso e Ruolo

L'autore (unico) di ciascun romanzo è caratterizzato da:

Nome, Anno di Nascita, Anno di morte (opzionale), Nazione

Identificare opportunamente le relazioni che intercorrono tra le entità ed i vincoli presenti nella descrizione del modello.



SQL e Database

Le principali operazioni che si possono effettuare sul database sono:

- **creazione** dell'archivio sul supporto → DDL
- **manipolazione** delle informazioni contenute (inserimento, variazione, cancellazione) → MDL
- **consultazione** e interrogazione dell'archivio fornendo i risultati con visualizzazioni o stampe → Query

SQL – Cos'è

Structured Query Language

è un linguaggio standard per l'accesso e la manipolazione di database.

SQL – a cosa serve

- SQL può eseguire query su un database
- SQL può recuperare i dati da un database
- SQL può inserire record in un database
- SQL può aggiornare i record in un database
- SQL può eliminare i record da un database

SQL – a cosa serve

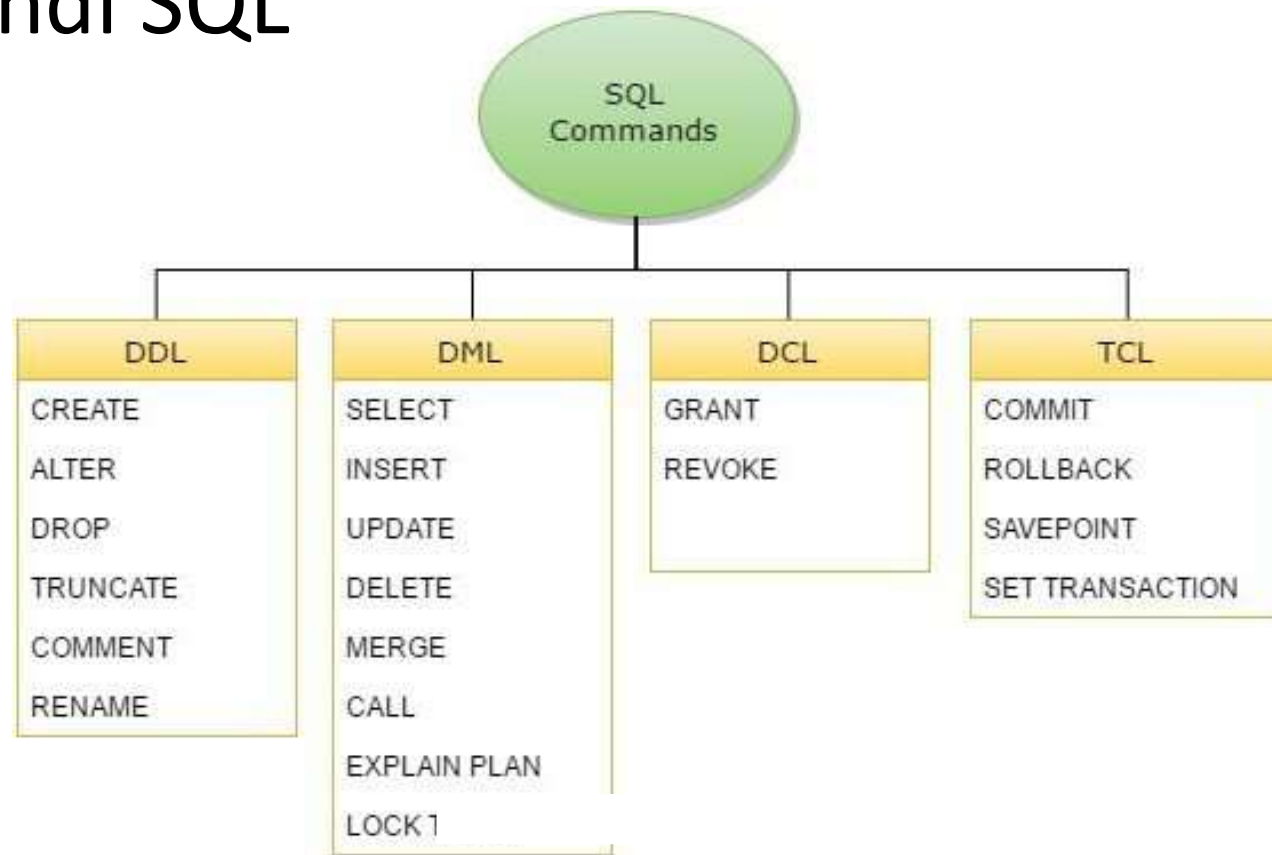
- SQL può creare nuovi database
- SQL può creare nuove tabelle in un database
- SQL può creare stored procedure in un database
- SQL può creare viste in un database
- SQL può impostare autorizzazioni per tabelle, procedure e viste

