Appunti 06-06-26 - Modulo 5

Selezione sui gruppi e clausola HAVING

Nell'analisi di grandi moli di dati, come ad esempio l'intera popolazione italiana composta da circa sessanta milioni di record, emerge spesso la necessità di operare calcoli aggregati, come il salario medio per città. Dopo aver raggruppato i dati tramite l'istruzione `GROUP BY`, può presentarsi l'esigenza di filtrare questi gruppi in base a criteri aggregati, ad esempio considerando solo le città con una popolazione significativa superiore ai 30.000 abitanti per escludere i piccoli nuclei.

È fondamentale comprendere che tale filtro non può essere applicato alla singola riga tramite la clausola `WHERE`, poiché il conteggio degli abitanti è una proprietà del gruppo intero e non del singolo record. Lo strumento corretto per operare questa selezione è la clausola `HAVING`, che permette di applicare un predicato a una funzione di gruppo, come nel caso di `HAVING COUNT(*) > 30000`, trattenendo solo i gruppi che soddisfano la condizione imposta.

È possibile combinare queste operazioni in query complesse che includono proiezione, selezione con `WHERE` per filtrare i singoli record (ad esempio escludendo salari bassi), raggruppamento, filtro sui gruppi con `HAVING` e infine ordinamento dei risultati.

Interrogazioni su più tabelle e Prodotto Cartesiano

Quando l'analisi richiede di confrontare dati distribuiti su entità diverse, come il salario delle persone e il costo della vita nelle rispettive città, è necessario operare su più tabelle contemporaneamente.

Ipotizzando di avere una tabella `Person` con i dati salariali e una tabella `City` con i costi medi, un primo tentativo di interrogazione che elenchi entrambe le tabelle nella clausola `FROM` genera quello che viene definito prodotto cartesiano o Cross Join.

Questa operazione combina indiscriminatamente ogni riga della prima tabella con tutte le righe della seconda, generando un numero di "super-righe" pari al prodotto delle righe delle due tabelle originali. Il risultato contiene tutte le possibili

combinazioni, molte delle quali prive di senso logico, come associare una persona residente a Roma ai costi della città di Milano. Sebbene la proiezione permetta di selezionare solo alcune colonne, il problema di fondo persiste poiché le righe rimangono combinate in modo non correlato.

Il concetto di JOIN e INNER JOIN

Per ottenere informazioni coerenti è necessario filtrare il prodotto cartesiano imponendo una condizione di selezione che specifichi quali combinazioni di righe mantenere. Questa operazione trasforma il prodotto cartesiano in un `JOIN`.

Nello specifico, l'`INNER JOIN` è un'operazione che trattiene solo le righe le cui componenti sono in relazione tra loro secondo un predicato specifico, scartando tutte le altre. In pratica, si realizza aggiungendo una condizione nella clausola `WHERE`, come `Person.City = City.Name`, che istruisce il database ad accoppiare le righe solo quando il campo città della persona coincide con il nome della città.

Questo predicato di join funge da filtro logico che, analizzando le righe generate dal prodotto cartesiano, restituisce come risultato finale solo le associazioni corrette, permettendo così di calcolare la differenza tra salario e costi per la città effettiva di residenza.

Il concetto formale e l'applicazione del JOIN

Formalmente, il JOIN è un'operazione che combina i dati provenienti da due o più tabelle eseguendo un prodotto cartesiano a cui viene immediatamente applicata una specifica condizione di filtro, definita predicato di join. Questa condizione funge da regola di selezione fondamentale per stabilire quali associazioni tra le righe delle diverse tabelle debbano essere preservate, in quanto logicamente correlate, e quali debbano essere scartate. In sostanza, il JOIN è lo strumento tecnico che concretizza la relazione tra entità diverse, permettendo al database di costruire un risultato unificato che mantiene solo le combinazioni di righe che soddisfano il criterio di corrispondenza imposto.

Definizione Formale di Predicato di Join

Il **predicato di join** (o condizione di unione) è l'espressione logica che determina quali coppie di righe, provenienti da tabelle diverse, debbano essere considerate

"collegate" e quindi mantenute nel risultato finale.

Tecnicamente, dato un prodotto cartesiano tra una Tabella A e una Tabella B (che combina ogni riga di A con ogni riga di B), il predicato di join agisce come un filtro selettivo. Esso specifica una regola di confronto, solitamente un'uguaglianza (`=`), tra una colonna della prima tabella e una colonna della seconda.

