Database

Un database è un archivio strutturato di dati

Un DBMS (database management system) è un software per gestire database

Lo scopo di un database è memorizzare dati e cercare all’interno dei dati

I DBMS si differenziano per diverse caratteristiche, le principali sono:

* Limite di dimensione dei database
* Efficienza nella gestione dei dati
* Costo della licenza

I DBMS si dividono in:

* Server based
* File based

Oppure in:

* Relazionali
* Non relazionabili

Tabelle

I dati all’interno dei database sono memorizzati in tabelle

Una tabella è un contenitore strutturato, organizzato in campi

Un campo è una “colonna” di una tabella, definisce la struttura

un record è una “riga” di una tabella, definisce il contenuto

un campo è caratterizzato da un nome, un tipo di dato e una serie di vincoli

heidiSQL

heidiSQL è un client per MySQL/mariaDB

il tipo di dato è un vincolo implicito: limita la possibilità di inserire dati all’interno del campo

null è un valore speciale che indica che il campo non ha nessun valore, questo consente di distinguere 2 casi:

* L’utente non ha inserito il dato (valore null)
* L’utente ha inserito un testo vuoto (stringa vuota, non null)

“permetti valori null” (not null) è un vincolo esplicito: limita limita la possibilità di inserire dati all’interno del campo

I vincoli consentono di limitare gli errori di inserimento, garantendo un insieme di dati il più possibile corretto e affidabile

Backup

Un backup è una copia dei dati fatta per motivi di sicurezza o distribuzione

Tutti i DBMS consentono di creare backup di uno o più database su file, per poi ripristinarli quando serve

Ripristino

Il ripristino (restore) ricrea un database a partire da un backup

Query

I DBMS utilizzano un linguaggio standard per interagire con gli utenti il linguaggio si chiama SQL (Structured Query Language)

Una query è un’interrogazione fatta al database, ossia una domanda espressa in linguaggio SQL

Il linguaggio SQL è strettamente formalizzato, con una sintassi estremamente rigida: l’ordine degli elementi è fisso

Le operazioni principali sono:

* Proiezione, che limita il numero di campi mostrati nel risultato
* Selezione, che limita il numero di record mostrati nel risultato

Query SELECT / FROM

Tutte le query partono con SELECT, la clausola SELECT consente di specificare quali campi mostrare nel risultato

La clausola FROM consente di specificare in quale tabella cercare i dati

La sintassi è:

**SELECT** <campi>

**FROM** <tabella>

I risultati (record) sono in ordine impredicibile, per forzare un ordinamento si utilizzala clausola ORDER BY

La sintassi è: **ORDER** **BY** <campi> ASC / DESC

ORDER BY prevede 2 opzioni: ASC o DESC, se non specificato viene applicato ASC

ASC ordina in ordine ascenente (crescent), DESC in ordine discendente (decrescente)

Le clausole all’interno di una query sono sempre indipendenti, quindi i campi presenti nella SELECT non dipendono da quelli presenti nella ORDER BY e viceversa

L’unica dipendenza è: tutti i campi presenti nella query devono essere presenti nella tabella FROM

Per fare più query, scrivere nell’ultima riga “;”

Esercizio:

scrivere la query che trova data di arrivo, data di partenza e numero di camera per ogni prenotazione

**SELECT** partenza, arrivo, camera

**FROM** prenotazioni

Criterio logico

Un criterio logico è un’espressione che produce come risultato VERO O FALSO

In algebra decimale, un’espressione produce come risultato un numero

WHERE

Una query può utilizzare la clausola per filtrare i record

La WHERE esprime un criterio logico: se il risultato è VERO, il corrispondente record comparirà nel risultato, se il risultato è FALSO, il corrispondente record non comparirà nel risultato

La sintassi è:

**SELECT** <campi>

**FROM** <tabella>

**WHERE** <criterio>

**ORDER** **BY** <campi> ASC / DESC

La clausola WHERE può confrontare qualsiasi coppia:

* 2 campi
* Un campo e una costante
* 2 costanti

Esercizio

Trovare data di arrive, data di partenza, import e caparra per le prenotazioni relative alle strutture 2 stelle, ordinate per importo crescente

