Basi di Dati

Il Linguaggio SQL

Il Linguaggio SQL

SQL (*Structured Query Language*)

- è di fatto lo standard tra i linguaggi per la gestione di basi di dati relazionali
- è un linguaggio dichiarativo (non-procedurale), ovvero non specifica la sequenza di operazioni da compiere per ottenere il risultato
- è "relazionalmente completo", nel senso che ogni espressione dell'algebra relazionale può essere tradotta in SQL

Il linguaggio SQL contiene funzionalità di :

Data Definition Language DDL - Linguaggio di definizione dei dati

- È l'insieme di istruzioni dedicate alla definizione di dati e tabelle

Data Manipulation Language DML - Linguaggio di manipolazione dei dati

- È l'insieme di istruzioni dedicate alla manipolazione di dati, include istruzioni per l'inserimento, la cancellazione e la modifica di dati.

Il Linguaggio SQL: un pò di storia

L'attuale SQL è lo sviluppo di un linguaggio di interrogazione per il DBMS Relazionale **System R,** proposto originariamente presso i laboratori di ricerca dell'IBM nella seconda metà degli anni '70.

- il suo nome era **SEQUEL** (Structured English Query Language)
- nelle evoluzioni successive divenne **SQL**
- Oggetto di un' intensa attività di standardizzazione ad opera sia dell'ANSI che dell'ISO (International Standard Organization).

Il Linguaggio SQL:

I diversi DBMS disponibili presentano delle piccole differenze nella implementazione del Linguaggio SQL, specie rispetto alle funzionalità più innovative.

- L'uniformità è molto maggiore per quanto riguarda le funzionalità più consolidate.
- Molti sistemi mettono a disposizione interfacce amichevoli per l'interrogazione, la definizione e l'aggiornamento dei dati

  (ad esempio, interfacce di natura visuale).
- Tutti consentono, comunque, di utilizzare in modo esplicito (e completo) il linguaggio SQL.

Tipi di dato

Il concetto di **tipo di dato** appartiene ai linguaggi di programmazione e concerne l'astrazione sui dati.

- un **tipo di dato** è costituito:
  - da un insieme di valori
  - da un insieme di operazioni definite sul tipo
- gli elementi che vengono manipolati dall' SQL appartengono tutti ad un tipo di dato (e deve essere dichiarato)
  - quando si dichiara lo schema di una tabella, per ogni colonna occorre specificare il tipo di dato

Tipi di dato

I Tipi di Dato o (Domini) possono essere classificati in:

- **tipi di dato predefiniti**
  - **Numerici**
    - Esatti: rappresentano valori interi e valori decimali in virgola fissa (es. 75,4.5,-4.6...)
    - Approssimati: rappresentano valori reali in virgola mobile (es. $1378e^{-4}$)
  - **Carattere** (singoli caratteri o stringhe, anche di lunghezza variabile)
  - **Temporali**
  - **Bit** (singoli booleani o stringhe)
  - **Booleani** (introdotti in SQL1999)
- **tipi di dato definiti dall'utente**

Tipi di dato predefiniti - Tipi numerici

Tipi numerici esatti:

- **NUMERIC(precisione, scala)**: rappresenta i valori decimali:
  - può contenere numeri (interi o decimali, positivi e negativi)
    - precisione è il numero totale di cifre, da 1 a 38
    - scala è il numero di cifre alla destra del separatore decimale. Varia da -84 a 127
  - Sia dimensione che precisione sono parametri facoltativi: se non vengono esplicitate, la prima è impostata automaticamente a 38.
- **DECIMAL(precisione, scala)**: identico a NUMERIC
- **INTEGER**: Rappresenta i valori interi (senza cifre decimali). Identico a NUMERIC (38). Non è possibile dimensionarlo (cioè specificarne la lunghezza in byte).
- **SMALLINT**: Rappresenta i valori interi.Identico a NUMERIC (38). I valori di questo tipo sono usati per eventuali ottimizzazioni poichè richiedono minore spazio di memorizzazione.