SQL è uno standard, ma ...

Sebbene SQL sia uno standard ANSI / ISO, esistono diverse versioni del linguaggio SQL.

Tuttavia, per essere conformi allo standard ANSI, supportano tutti almeno i comandi principali in modo simile.

Comandi SQL



SQL - Data Definition Language

Data Definition Language (DDL) si occupa di schemi e descrizioni di database, di come i dati dovrebbero risiedere nel database.

- CREATE: creare il database e i suoi oggetti (tabelle, indici, viste, procedura di memorizzazione, funzioni e trigger)
- ALTER: modifica la struttura del database esistente
- DROP: elimina gli oggetti dal database
- TRUNCATE: rimuove tutti i record da una tabella, inclusi tutti gli spazi allocati per i record
- COMMENT: aggiungi commenti al dizionario dei dati
- RENAME: rinomina un oggetto

SQL - Data Manipulation Language

Data Manipulation Language (DML) si occupa della manipolazione dei dati e include le istruzioni SQL più comuni come SELECT, INSERT, UPDATE, DELETE ecc. E viene utilizzato per archiviare, modificare, recuperare, eliminare e aggiornare i dati nel database.

- SELECT: recupera i dati da un database
- INSERT: inserire i dati in una tabella
- UPDATE: aggiorna i dati esistenti all'interno di una tabella
- DELETE - Elimina tutti i record da una tabella del database
- MERGE - Funzionamento UPSERT (inserire o aggiornare)
- CALL: chiama un sottoprogramma PL / SQL o Java
- EXPLAIN PLAN - interpretazione del percorso di accesso ai dati
- LOCK TABLE - controllo della concorrenza

SQL - Data Control Language

Data Control Language (DCL) include comandi come GRANT e riguarda principalmente diritti, autorizzazioni e altri controlli del sistema di database.

- GRANT: consente agli utenti di accedere ai privilegi del database
- REVOKE: revoca agli utenti i privilegi di accesso forniti utilizzando il comando GRANT

SQL - Transaction Control Language

Transaction Control Language (TCL) si occupa delle transazioni all'interno di un database.

COMMIT: commette una transazione

ROLLBACK: rollback di una transazione in caso di errore

SAVEPOINT: per ripristinare i punti di transazione all'interno dei gruppi

SET TRANSACTION: specifica le caratteristiche per la transazione

SQL Data Types (alcuni)

Data Type	Definition
NCHAR(N)	Stringa con lunghezza fissa N
NVARCHAR(N)	Stringa di lunghezza variabile
BIT	0-1
INT	Numeri non decimali
DECIMAL(p,s)	Numeri decimali formati da p cifre di cui s decimali
DATE	Data
TIME	Orario
MONEY	Valuta

Demo

Accedere a SQL Server



CREATE

Il comando CREATE in SQL viene utilizzato per gestire la creazione delle strutture del database.