**SELECT** arrivo, partenza, importo, caparra

**FROM** prenotazioni

**WHERE** tipo\_struttura = '2 stelle'

**ORDER** **BY** importo

Connettori logici

Il criterio logico (WHERE) consente di effettuare un confront, se serve effettuare più confronti è necessario unirli con un connettore logico, in modo da creare un criterio complesso

I connettori logici sono 2:

* AND
* OR

Es:

A = b AND c = d

A = b OR c = d

Il connettore produce un criterio composito: l’intera struttura può valere solo VERO o FALSO

Per prima cosa vengono valutati I 2 criteri, poi il connettore unisce I risultati e produce il risultato complessivo

per ogni connettore è possibile definire una tabella di verità (legami fra input e output)

es:

trovare tutti I clienti che hanno nome Claudio e cognome Rossi

**SELECT** \*

**FROM** clienti

**WHERE** nome = 'Claudio'

**AND** cognome = 'Rossi'

Esercizio:

Trovare tutti I dati (\*)delle prenotazioni relative aale strutture 3 stelle con caparra superior a 100

**SELECT** \*

**FROM** prenotazioni

**WHERE** tipo\_struttura = '3 stelle'

**AND** caparra > 100

Range

Quando si ha a che fare con range (intervalli) di dati, la logica dei connettori è semplice:

* Se si tratta di un range interno si utilizza AND
* Se si tratta di un range esterno si utilizza OR

Es:

Tutti I dati compresi fra 40 e 70: dato > 40 AND dato < 70 (range interno)

Tutti I dati tranne quelli compresi fra 40 e 70: dato < 40 OR dato > 70 (range esterno)

Esercizio:

Trovare data di arrivo, importo, caparra per tutte le strutture e 4 stelle

**SELECT** arrivo, importo, caparra

**FROM** prenotazioni

**WHERE** tipo\_struttura = '3 Stelle'

**OR** tipo\_struttura = '4 Stelle'

Chiave primaria

La chiave primaria è un insieme di campi che definisce univocamente un record

La chiave primaria consentedi semplificare le ricerche: fornire I valori dei campi della chiave primaria consente di identificare esattamente un record

La chiave primaria costituisce un vincolo implicito:

* I campi della chiave primaria non possono contenere null
* I valori della combinazione di campi della chiave primaria non possono essere duplicati

L’utilizzo primario della chiave primaria è legato alle ricerche: la chiave primaria fornisce un accesso privilegiato ai record perchè è correlata da una scorciatoia per le ricerche (indice) che le rende più veloci

Per motivi di performance, si cerca di utilizzare campi numerici interi come chiavi primarie

Se la chiave primaria è un numero intero, è facile garantire che sia univoca generando automaticamente il valore del campo: questo comportamento si chiama IDENTITY (autoincremento)

Una chiave primaria autoincrementante garantisce che il valore sia sempre presente e sempre univoco

Relazioni

Nei database relazionali, I dati possono essere suddivisi in più tabelle messe in relazone fra loro

Le relazioni servono a scrivere I dati una volta sola, risparmiando spazio sullo storage e riducendo gli errori dei dati

Le relazioni rendono meno leggibili I dati perchè per mettere in relazione 2 tabelle si importa il valore della chiave primaria

Diagramma e/r

Le relazioni e le strutture delle tabelle vengono rappresentate nel diagramma e/r (entity/relationship)

Le frecce identificano le relazioni, mostrando quali tabelle sono in relazione reciproca e quali campi sono oggetto della relazione

AND/OR

All’interno di una query si possono utilizzare I connettori AND e OR

Se si utilizzano più volte l’effetto si concatena

Es:

SELECT\*

FROM prenotazioni

WHERE camera = ‘22’

AND importo > 200

AND caparra < 50

Se si utilizzano contemporaneamente AND e OR, è necessario tenere conto della priorità degli operatori

Es:

SELECT \*

FROM prenotazioni

WHERE camera = 22

OR camera > ‘25’

AND caparra < 50 🡪 V/F

AND ha la priorità superiore a OR, quindi viene sempre eseguito prima (come in algebra decimale \* e +)

Per alterare la priorità si utilizzno le parentesi

Es:

SELECT \*

FROM prenotazioni

WHERE (camera = ‘22’

OR camera = ‘25’)