Se abbiamo una tabella `Person` con un campo `CityID` e una tabella `City` con un campo `ID`:

- Il prodotto cartesiano genera tutte le combinazioni (es. Persona di Roma associata ai dati di Milano).
- Il **predicato di join** `Person.CityID = City.ID` controlla ogni combinazione.
- Se il numero identificativo coincide (es. 5 = 5), il predicato è vero e la riga viene tenuta. Se non coincide (es. 5 = 9), la riga viene scartata.

Evoluzione delle relazioni e Chiavi Esterne

L'utilizzo dei nomi delle città come elemento di collegamento presenta criticità strutturali, poiché i nomi non sono necessariamente univoci e possono portare ambiguità. Per ovviare a questo problema, è buona norma professionale rifattorizzare la tabella `City` introducendo un campo numerico `ID` che funga da chiave primaria univoca. Conseguentemente, la tabella `Person` viene modificata sostituendo il nome della città con un riferimento a questo identificativo, creando un campo `CityID`.

Questo campo costituisce una chiave esterna, ovvero un puntatore che collega logicamente una riga della tabella persone a una specifica riga della tabella città. A differenza della chiave primaria, la chiave esterna `CityID` in `Person` può contenere valori duplicati, riflettendo la relazione uno-a-molti per cui una singola città può ospitare più persone. Il predicato di join viene quindi aggiornato per confrontare gli identificativi numerici, garantendo un collegamento privo di ambiguità.

Relazioni multiple e Integrità Referenziale

Tra due tabelle possono esistere molteplici relazioni distinte. Prendendo come esempio un sistema di gestione auto, una vettura (tabella `Car`) è collegata alle persone (tabella `Person`) sia attraverso la relazione di proprietà che tramite quella

di revisione. Questo scenario richiede la presenza di due distinte chiavi esterne nella tabella `Car`: `OwnerID` e `RevisorID`, ciascuna delle quali punta all'ID della tabella `Person` ma rappresenta un legame logico differente che deve essere gestito separatamente nelle interrogazioni

L'uso delle chiavi esterne introduce il tema dell'integrità referenziale, necessaria per evitare il fenomeno delle righe "orfane", ovvero record che puntano a entità non esistenti (come un `CityID` che non trova corrispondenza nella tabella `City`). In ambito enterprise, la prassi comune prevede di impedire la creazione di orfani dichiarando esplicitamente le relazioni tramite il vincolo `FOREIGN KEY` al momento della creazione della tabella.

Questo vincolo obbliga il database a garantire che ogni valore inserito nella chiave esterna esista nella tabella di riferimento. Inoltre, vengono definite politiche di gestione dei cambiamenti tramite le clausole `ON UPDATE` e `ON DELETE`. La clausola `ON UPDATE CASCADE` assicura che, se l'ID di una persona viene modificato, il cambiamento si propaghi automaticamente a tutte le righe collegate nella tabella auto.

La clausola `ON DELETE RESTRICT`, invece, impedisce la cancellazione di un record "padre" (la persona) fintanto che esistono record "figli" (le auto) ad esso collegati, preservando la coerenza dei dati storici. L'insieme di queste regole costituisce l'integrità referenziale, un meccanismo di sicurezza fondamentale che garantisce l'assenza di orfani e la consistenza delle relazioni nel database.

Analisi della relazione tra tabelle

Nel contesto della progettazione di database, la relazione tra tabelle rappresenta il vincolo logico che determina quali righe di un'entità debbano essere associate alle righe di un'altra.

Quella che nel codice viene definita come un'associazione tra classi si traduce nel database in una relazione tra tabelle caratterizzata da una specifica cardinalità.

Un aspetto avanzato e fondamentale dell'analisi riguarda la possibilità che tra le stesse due tabelle intercorrano relazioni multiple e simultanee. Prendendo in esame un sistema di gestione veicoli, una singola automobile si relaziona con l'entità "Persona" attraverso due canali distinti: il proprietario del mezzo e il revisore tecnico che ne ha certificato l'idoneità.

PRODOTTO CARTESIANO

La sintassi "SELECT … FROM A, B" esegue un "prodotto cartesiano" fra la prima tabella (A) e la seconda (B) detto anche "cross join".