COMANDO
CREATE DATABASE
CREATE TYPE
CREATE TABLE
CREATE INDEX
CREATE VIEW
CREATE TRIGGER
CREATE PROCEDURE
CREATE USER

CREATE DATABASE

Sintassi completa di
creazione del database



```
CREATE DATABASE database_name
[ CONTAINMENT = { NONE | PARTIAL } ]
[ ON
    [ PRIMARY ] <filespec> [ ,...n ]
    [ , <filegroup> [ ,...n ] ]
    [ LOG ON <filespec> [ ,...n ] ]
]
[ COLLATE collation_name ]
[ WITH <option> [ ,...n ] ]
[;]

<option> ::=
{
    FILESTREAM ( <filestream_option> [ ,...n ] )
    | DEFAULT_FULLTEXT_LANGUAGE = { lcid | language_name | language_alias }
    | DEFAULT_LANGUAGE = { lcid | language_name | language_alias }
    | NESTED_TRIGGERS = { OFF | ON }
    | TRANSFORM_NOISE_WORDS = { OFF | ON }
    | TWO_DIGIT_YEAR_CUTOFF = <two_digit_year_cutoff>
    | DB_CHAINING { OFF | ON }
    | TRUSTWORTHY { OFF | ON }
    | PERSISTENT_LOG_BUFFER=ON ( DIRECTORY_NAME='<Filepath to folder on DAX formatted volume>' )
}

<filestream_option> ::=
{
    NON_TRANSACTED_ACCESS = { OFF | READ_ONLY | FULL }
    | DIRECTORY_NAME = 'directory_name'
}

<filespec> ::=
{
    (
        NAME = logical_file_name ,
        FILENAME = { 'os_file_name' | 'filestream_path' }
        [ , SIZE = size [ KB | MB | GB | TB ] ]
        [ , MAXSIZE = { max_size [ KB | MB | GB | TB ] | UNLIMITED } ]
        [ , FILEGROWTH = growth_increment [ KB | MB | GB | TB | % ] ]
    )
}

<filegroup> ::=
{
    FILEGROUP filegroup_name [ [ CONTAINS FILESTREAM ] [ DEFAULT ] | CONTAINS MEMORY_OPTIMIZED_DATA ]
    <filespec> [ ,...n ]
}
```

CREATE

Questo comando si riduce



`CREATE DATABASE` nome_database

CREATE TABLE

CREATE TABLE

```
[ database_name . [ schema_name ] . | schema_name . ] table_name
[ AS FileTable ]
( { <column_definition> | <computed_column_definition>
  | <column_set_definition> | [ <table_constraint> ] [ ,...n ] } )
[ ON { partition_scheme_name ( partition_column_name ) | filegroup
  | "default" } ]
[ { TEXTIMAGE_ON { filegroup | "default" } } ]
[ FILESTREAM_ON { partition_scheme_name | filegroup
  | "default" } ]
[ WITH ( <table_option> [ ,...n ] ) ]
[ ; ]
```

```
CREATE TABLE table_name
(
column_name1 data_type(size),
column_name2 data_type(size),
column_name3 data_type(size),
....
);
```

Solitamente si devono costruire specificando anche chiavi primarie, esterne, vincoli interni ed esterni; ma è anche possibile modificare la struttura della tabella in seguito, variando campi e vincoli.

CREATE TABLE

```
CREATE TABLE dbo.Esempio
(
    Id            INT PRIMARY KEY IDENTITY(1, 1),
    Campo1        NVARCHAR(30) NULL,
    Campo2        NVARCHAR(30) NOT NULL,
    Campo3        NVARCHAR(30) NOT NULL DEFAULT 'my_default',
    Campo4        INT NULL,
    Campo5        INT NOT NULL,
    Campo6        INT NOT NULL DEFAULT 0
)
```

Le **proprietà/vicoli** principali sono:

- **NULL**: il campo può assumere il valore NULL.
- **NOT NULL**: il campo non può assumere il valore NULL.
- **NOT NULL DEFAULT X**: il campo non può assumere il valore NULL; se non impostato nella query di inserimento allora assume il valore indicato dopo la parola 'DEFAULT'.
- **PRIMARY KEY**: il campo è la chiave primaria della tabella.
- **IDENTITY(N, M)**: il campo è generato dal database a partire dal valore N e incrementato ogni volta del valore M.