AND caparra < 50

Esercizio: trovare tutti I dati delle prenotazioni che hanno importo maggiore di 100, caparra minore di 50, per le strutture 2 o 3 stelle

**SELECT** \*

**FROM** prenotazioni

**WHERE** importo > 100

**AND** caparra < 50

**AND** (tipo\_struttura = "2 stelle"

**OR** tipo\_struttura = "3 stelle")

Esercizio: trovare nome, cognome e data di nascita per tutti I clienti che hanno nome Mario, Giorgia, Luigi e cognome Rossi, Bianchi, Rossini

**SELECT** nome, cognome, dataNascita

**FROM** clienti

**WHERE** (nome = 'Mario'

**OR** nome = 'Giorgia'

**OR** nome = 'Luigi')

**AND** (cognome = 'Rossi'

**OR** cognome = 'Bianchi'

**OR** cognome = 'Rossini')

Espressioni algebriche

All’interno di una query è possibile utilizzare espressioni algebriche

Se le espressioni sono all’interno della SELECT, il campo risultante non ha nome

Es:

SELECT arrivo, partenza, importo / 100 \* 22

FROM prenotazioni

Alias

All’interno della SELECT è possibile cambiare (o associare) il nome ad un campo

Per farlo si utilizza AS

Esercizio: rovare data di arrivo numero di camera e saldo per tutte le prenotazioni relative a strutture 2, 3, e 4 stelle

**SELECT** arrivo, camera, importo - caparra **AS** saldo

**FROM** prenotazioni

**WHERE** tipo\_struttura = '2 stelle'

**OR** tipo\_struttura = '3 stelle'

**OR** tipo\_struttura = '4 stelle’

IN

L’operatore IN è un operatore che consente di confrontare un campo con una serie valore, il risultato sarà VERO se almeno uno dei valori è uguale al contenuto del campo

L’operatore IN può sostituire una sequenza di OR che operano sullo stesso campo

Es:

**SELECT** \*

**FROM** prenotazioni

**WHERE** tipo-struttura **IN** ('2 stelle', '3 stelle', '4 stelle')

**SELECT** \*

**FROM** prenotazioni

**WHERE** tipo\_struttura = '2 stelle'

**OR** tipo\_struttura = '3 stelle'

**OR** tipo\_struttura = '4 stelle')

Queste query sono equivalenti

Prodotto Cartesiano

Quando all’interno della FROM si utilizzano più tabelle separate con una virgola, si genera un prodotto cartesiano

Il prodotto cartesiano è l’insieme di tutte le combinazioni di record presenti nelle tabelle specificate nella FROM

Filtrando un prodotto cartesiano è possibile ottenere la tabella corretta (quella che ricostruisce la relazione)

Es:

**SELECT** \*

**FROM** prenotazioni, clienti

**WHERE** cliente = id\_cliente

JOIN

Le operazioni di JOIN consentono di unire più tabelle, per creare query che interrogano contemporaneamente più di una tabella

Le JOIN permettoo di modellare realtà complesse: I dati memorizzati in più tabelle possono essere visti come se fossero scritti in un’unica grande tabella

Le operazioni di JOIN sono di diverso tipo

INNER JOIN

LEFT OUTER JOIN

RIGHT OUTER JOIN

FULL OUTER JOIN

L’operazione INNER JOIN ricostruisce il legame fra 2 tabelle, ricostruendo la tabella complessive che deriva dall’unione

La sintassi è del tipo

<tabella 1> INNER JOIN <tabella 2> ON <criterio logico>

Il criterio logico definisce la regola di unione (relazione)

La INNER JOIN dipende dal diagramma E/R, non dalla query che la contiene

Es:

**SELECT** \*

**FROM** prenotazioni

**INNER** **JOIN** clienti **ON** cliente = id\_cliente

All’interno di una query è sempre possibile utilizzare il nome della tabella come prefisso al nome del campo, questo prefisso è obbligatorio in caso di campi omonimi

Es:

Prenotazioni.cliente identifica il campo cliente presente nella tabella prenotazioni

Esercizio

Trovare nome e cognome di tutti I clienti che hanno prenotato in strutture 2 o 3 stelle con importi maggiori di 100