Combina tutte le righe della prima e della seconda tabella producendo super-righe composte, a sinistra, da una riga di A e a destra da una di B. Il prodotto cartesiano restituisce quindi tutte le possibili combinazioni fra le righe di A e di B.

INNER JOIN

Un JOIN è un prodotto cartesiano a cui viene imposta una condizione, vale a dire una forma di selezione. Essendo il prodotto cartesiano formato da super righe provenienti da più origini, il JOIN è la condizione che specifica quali combinazioni tenere e quali buttare. Un INNER JOIN è un caso particolare di JOIN, per cui vengono mantenute solo le super righe le cui componenti sono in relazione fra loro.

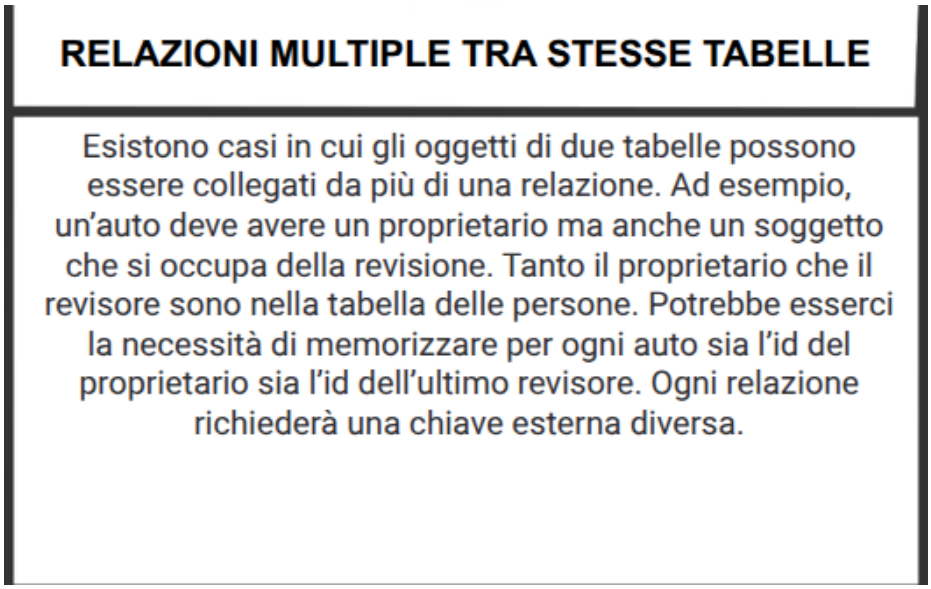
PREDICATO DI JOIN

Il predicato di join rappresenta la "condizione di unione" e serve per decidere in che modo le righe della prima tabella sono in relazione con le righe della seconda.

Es. Person.City = City.Name

RELAZIONI 1-N

È un rapporto tra valori per cui ad un oggetto (ossia una riga) in una tabella Person possono essere collegati più oggetti nell'altra tabella.
Ad esempio, a una città assoceremo più persone, mentre ogni persone vive in una città.



DOMANDE MODULO 5

1. Il prodotto cartesiano

Risposta corretta: combina le righe di due o più tabelle mettendo in relazione ogni riga con ogni altra

Spiegazione: La slide 12 definisce formalmente il prodotto cartesiano come "qualunque possibile combinazione" delle righe di A con le righe di B, senza applicare alcun filtro o chiave esterna. Viene generato un numero di righe pari al prodotto delle righe delle tabelle coinvolte (n x m), creando associazioni indiscriminate (es. Slide 11: "Il signor Rossi risulta vivere sia a Roma che a Milano").

2. L'Inner Join

Risposta corretta: combina le righe di due o più tabelle scartando le combinazioni di righe che non rispettano una condizione

Spiegazione: La slide 18 descrive l'INNER JOIN come un caso particolare in cui "vengono mantenute solo le super righe le cui componenti sono in relazione fra loro". Specifica chiaramente che "In caso contrario... la super-riga viene scartata".