ALTER TABLE

Il comando ALTER in SQL viene utilizzato per modificare la struttura di una tabella esistente nel database.

Viene utilizzato quindi per aggiungere, modificare o eliminare colonne o vincoli in una tabella esistente.

- **Aggiungere una nuova colonna in una tabella esistente**

Sintassi `ALTER TABLE Table_Name ADD New_Column_Name Data_Type (Size);`

- **Modificare il tipo di dati e le dimensioni di una colonna esistente**

Sintassi `ALTER TABLE Table_Name ALTER COLUMN Column_Name New_Data_Type`
(New_Size)

- **Eliminare la colonna esistente da una tabella:**

Sintassi `ALTER TABLE Table_Name DROP COLUMN Column_Name`

Esempio

Il seguente comando Crea creerà una nuova tabella 'Impiegato' nel database.



```
CREATE TABLE Employee
(
    Id INT PRIMARY KEY,
    Name VARCHAR(50),
    Salary DECIMAL(18, 2)
);
```

La seguente istruzione Alter **aggiungerà** una nuova colonna nella tabella "Impiegato".



```
ALTER TABLE Employee ADD City VARCHAR(20)
```

La seguente istruzione Alter viene utilizzata per **modificare** il tipo di dati e la dimensione di una colonna esistente "Città".



```
ALTER TABLE Employee ALTER COLUMN City NVARCHAR (100);
```

La seguente istruzione Alter viene utilizzata per **eliminare** la colonna esistente "Città" dalla tabella.



```
ALTER TABLE Employee DROP COLUMN City;
```

Vincoli SQL

I vincoli SQL vengono utilizzati per specificare le **regole** per i dati in una tabella quindi per limitare il tipo di dati che possono essere inseriti in quella tabella.

Ciò garantisce **l'accuratezza e l'affidabilità dei dati** nella tabella.

In caso di violazione tra il vincolo e l'azione sui dati, l'azione viene interrotta.

I vincoli possono essere:

- a livello di colonna → si applicano a una colonna e i vincoli
- a livello di tabella → si applicano all'intera tabella.

I vincoli più usati in SQL:

- [NOT NULL](#) - Assicura che una colonna non possa avere un valore NULL
- [UNIQUE](#) - Assicura che tutti i valori in una colonna siano diversi
- [PRIMARY KEY](#) - Combinazione di a NOT NULL e UNIQUE. Identifica in modo univoco ogni riga in una tabella
- [FOREIGN KEY](#) - Impedisce azioni che distruggerebbero i collegamenti tra le tabelle
- [CHECK](#) - Assicura che i valori in una colonna soddisfino una condizione specifica
- [DEFAULT](#) - Imposta un valore predefinito per una colonna se non viene specificato alcun valore
- [CREATE INDEX](#) - Utilizzato per creare e recuperare dati dal database molto rapidamente

Primary Key

Sintassi

```
-- Column level Primary key
CREATE TABLE TableName
(
  Column1 data_type [NOT NULL ] [ PRIMARY KEY ],
  Column2 data_type [ NULL | NOT NULL ],
  Column3 ...
);

-----

-- Table level Primary key
CREATE TABLE TableName
(
  Column1 data_type [ NULL | NOT NULL ],
  Column2 datatype [ NULL | NOT NULL ],
  Column3 ...
  CONSTRAINT ConstraintName PRIMARY KEY (Column1, Column2)
);
```

Esempio pratico:

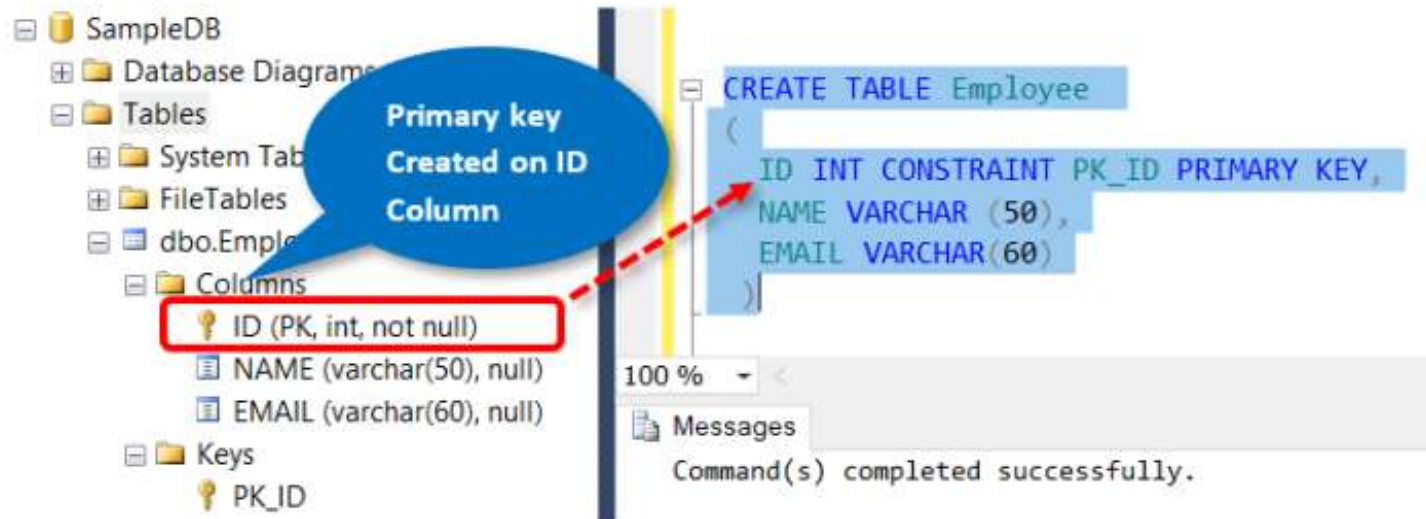
```
-- Column level Primary key
CREATE TABLE Employee
(
  ID INT CONSTRAINT PK_ID PRIMARY KEY,
  NAME VARCHAR (50),
  EMAIL VARCHAR(60)
)

-- OR

-- Table level Primary key
CREATE TABLE Employee
(
  ID INT NOT NULL,
  NAME VARCHAR (50),
  EMAIL VARCHAR(60)
  CONSTRAINT PK_ID PRIMARY KEY(ID)
)
```

Primary Key

Da interfaccia:



Primary Key composta

La chiave primaria costituita da più colonne o campi è nota come **chiave primaria composta**.

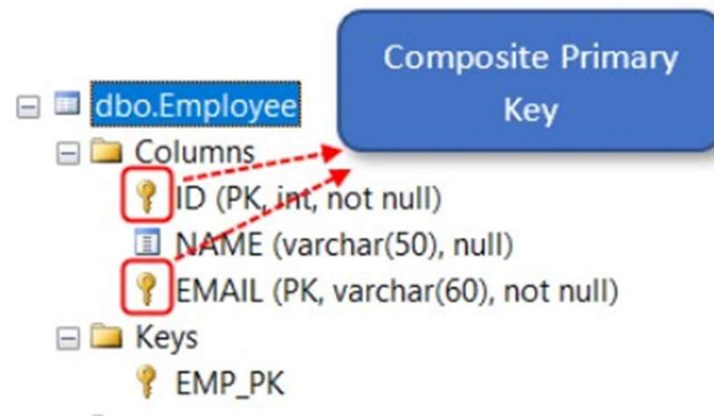
Nell'esempio seguente, creeremo una chiave primaria composta su più colonne come ID ed EMAIL.

Nota: Le 2 colonne devono essere entrambe NOT NULL altrimenti si avrà un errore durante la

```
-- Create composite primary key
```

```
ALTER TABLE Employee ADD CONSTRAINT EMP_PK PRIMARY KEY (ID, EMAIL);
```

Da Interfaccia:



Foreign Key

La **FK (Foreign Key)** è un campo (o una raccolta di campi) di una tabella, che fa riferimento a una PK (Primary Key) di un'altra tabella.