**SELECT** nome, cognome

**FROM** prenotazioni

**INNER** **JOIN** clienti **ON** cliente = id\_cliente

**WHERE** tipo\_struttura **IN** ('2 stelle', '3 stelle')

**AND** importo > 100

Esercizio

Trovare nome, cognome, data di arrivo, data di partenza e importo per le prenotazioni dei clienti che abitano in provincia di Milano e hanno prenotato una struttura 3 o 4 stelle, ordinando per data di arrivo (dalla più recente alla più vecchia) e cognome (ascendente)

**SELECT** nome, cognome, arrivo, partenza, importo

**FROM** prenotazioni

**INNER** **JOIN** clienti **ON** ID\_cliente = cliente

**INNER** **JOIN** citta **ON** id\_citta = clienti.citta

**WHERE** provincia\_nome = 'Milano'

**AND** tipo\_struttura **IN** ('3 stelle', '4 stelle')

**ORDER** **BY** arrivo **DESC**, cognome

LIKE

l’operatore LIKE consente di verificare se 2 parole corrispondono (in maniera simile a =) e consente di utilizzare caratteri jolly

un carattere jollyè un simbolo che soddisfa il criterio con diverse combinazioni di valori

es:

**SELECT** \*

**FROM** clienti

**WHERE** cognome **LIKE** 'C%'

Trova tutti I clienti il cui cognome comincia con la c

Il carattere jolly % può essere sostituito da qualsiasi sequenza di caratteri (anche una stringa vuota)

Esercizio

Trovare nome, cognome, importo, camera per tutte le prenotazioni dei clienti il cui nome inizia con A

**SELECT** nome, cognome, importo, camera

**FROM** prenotazioni

**INNER** **JOIN** clienti **ON** prenotazioni.cliente = id\_cliente

**WHERE** nome **LIKE** 'A%

Relazioni

All'interno di un database le tabelle possono essere in relazione fra loro secondo diversi modelli

Le relazioni possono essere:

- 1:n

- m:n

- 1:1

1:n

Una relazione 1:n (uno a molti) è una relazione nella quale un record della tabella padre può corrispondere a più record della tabella figlia

M:n

Una relazione m:n (molti a molti) è una relazione nella quale non c'è il concetto di padre e figlia, ma ogni record di ognuna delle due tabelle può essere in relazione con più record dell'altra

1:1

Una relazione 1:1 (uno a uno) è una relazione nella quale c'è corrispondenza fra le chiavi primarie di due tabelle

Serve a suddividere una tabella in due parti

Integrità referenziale

Nelle relazioni (in particolare 1:n) si può utilizzare l’integrità referenziale

L’integrità referenziale è una regola che definisce quali vincoli ci sono fra le tabelle

In particolare è possibile attivare il controllo di integrità, per garantire che ci sia la corrispondenza con il padre: integrità di inserimento, questo controllo impedisce di inserire record che hanno valore di chiave importata che non corrisponde a nessun valore della chiave primaria della tabella in relazione

L’integrità referenziale può definire dei vincoli anche in fase di modifica o di cancellazione

In fase di modifica, può impedire la modifica del record se è stato specificato un valore di chiave importata che non corrispone alla chiave primaria in relazione

L’integrità referenziale può definire dei vincoli anche in fase di modifica o di cancellazione

In fase di modifica nella tabella figlia, se viene indicato un valore di chiave importata che non corrisponde a nessuna chiave primaria della tabella in relazione, la modifica viene bloccata

In fase di cancellazione nella tabella padre, se esiste almeno un record nella tabella figlia, la cancellazione viene bloccata

L’integrità referenziale può anche propagare le modifiche:

* Se si modifica la chiave primaria della tabella padre, i record figli vengono automaticamente modificati
* Se si elimina un record della tabella padre, i record figli vengono automaticamente eliminati

Distinct

DISTINCT consente di eliminare dal risultato i record duplicati

La sintassi è

SELECT DISTINCT <campi>

**SELECT** **DISTINCT** citta.citta, nome

**FROM** citta

**INNER** **JOIN** clienti **ON** id\_citta = clienti.citta

**INNER** **JOIN** prenotazioni **ON** id\_cliente = cliente

**WHERE** tipo\_struttura = '3 stelle'

Esercizio

Trovare nomi e cognomi dei clienti della provincia di Bologna e Rimini che hanno prenotato in una struttura 3 o 4 stelle con importo compreso fra 100 e 150

**SELECT** **DISTINCT** nome, cognome

**FROM** clienti

**INNER** **JOIN** citta **ON** clienti.citta = id\_citta

**INNER** **JOIN** prenotazioni **ON** id\_cliente = cliente

**WHERE** importo > 100

**AND** importo < 150

**AND** tipo\_struttura **IN** ('3 stelle', '4 stelle')

**AND** provincia\_nome **IN** ('Bologna', 'Rimini')

ATTENZIONE!!!