Tipi di dato predefiniti - Tipi numerici

Tipi numerici approssimati:

- **REAL**: rappresenta valori a singola precisione in virgola mobile. La precisione dipende dalla specifica implementazione di SQL.
- **DOUBLE PRECISION:** rappresenta valori a doppia precisione in virgola mobile. La precisione dipende dalla specifica implementazione di SQL.
- **FLOAT**: permette di richiedere la precisione che si desidera (il parametro p indica il numero di cifre da rappresentare). Il formato è FLOAT(p).

Tipi di dato predefiniti – Tipi Caratteri

Tipi Caratteri

- vengono utilizzati per rappresentare stringhe (sequenze) di caratteri, inclusa la rappresentazione a carattere di un numero
- assegnare un valore più lungo della lunghezza specificata nella definizione del tipo di dato costituisce un errore a tempo di esecuzione (run time error)

  - la sintassi per la definizione è:
    - tipo_di_dato(dimensione)
    - esempio: CHAR, CHAR(100),
    - in alcuni casi è obbligatorio indicare la dimensione
    - se non si indica, viene assunto un valore di default

Tipi di dato predefiniti – Tipi Caratteri

Tipi Caratteri:

- CHARACTER (abbreviato **CHAR**): permette di definire stringhe di caratteri di lunghezza fissata n. La specifica è della forma CHAR(n).
  - la dimensione massima è 2000
  - la dimensione minima e di default è1
  - Essendo una lunghezza fissa, nel caso che venga inserita una stringa di *m* caratteri (dove *m* è minore di *n*), il DBMS aggiunge in coda alla stessa *n-m* caratteri ' ' (spazio o blank) in modo da raggiungere la lunghezza massima dichiarata.
- CHARACTER VARYING (abbreviato **VARCHAR**): permette di definire stringhe di caratteri a lunghezza variabile con lunghezza massima predefinita n. La specifica è della forma VARCHAR(n).
  - la dimensione è obbligatoria e va da 1 a 4000
  - se si immette un dato di lunghezza inferiore, non si effettua il blank padding

Tipi di dato predefiniti – Tipi Temporali

Tipi Temporali: permettono di gestire date e ore

Il formato usato è quello anglosassone:

- Data 'anno-mese-giornò
- Ora 'ore:minuti:secondi'

Nella maggior parte dei DBMS date e ore sono inserite sotto forma di stringhe di caratteri, per cui vanno racchiuse tra apici "anno-mese-giorno".

- **DATE**: rappresenta le date espresse come anno (4 cifre), mese (2 cifre comprese tra 1 e 12), giorno (2 cifre comprese tra 1 e 31 con ulteriori restrizioni a seconda del mese).
- **TIME**: rappresenta I tempi espressi come ora (2 cifre), minuti (2 cifre) e secondi (2 cifre),
- **Timestamp**: rappresenta una concatenazione dei due tipi di dato precedenti con una precisione al microsecondo,
  - Sono specificati nella forma 'anno-mese-giorno ore:minuti:secondi.microsecondi'

Tipi di dato predefiniti – Tipi Booleani

Tipi Booleani:

I Tipi Booleani non sono presenti in tutti i DBMS.

Spesso sono sostituiti con un dominio definito dall'utente.

- **BOOLEAN**: rappresenta valori booleani.

  Generalmente un dato booleano occupa soltanto un bit.
  - Esempi di valori booleani:
    - TRUE, FALSE, UNKNOWN, '1', '0'

Il valore NULL

Il **valore NULL** è una speciale costante presente in tutti i DBMS

- Rappresenta l' assenza di informazione
- è compatibile con ogni tipo di dato
- Non va confuso con lo 0, non sono la stessa cosa, anche se internamente possono avere la stessa rappresentazione
- Utilizzato nelle espressioni aritmetiche, restituisce sempre il valore NULL

Il valore NULL

Confronti con NULL

- per verificare se un valore è null, occorre l'operatore IS NULL o IS NOT NULL, non l'operatore di uguaglianza

- se A vale NULL
  - A = 0 è falsa
  - A > 0 è falsa
  - A < 0 è falsa
  - A != 0 è vera
  - **A = NULL è falsa**
  - **A IS NULL è vera**

Tipi di dato definiti dall' utente

Domini:

- La definizione dei Domini fa parte del DDL. La anticipiamo in quanto concernente la definizione dei tipi di dato.
- SQL permette di definire domini e utilizzarli nella definizione delle tabelle.