- La tabella con la chiave esterna FK è chiamata tabella **figlio**
- La tabella con la chiave primaria PK è chiamata tabella **referenziata o padre**.

Sintassi creazione di una FK
in fase di creazione di una
nuova tabella:

```
CREATE TABLE Orders (  
    OrderID int NOT NULL PRIMARY KEY,  
    OrderNumber int NOT NULL,  
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)  
);
```

per definire una FK
eventualmente anche su più
colonne

```
CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)  
    REFERENCES Persons(PersonID)  
);
```

Foreign Key

Sintassi creazione di una FK
attraverso la modifica di una
tabella già esistente:

per definire una FK
eventualmente anche su più
colonne



```
ALTER TABLE Orders  
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```



```
ALTER TABLE Orders  
ADD CONSTRAINT FK_PersonOrder  
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

Violazione Vincoli di Integrità

```
CREATE TABLE Tabella (  
    . . . . . ,  
    Campo_FK Tipo Vincolo,  
    CONSTRAINT [Nome_Vincolo]  
    FOREIGN KEY (Campo_FK) REFERENCES AltraTabella (AltroCampo)  
        ON DELETE NO ACTION | CASCADE | SET NULL | SET DEFAULT  
        ON UPDATE NO ACTION | CASCADE | SET NULL | SET DEFAULT  
    . . . . .  
);
```

dopo aver specificato la chiave esterna è possibile indicare uno o due clausole di reazione:

- **ON DELETE**, che viene attivata nel caso sia cancellata una riga dalla tabella primaria
- **ON UPDATE**, che viene attivata nel caso sia modificato il valore della chiave primaria in una riga della tabella primaria

Violazione Vincoli di Integrità

Per ciascuna delle due clausole è possibile scegliere uno tra tre possibili eventi:

NO ACTION, significa che il comando è vietato e quindi la cancellazione o la modifica nella tabella primaria non deve avere effetti. È l'evento di default.

CASCADE, significa che le righe della tabella secondaria subiscono la stessa sorte di quelle della tabella primaria (ovvero sono a loro volta cancellate o modificate)

SET NULL, significa che nel campo chiave esterna delle righe correlate si impone il valore nullo. Questa opzione è ammissibile solo se la chiave esterna non sia obbligatoria (not null), altrimenti equivale a Non ACTION.

SET DEFAULT, significa che nel campo chiave esterna delle righe correlate si impone il valore di base, indicato dalla CREATE TABLE.

UNIQUE

Il vincolo UNIQUE garantisce che tutti i valori in una colonna siano diversi.

Entrambi i vincoli UNIQUE e PRIMARY KEY forniscono una garanzia di unicità per una colonna o un insieme di colonne.

Un vincolo PRIMARY KEY ha automaticamente un vincolo UNIQUE.

Tuttavia, puoi avere molti vincoli UNIQUE per tabella, ma solo una PRIMARY KEY per tabella.

```
CREATE TABLE Persons (  
    ID int NOT NULL UNIQUE,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int  
);
```

```
ALTER TABLE Persons  
ADD UNIQUE (ID);
```

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
```

```
ALTER TABLE Persons  
ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

CONTROLLI (CHECK)

Il vincolo CHECK serve per compiere un controllo come la verifica se il valore è uguale ad un certo valore, oppure è compreso in un intervallo o in un elenco.

```
CREATE TABLE NomeTabella (  
    . . . . . ,  
    [Campo1] Tipo Vincolo,  
    [Campo2] Tipo Vincolo,  
    [Campo3] Tipo Vincolo,  
    CONSTRAINT [Nome_Vincolo_1]  
        CHECK ( condizione)  
)
```

I vincoli CHECK possono anche essere imposti dopo la creazione della tabella usando (l'alter table add constraint).

Demo

Creazione di un Database



Esercitazione n.1

Si deve progettare la base di dati di una applicazione relativa ad una agenzia di viaggio che organizza gite turistiche.