Una INNER JOIN costituisce un criterio di filtraggio implicito

Una INNER JOIN non necessaria può eliminare dal risultato dei record significativi

Funzioni di aggregazione

Le funzioni di aggregazione riducono il numero di righe di una tabella

Le righe risultanti da un’aggregazione vengono calcolate, quindi mostrano dati che derivano da un calcolo fatto sui valori presenti in tabella

Le funzioni di aggregazione presenti in SQL sono:

* COUNT (\*), che conta i record
* SUM (<campo>), che calcola il totale dei valori presenti nel campo
* AVG (<campo>), che calcola la media dei valori presenti nel campo
* MIN (<campo>), che restituisce il valore più piccolo presente nel campo
* MAX (<campo>), che restituisce il valore più grande presente nel campo

All’interno della SELECT è possibile utilizzare più funzioni di aggregazione, ma non è possibile mescolare campi e funzioni di aggregazione

La sintassi è:

SELECT <funzioni di aggregazione>

FROM <tabella>

**SELECT** **COUNT**(\*) **AS** conteggio, **SUM**(importo) **AS** totale, **AVG**(importo) **AS** media

**FROM** prenotazioni

Esercizio

Trovare il numero di prenotazioni relative a clienti che risiedono in provincia di milano e hanno importo superiore a 200

**SELECT** **COUNT**(\*) **AS** conteggio

**FROM** prenotazioni

**INNER** **JOIN** clienti **ON** cliente = id\_cliente

**INNER** **JOIN** citta **ON** clienti.citta = id\_citta

**WHERE** provincia\_nome = 'Milano'

**AND** importo > 200

Esercizio

Trovare l’importo toale e l’importo medio delle prenotazioni relative a clienti che risiedono in provincia di roma e bologna

**SELECT** **SUM**(importo) **AS** totale, **AVG**(importo) **AS** media

**FROM** prenotazioni

**INNER** **JOIN** clienti **ON** cliente = id\_cliente

**INNER** **JOIN** citta **ON** clienti.citta = id\_citta

**WHERE** provincia\_nome **IN** ('bologna', 'roma')

GROUP BY

Per utilizzare nella SELECT sia campi che funzioni di aggregazione, è necessario utilizzare la clausola GROUP BY e inserire nella GROUP BY tutti I campi presenti nella SELECT

La GROUP BY raggruppa I record: crea un gruppo per ogni valore univoco e applica le funzioni di aggregazione al gruppo

La GROUP BY va posizionata dopo WHERE e prima di ORDER BY

La sintassi è:

GROUP BY <campi>

Es:

**SELECT** nome, cognome, **SUM**(importo) **AS** totale

**FROM** prenotazioni

**INNER** **JOIN** clienti **ON** cliente = id\_cliente

**GROUP** **BY** nome, cognome

Funzioni di supporto

In SQL si possono utilizzare funzioni di supporto per elaborare I dati

Funzioni comuni sono:

* ROUND(valore, decimali), arrotonda il valore al numero di decimali indicato
* YEAR(data), restituisce l’anno di una data
* CONCAT(stringa 1, stringa 2, …), concatena le stringhe

Es:

**SELECT** **ROUND**(importo/100\*22, 2) **AS** iva

**FROM** prenotazioni

**WHERE** **YEAR**(arrivo) = 2015

**AND** **MONTH**(arrivo) = 10;

**SELECT** **CONCAT**(nome, ' ', cognome) **AS** nome\_e\_cognome

**FROM** clienti;

Esercizio

Trovare l’importo medio e l’importo massimo speso dai clienti di ogni città

**SELECT** citta.citta, **ROUND**(**AVG**(importo), 2) **AS** media, **MAX**(importo) **AS** **max**

**FROM** citta

**INNER** **JOIN** clienti **ON** id\_citta = clienti.citta

**INNER** **JOIN** prenotazioni **ON** id\_cliente = cliente

**GROUP** **BY** citta.citta

Esercizio

Trovare, per ogni regione, il numero e l'importo medio delle prenotazioni che hanno importo compreso fra 100 e 300 e sono per strutture 2 o 4 stelle