Un Dominio viene creato tramite istruzione CREATE DOMAIN

Sintassi:

CREATE DOMAIN NomeDominio

AS Tipo **[DEFAULT** value**] [CHECK(** Vincolo**)]**

Esempi:

CREATE DOMAIN TipoEta **AS INTEGER CHECK** (**VALUE BETWEEN** 1 **AND** 120);

CREATE DOMAIN Occupazione **AS VARCHAR(30) DEFAULT** 'Disoccupato' ;

CREATE DOMAIN Voto **AS SMALLINT DEFAULT** NULL **CHECK** (**VALUE** >=18 **AND VALUE** <= 30);

Vincoli d'integrità

Un vincolo è una regola che specifica delle condizioni sui valori delle colonne

di una relazione.

Un vincolo può essere associato ad una tabella o ad un singolo attributo.

- Vincoli intra-relazionali
- Vincoli inter-relazionali
- Vincoli controllo

Vincoli intra-relazionali

Vincoli intra-relazionali

- NOT NULL (su singoli attributi)
- UNIQUE: permette di definire attributi che identificano la tupla:
  - singolo attributo: unique dopo la specifica del dominio
  - più attributi: unique (Attributo,...,Attributo)
- PRIMARY KEY: definizione della chiave primaria (una sola, implica not null);
  - sintassi come per unique
- CHECK: per vincoli complessi

Vincoli inter-relazionali

Vincoli inter-relazionali

- CHECK: per vincoli complessi
- REFERENCES e FOREIGN KEY permettono di definire vincoli di integrità referenziale

Sintassi:

- per singoli attributi:
  - REFERENCES dopo la specifica del dominio
- riferimenti su più attributi:
  - FOREIGN KEY*(Attributo,...,Attributo)* REFERENCES ...

Le Interrogazioni in SQL

Le **interrogazioni** o query di selezione, sono la funzionalità principale di SQL.

- I dati estratti tramite query possono essere:
  - **Valori singoli:** il valore restituito è una tabella formata da un solo record con una sola colonna
  - **Liste:** il valore restituito è una tabella formata da una sola colonna
  - **Tabella**: il risultato della query è una tabella temporanea
- Una query SQL può essere espressa per mezzo dell'**espressione algebrica**

$$\Pi_{(colonna1,\ colonna2,\ldots)}(\sigma_{condizione}(tabella1\ X\ tabella2))$$

Interrogazione base in SQL

Sintassi di base di una interrogazione: **SELECT**

SELECT ListaAttributi
FROM ListaTabelle
[WHERE Condizione**]**

- ListaAttributi (cosa si vuole come risultato) è la lista delle colonne da estrarre, l'ordine e il nome.
- ListaTabelle (da dove si prende) è la lista delle tabelle sulle quali verranno estratti i dati
- Condizione (che condizioni deve soddisfare) definisce una condizione su cui verranno filtrati i record da estrarre

Le Interrogazioni in SQL

NOTE:

- Non è obbligatorio scrivere le clausole su righe separate
- Solo le clausole **SELECT** e **FROM** sono obbligatorie
- In ogni istruzione SQL si possono inserire **commenti**
  - con -- se su singola riga
  - oppure /* in questo modo se si tratta di commenti su più righe */

Clausola SELECT

Clausola SELECT:

SELECT[ALL | DISTINCT]{* | <espressione> [AS<nome>] }[,…]

- Definisce le colonne nel risultato della query e permette anche di rinominare colonne e di generarne di nuove.
- Corrisponde ad un'operazione di proiezione.
- L'operatore * permette di recuperare tutti gli attributi della tabella specificata nella clausola FROM.