Ogni gita turistica ha un responsabile (di cui si conosce solo nome, cognome e contatto telefonico), una data di partenza, un elenco di partecipanti e fa riferimento ad uno specifico itinerario.

Per ogni partecipante bisogna riportare nome, cognome e data di nascita, città ed indirizzo.

Di ogni itinerario si vuole memorizzare una durata ed il prezzo.

Descrivere il modello concettuale mediante lo schema grafico E/R



SELECT

L'istruzione *SELECT* viene utilizzata per selezionare i dati da un database.

Gli operatori *UNION*, *EXCEPT* e *INTERSECT* possono essere utilizzati per combinare i risultati in un set di risultati.

```
SELECT *  
FROM table_name;
```

```
SELECT column1, column2, ...  
FROM table_name;
```

```
SELECT DISTINCT column1, ...  
FROM table_name;
```

INSERT

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

L'istruzione INSERT aggiunge una o più righe a una tabella.

È anche possibile inserire dati solo in colonne specifiche.

UPDATE

```
UPDATE table_name SET column1 = value1, ...
```

```
UPDATE table_name SET column1 = value1, ...  
WHERE condition;
```

L'istruzione UPDATE viene utilizzata per modificare i record esistenti in una tabella.

Nota: fare attenzione quando si aggiornano i record in una tabella! La clausola WHERE specifica quali record devono essere aggiornati. Se si omette la clausola WHERE, tutti i record nella tabella verranno aggiornati!

DELETE

```
DELETE FROM table_name;
```

```
DELETE FROM table_name WHERE condition;
```

L'istruzione UPDATE viene utilizzata per eliminare record esistenti in una tabella.

Nota: fare attenzione quando si eliminano i record in una tabella! La clausola WHERE specifica quali record devono essere eliminati. Se si omette la clausola WHERE, tutti i record nella tabella verranno eliminati!

Demo

Inserire, aggiornare e eliminare dati
Select



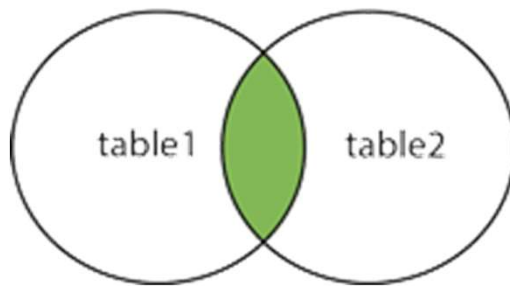
JOIN

Una clausola JOIN viene utilizzata per combinare righe da due o più tabelle, in base a una colonna correlata tra loro.

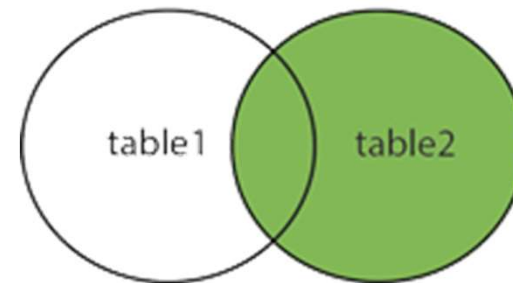
```
SELECT column1, column2, ...  
      FROM table1  
INNER [ LEFT / RIGHT / FULL OUTER ] JOIN table2  
      ON table1.column_name = table2.column_name;
```