**SELECT** regioni.regione, **COUNT**(\*) **AS** conteggio, **ROUND**(**AVG**(importo), 2) **AS** media

**FROM** prenotazioni

**INNER** **JOIN** clienti **ON** id\_cliente = cliente

**INNER** **JOIN** citta **ON** clienti.citta = id\_citta

**INNER** **JOIN** regioni **ON** citta.regione = id\_regione

**WHERE** importo >= 100

**AND** importo <= 300

**AND** tipo\_struttura **IN** ('2 stelle', '4 stelle')

**GROUP** **BY** regioni.regione

Viste

Una vista è una query salvata nel database

Quando si crea una vista si salva la domanda, non la risposta

Having

La clausola HAVING consente di imporre criteri su funzioni di aggregazione

La sintassi è:

HAVING <criterio>

Es:

SELECT tipo\_struttura, AVG(importo)

FROM prenotazioni

GROUP BY tipo\_struttura

HAVING AVG(importo) > 150

Esercizio

Trovare tutte le camere (campo camera) che hanno generato un fattorato (importo totale) maggiore di 500

**SELECT** camera

**FROM** prenotazioni

**GROUP** **BY** camera

**HAVING** **SUM**(importo) > 500

Siccome nella having è presente una funzione di aggregazione e nella SELECT no, il GROUP BY è necessario

Procedura per scrivere una query

1. Identificare le tabelle (FROM)
2. Identifichiamo I campi e le unzioni di aggregazione (SELECT)
3. Se necessario, aggiungere GROUP BY
4. Se necessario, aggingere I criteri sui campi (WHERE)
5. Se necessario, aggiungere I criteri sulle funzioni di aggregazione (HAVING)
6. Se necessario aggiungere l’ordinamento (ORDER BY)

Ordine delle clausole

Le clausole sono sempre nello stesso ordine:

1. SELECT
2. FROM
3. WHERE
4. GROUP BY
5. HAVING
6. ORDER BY

Esercizio:

Trovare quali province (nome della provincia) hanno generato più di 10 prenotazioni con importo superiore a 100

**SELECT** provincia\_nome

**FROM** citta

**INNER** **JOIN** clienti **ON** id\_citta = clienti.citta

**INNER** **JOIN** prenotazioni **ON** id\_cliente = cliente

**WHERE** importo > 100

**GROUP** **BY** provincia\_nome

**HAVING** **COUNT**(\*) > 10

DELETE

Le query di cancellazione eliminano I record da una tabella

NB: non è possibile annullare la cancelazione, è importante fare un backup del database (o parte di esso) prima di modificare il suo contenuto

La sintassi è:

DELETE

FROM <tabella>

WHERE <criterio>

Es:

**DELETE**

**FROM** clienti

**WHERE** cognome = 'rossi'

Questa query elimina dalla tabella clienti tutti I record nei quali il campo cognome contiene ‘rossi’

NB: le query di DELETE hanno la stessa logica e la stessa sintassi delle SELECT, quindi è possibile scrivere una select per vedere l’anteprima dei dati che verranno cancellati, per poi sostituire la clausola SELECT con DELETE

Esercizio:

Eliminare tutte le prenotazioni relative ai 2 stelle con importo minore di 100 e caparra mnore di 50

**DELETE**

**FROM** prenotazioni

**WHERE** importo < 100

**AND** caparra < 50

**AND** tipo\_struttura = '2 stelle'

UPDATE

Le query di aggiornamento dati (UPDATE) consentono di modificare I dati presenti nei campi di una tabella

La sintassi è:

UPDATE <tabella>

SET <campo 1> = <valore 1>, <campo 2> = <valore 2>, …, <campo n> = <valore n>

WHERE <criterio>

Es:

**UPDATE** prenotazioni

**SET** importo = 200, caparra = 50

**WHERE** camera = '22'

NB: il nuovo valore può essere il risultato di un calcolo

**UPDATE** prenotazioni

**SET** importo = importo + (importo / 100 \* 10), caparra = caparra + (caparra / 100 \* 10)

**WHERE** camera = '22'

Questa query aumenta del 10% importo e caparra per le prenotazioni relative alla camera 22