3. Una chiave esterna

Risposta corretta: è un id che corrisponde alla chiave primaria di un'altra riga

Spiegazione: La slide 28 definisce la chiave esterna come un campo che "indica una riga a cui la riga in cui siamo è collegata". Solitamente si riferisce alla chiave primaria di un'altra tabella (es. `cityid` in `Person` punta a `id` in `City`), ma la definizione più precisa tra le opzioni è che corrisponda alla chiave primaria di un'altra riga (anche della stessa tabella, come nel caso del revisore citato nella slide 33, che è un'altra persona). L'opzione "di un'altra riga della stessa tabella" è troppo restrittiva, mentre "identifica la riga in cui è presente" è la definizione di chiave primaria.

4. Una inner join è

Risposta corretta: `nessuna delle precedenti`

MODULO 6

Gestione degli "Orfani" e Integrità Referenziale

Un problema cruciale nei database relazionali è quello delle righe "orfane", ovvero record che fanno riferimento a un'entità inesistente. Immaginiamo una tabella `Person` con un campo `cityid`: se questo campo contiene un valore (es. -30 o 999) che non corrisponde a nessuna riga valida nella tabella `City`, ci troviamo di fronte a un orfano. La persona "non sa dove vive" perché il suo puntatore è rotto. Questo può accadere se inseriamo un ID sbagliato o se la città di riferimento viene cancellata o modificata.

Di fronte agli orfani, la domanda è se accettarli o impedirli. La risposta dipende dal contesto:

- Se la relazione è **opzionale**, potremmo tollerare il dato mancante.
- Se la relazione è **obbligatoria** per la logica del programma (es. un'auto deve avere un proprietario), gli orfani vanno evitati.

La prassi comune in ambito enterprise è **impedire gli orfani** a monte. Questo si ottiene dichiarando esplicitamente le relazioni nel database tramite vincoli.

La Clausola FOREIGN KEY

Per formalizzare queste relazioni si usa la clausola `FOREIGN KEY`. In una tabella `Car`, ad esempio, definiamo che i campi `ownerid` e `revisorid` sono chiavi esterne che puntano alla tabella `Person`.

Questo ha un effetto immediato: il database impedisce l'inserimento di valori nulli o inesistenti. Non posso creare un'auto assegnandola a un proprietario con ID 500 se non esiste una persona con ID 500

Gestione dei Cambiamenti (ON UPDATE / ON DELETE)

Oltre a controllare l'inserimento, bisogna decidere cosa succede quando il dato "padre" (la persona) cambia o viene cancellato. Le clausole specifiche sono:

1. **ON UPDATE CASCADE:** Gestisce la modifica dell'ID padre. Se l'ID del Sig. Rossi cambia da 3 a 6, il database "propaga" automaticamente questo cambiamento a tutte le sue auto (i record "figli"), che aggiorneranno il loro `ownerid` a 6. Il legame viene preservato.
2. **ON DELETE RESTRICT:** Gestisce la cancellazione. Se provo a cancellare il Sig. Rossi e lui possiede delle auto, il database **blocca** l'operazione. È una misura di sicurezza: non posso eliminare il padre finché esistono figli che dipendono da lui. Per procedere, dovrei prima eliminare o riassegnare le sue auto.
3. **ON DELETE CASCADE (Alternativa):** Se avessimo scelto questa opzione, cancellando il proprietario verrebbero cancellate automaticamente anche tutte le sue auto. Utile per eliminare lo storico completo di un utente, ma pericoloso in altri contesti (es. non vorremmo cancellare le revisioni fatte da un meccanico solo perché il meccanico viene rimosso dal sistema).

Integrità Referenziale

L'insieme di queste regole (`FOREIGN KEY` + `ON UPDATE` + `ON DELETE`) costituisce l'**integrità referenziale**. È la garanzia tecnica che nel database non esistano collegamenti interrotti: ogni riferimento punta a un dato esistente e valido. In sintesi: "niente orfani".