JOIN

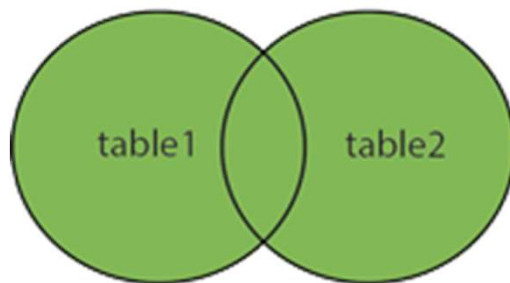
INNER JOIN



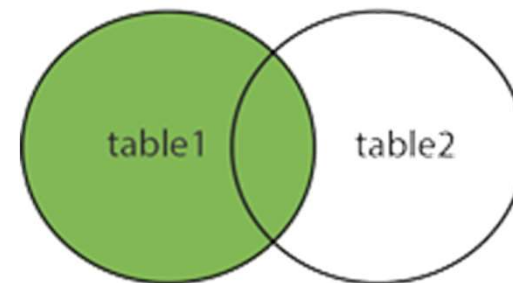
RIGHT JOIN



FULL OUTER JOIN



LEFT JOIN



GROUP BY

L'istruzione GROUP BY raggruppa righe con gli stessi valori in righe di riepilogo, ad esempio "trova il numero di clienti in ciascun paese".

```
SELECT column1, ...  
FROM table_name  
GROUP BY column_name
```

L'istruzione GROUP BY viene spesso utilizzata con funzioni aggregate (COUNT, MAX, MIN, SUM, AVG) per raggruppare il set di risultati per una o più colonne.

GROUP BY

- **MIN()** restituisce il valore più piccolo della colonna selezionata
- **MAX()** restituisce il valore più grande della colonna selezionata
- **COUNT()** restituisce il numero di righe che corrisponde a un criterio specificato

```
SELECT MIN(column1)  
FROM table_name  
GROUP BY column_name
```

```
SELECT MAX(column1)  
FROM table_name  
GROUP BY column_name
```

```
SELECT COUNT(column1)  
FROM table_name  
GROUP BY column_name
```

GROUP BY

- **AVG()** restituisce il valore medio di una colonna numerica
- **SUM()** restituisce la somma totale di una colonna numerica

```
SELECT AVG(column1)  
FROM table_name  
GROUP BY column_name
```

```
SELECT SUM(column1)  
FROM table_name  
GROUP BY column_name
```

HAVING

La clausola HAVING è stata aggiunta a SQL perché non è possibile utilizzare la parola chiave WHERE con le funzioni di aggregazione.

```
SELECT column1, MAX(column2), ...  
      FROM table_name  
      GROUP BY column1  
      HAVING COUNT(column2) > 0
```


ORDER BY

La parola chiave ORDER BY viene utilizzata per ordinare il set di risultati in ordine crescente o decrescente.

La parola chiave ORDER BY ordina i record in ordine crescente per impostazione predefinita.

Per ordinare i record in ordine decrescente, utilizzare la parola chiave DESC.

```
SELECT column1, column2, ...  
      FROM table1  
ORDER BY column1, column2 DESC;
```

IN

L'operatore IN consente di specificare più valori in una clausola WHERE.

È una scorciatoia per sostituire più condizioni OR.

```
SELECT column1, column2, ...  
FROM table1  
WHERE column_name = value1 OR column_name = value2 OR ...;
```



```
SELECT column1, column2, ...  
FROM table1  
WHERE column_name IN (value1, value2, ...);
```

BETWEEN

L'operatore BETWEEN seleziona i valori all'interno di un determinato intervallo. I valori possono essere numeri, testo o date.

È un operatore inclusivo, ovvero i valori di inizio e fine sono inclusi nell'intervallo.

```
SELECT column1, column2, ...  
      FROM table1  
WHERE column_name BETWEEN value1 AND value2;
```

Demo

Join

Order By / Group by

Having



Esercitazione n.3

Scrivere le seguenti query SQL

- 1- Il titolo dei romanzi del 19° secolo
- 2- Il titolo, l'autore e l'anno di pubblicazione dei romanzi di autori russi, ordinati per autore e, per lo stesso autore, ordinati per anno di pubblicazione
- 3- I personaggi principali (ruolo ="Protagonista") dei romanzi.
- 4- Per ogni autore italiano, l'anno del suo primo e dell'ultimo romanzo.
- 5- I nomi dei personaggi che compaiono in più di un romanzo, ed il numero dei romanzi nei quali compaiono
- 6- Visualizzare tutti i romanzi contemporanei distinti per titolo