Esercizio

Impostare l’importo della caparra a 100 e l’importo della prenotazione al doppio dell’importo attuale per le prenotazioni relative ai 3 e 4 stelle con più di 10 giorni di permanenza

**UPDATE** prenotazioni

**SET** importo = importo \* 2, caparra = 100

**WHERE** tipo\_struttura **IN** ('3 stelle', '4 stelle')

**AND** giorni\_permanenza > 10

INSERT

Le query di inserimento (INSERT) inseriscono record in una tabella del database

Esistono diverse sintassi, a seconda del tipo di inserimento che si vuole realizzare

I dati possono essere scritti nella query (INSERT VALUES) oppure selezionati da tabella (INSERT SELECT)

La sintassi per l’inserimento di valori è:

INSERT INTO <tabella> (<campo1>, <campo2>, …, <campo n>)

VALUES (<valore1>, <valore2>, …, <valore n>)

Es:

INSERT INTO clienti (nome, cognome)

VALUES (‘mario’, ‘rossi’)

I campi che non sono presenti nella query verranno valorizzati con:

* Valore automatico, se è una chiave primaria autoincrementante
* Null, se la tabella non prevede default per quei campi
* Valore predefinito, se la tabella prevede default per quei campi

Una query di INSERT può fallire perchè I dati da inserire violano uno o più vincoli presenti in tabella

Esercizio:

Inserire una prenotazione per il cliente 15, camera 33, arrivo 1/1/2024, partenza 4/1/2024, tipo struttura 4 stelle, importo 350, caparra 75

**INSERT** **INTO** prenotazioni (cliente, camera, arrivo, partenza, tipo\_struttura, importo, caparra)

**VALUES** (15, 33, '2024-01-01', '2024-01-04', '4 stelle', 350, 75)

La INSERT SELECT consente di inserire dati in una tabella, estraendo I valori da una query di SELECT

La sintassi è:

INSERT INTO <tabella> (<campo 1, <campo2>, …, <campo n>)

SELECT <campo 1>, <campo 2>, …, <campo n>

FROM <tabella>

WHERE <criterio>

Es:

INSERT INTO prenotazioni (cliente, arrivo partenza, importo, caparra)

SELECT id\_cliente, ‘2024-02-01’, ‘2024-02-05’, 150, 25

FROM clienti

WHERE nome = ‘mario’

AND cognome = ‘rossi’

CRUD

Le query consentono di effettuare qualsiasi tipo di operazione CRUD:

* Create
* Research
* Update
* Delete

Spesso si combinano le query per ottenere risultati articolati (es: INSERT SELECT)

LIMIT / TOP

La clausola LIMIT (opzione TOP in SQL Server) consente di limitareil numero di record restituiti da una query

La sintassi è:

SELECT <campi>

FROM <tabella>

WHERE <criterio>

ORDER BY <campi>

LIMIT <range>

Es:

*mySQL:*

SELECT \*

FROM clienti

WHERE cognome = ‘rossi’

LIMIT 0, 2

*SQL Server:*

SELECT TOP 2

FROM clienti

WHERE cognome = ‘rossi’

NB: LIMIT va utilizzata assieme a ORDER BY, altrimenti I risultati sono “a caso”

Esercizio:

Trovare le 10 prenotazioni con l’importo più alto per le strutture 3 stelle

**SELECT** \*

**FROM** prenotazioni

**WHERE** tipo\_struttura = '3 stelle'

**ORDER** **BY** importo **DESC**

**LIMIT** 0, 10

Esercizio:

Trovare nome, cognome e data di nascita del cliente che ha speso di più (importo)

**SELECT** nome, cognome, dataNascita

**FROM** clienti

**INNER** **JOIN** prenotazioni **ON** id\_cliente = cliente

**GROUP** **BY** id\_cliente, nome, cognome, dataNascita

**ORDER** **BY** **SUM**(importo) **DESC**

**LIMIT** 0, 1

OUTER JOIN

Le OUTER JOIN consentono di ricomporre I dati suddivisi in più tabelle, mantenendo anche I recordche non hanno corrispondenza fra le tabelle