ORFANO

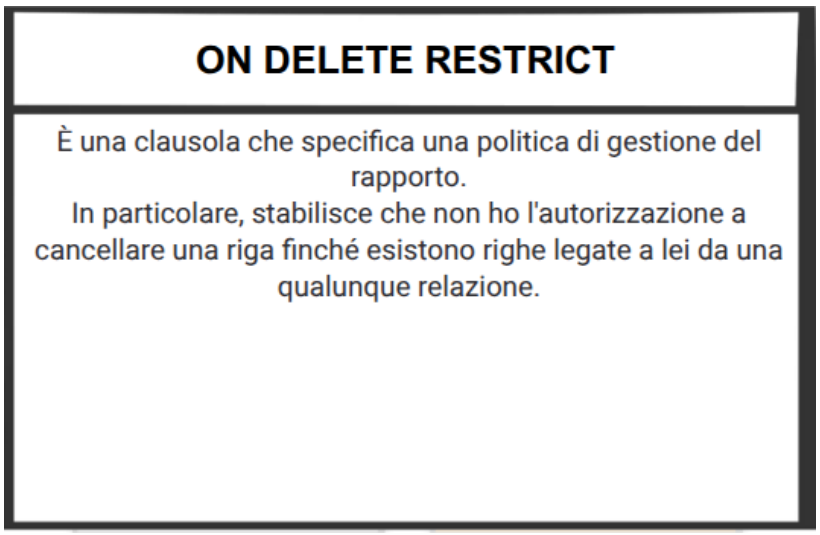
Nelle relazione tra tabelle di un DB, un "orfano" è una riga “senza padre” ossia senza un elemento di riferimento.
Questo accade quando alcuni valori nella tabella di riferimento vengono modificati o cancellati oppure quando il riferimento non viene proprio inserito da principio.

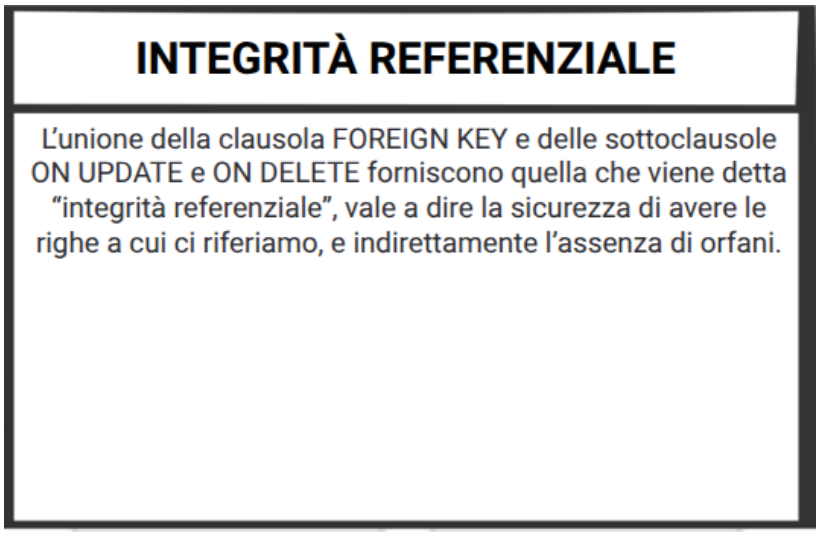
FOREIGN KEY

È una clausola che dichiara in maniera esplicita una relazione, vale a dire un legame fra due o più campi.

ON UPDATE CASCADE

È una clausola che specifica una politica di gestione del rapporto.
In particolare, stabilisce che un aggiornamento della chiave primaria sulla tabella padre porta a un aggiornamento della chiave esterna sulla tabella figlia.





DOMANDE MODULO 6

1. Si intende per "orfano":

Risposta: `Una riga lato n a cui non corrisponde una riga lato 1`

Perché: Nella slide "Definizione informale di orfano" (pag. 36), si spiega che un orfano è una riga della tabella figlia (lato 'n', come `Person` rispetto a `City`) il cui ID di riferimento non punta a nessuna riga esistente nella tabella padre (lato '1'). È "figlio" senza "padre".

2. L'integrità referenziale indica:

Risposta: Che non avremo orfani

Perché: La slide 44 afferma letteralmente: *"Integrità referenziale, vale a dire... l'assenza di orfani. Un modo rapido di ricordarne la funzione è appunto 'niente orfani'."* È il meccanismo di sicurezza che impedisce proprio la situazione descritta sopra.

3. L'opzione CASCADE dell'integrità referenziale riferita al caso DELETE indica:

Risposta: Che cancellando la riga padre verranno cancellate anche le righe figlie

Perché: La slide 43 discute le politiche alternative e dice espressamente: *"E se avessimo scelto on delete cascade... Cancellando la persona [padre], avremmo cancellato tutte le auto a lei collegate [figli] - la cancellazione si sarebbe propagata ai 'figli'."*