Esistono 3 tipi di OUTER JOIN:

* LEFT OUTER JOIN, che mantiene tutti I record della tabella “di sinistra” e I soli record della tabella “di destra” che hanno una corrispondenza
* RIGHT OUTER JOIN, che mantiene tutti I record della tabella “di destra” e I soli record della tabella “di sinistra” che hanno una corrispondenza
* FULL OUTER JOIN, che mantiene tutti I record di entrambe le tabelle

NB: mySQL non supporta FULL OUTER JOIN

I record che non hanno corrispondenza verranno riempiti con valori NULL nelle altre tabelle

Es:

SELECT \*

FROM clienti

LEFT OUTER JOIN prenotazioni ON id\_cliente = cliente

Trova tutti dati di tutti I clienti:

* Se il cliente ha delle prenotazioni verranno mostrati I dati delle prenotazioni
* Se il cliente non ha prenotazioni, I campi della tabella prenotazioni verranno riempiti con NULL

IS NULL

NULL è un valore speciale, pertanto non può essere utilizzato con I normali operatori logici (>, <, =, …) ma serve un operatore apposito: IS

La sintassi è:

<campo> IS NULL

Restituisce VERO se il valore presente nel campo è NULL, FALSO in tutti gli altri casi

ES:

SELECT \*

FROM prenotazioni

WHERE giorni\_permanenza IS NULL

IS NULL viene tipicamente utilizzato per trovare le mancate corrispondenze, assieme a OUTER JOIN

Es:

SELECT \*

FROM regioni

LEFT OUTER JOIN città ON id\_regione = citta.regione

WHERE id\_citta IS NULL

Esercizio:

Trovare id, nome, cognome di tutti I clienti che non hanno prenotazioni

**SELECT** id\_cliente, nome, cognome

**FROM** clienti

**LEFT** **OUTER** **JOIN** prenotazioni **ON** id\_cliente = cliente

**WHERE** id\_prenotazione **IS** **NULL**

Sottoquery

Una sottoquery una query che viene scritta all’interno di un’altra per fornire valori parziali

La sottoquery deve essere scritta fra parentesi tonde, il DBMS esegue prima la sottoquery, sostituisce suo testo con I valori del risultato, poi esegue la query principale

Es:

SELECT \*

FROM clienti

WHERE citta IN (

SELECT id\_citta

FROM citta

WHERE regione = 3

)

NB: se utilizzate con gli operatori =, >, <, <>, >=, <= le sottoquery devono restituire un solo valore

Se utilizzate con l’operatore IN, le sottoquery possono restituire più valori (record) ma un solo campo

UNION

La clausola UNION consente di concatenare I risultati di 2 query

La sintassi è

<query 1>

UNION

<query 2>

Il risultato avrà:

* La struttura (nomi di campo e tipi di dato) definita da <query 1>
* I dati (record) restituiti da query 1, seguiti da quelli restituiti da query 2

Le 2 query devono restituire esattamente lo stesso numero di campi, altrimenti le 2 strutture non corrispondono e l’intera query non viene eseguita

Attenzione: quando si utilizza UNION è pericoloso utilizzare SELECT \*, perchè le tabelle possono camiare struttura nel tempo (orizzonte di vita dell’applicazione)

Es:

**SELECT** nome

**FROM** clienti

**WHERE** cognome = 'rossi'

**UNION**

**SELECT** nome

**FROM** clienti

**WHERE** cognome = 'bianchi'

UNION consente di unire risultati presi da tabelle diverse

Le singole query concatenate possono utilizzare tutte le clausole SQL, tranne ORDER BY, se si vuole ordinare il risultato si deve mettere un’unica ORDER BY alla fine della query complessiva

UNION / UNION ALL

La UNION rimuove eventuali record restituiti da query 2 e già restituiti da query 1

La UNION ALL concatena completamente I risultati delle query, compresi i record duplicati

Utilizzando UNION (o UNION ALL) è possibile concatenare qualsiasi numero di query, la sintassi è:

<query 1>

UNION

<query 2>

UNION

[…]

<query n>

Tabelle derivate

In SQL è possibile utilizzare tabelle derivate, ossia tabelle create all’interno della query tramite sottoquery ed utilizzare come se fossero normali tabelle