Esempio:

Consideriamo la tabella persona(IdPersona, CodiceFiscale, Nome,Cognome, DataNascita)

La seguente query permette di recuperare l'intero contenuto di persona

SELECT * FROM persona;

ed equivale a:

SELECT IdPersona, CodiceFiscale, Nome,Cognome, DataNascita **FROM** persona;

Clausola FROM

Clausola FROM:

FROM <nome tabella> [[AS]<alias>]

- Specifica la tabella o le tabelle su cui si fa la query
- Costruisce la tabella intermedia a partire da una o più tabelle, su cui operano le altre clausole (SELECT, WHERE …)
- La tabella intermedia è il risultato del prodotto cartesiano delle tabelle elencate.

Clausola WHERE

Clausola WHERE:

[WHERE <Condizione>]

- Permette di filtrare le tuple in base ad una condizione
- consiste, nel caso generale, di una espressione logica di predicati
- viene usata la **logica a tre valori**
- i predicati possono essere combinati usando gli operatori logici (AND, OR, NOT), di comparazione, matematici e operatori sulle stringhe
- è opzionale

- Una tupla soddisfa la clausola **WHERE** se e solo se l'espressione risulta vera per tale tupla

Logica dei predicati a tre valori

SQL ricorre alla logica dei predicati a **tre valori**:

- vero (V)
- falso (F)
- sconosciuto (*unknown*, indicato con ?)

Fanno eccezione i predicati IS NULL e IS NOT NULL, il cui valore è sempre o vero o falso, anche se il valore di confronto è sconosciuto

Logica dei predicati a tre valori

Gli operatori Booleani vengono estesi al valore U nel seguente modo:

NOT ? = ?

V AND ? = ?

F AND ? = F

? AND ? = ?

V OR ? = V

F OR ? = ?

? OR ? = ?

In sostanza:

NOT A è vero se e solo se A è falso,

A AND B è vero se e solo se entrambi A e B sono veri

A OR B è vero se e solo almeno uno tra A e B è vero

DB di riferimento per esempi

Consideriamo i seguenti schemi di una base di dati relazionale:

- IMPIEGATI(Matricola, Cognome, Nome, Mansione, IdReparto, StipendioAnnuale, PremioProduzione, DataAssunzione)
- REPARTI(IdReparto, Nome, Indirizzo, Città)

Viene data la seguente istanza della base di dati:

Matricola	Cognome	Nome	Mansione	IdReparto	StipendioAnnuale	PremioProduzione	DataAssunzione
89	Neri	Luca	Dirigente	3	65000	6000	2001-10-27
125	Rossi	Giulia	Analista	2	25000	2000	2012-12-07
129	Rossi	Marco	Sviluppatore	2	40000	5000	2004-05-03
201	Verdi	Paola	Dirigente	3	60000	8000	2006-01-10
300	Bianchi	Mario	Segretario	1	25000	1000	2001-03-14
350	Bianchi	Andrea	Segretario	1	25000	NULL	2001-03-14

IdReparto	Nome	Indirizzo	Città
1	Amministrazione	Via Roma, 5	Milano
2	Sviluppo	Via Padova, 6	Roma
3	Direzione	Piazza Vittorio Emanuele, 23	Milano

Operatori DISTINCT e ALL

Operatori DISTINCT e ALL:

SELECT[ALL | DISTINCT]{* | <espressione> [AS<nome>] }[,…]

- Il risultato di una query SQL può contenere righe duplicate
- Per eliminare I duplicati si usa l'opzione **DISTINCT** nella clausola **SELECT**.
- ALL è il valore di default

Esempio:

1. **SELECT** Mansione **FROM** Impiegati **WHERE** PremioProduzione>1000;
2. **SELECT DISTINCT** Mansione **FROM** Impiegati **WHERE** PremioProduzione>1000;

Risultato Query 1

Mansione
Analista
Dirigente
Sviluppatore
Dirigente

Risultato Query 2

Mansione
Analista
Dirigente
Sviluppatore

Espressioni nella clausola SELECT

SELECT[ALL | DISTINCT]{* | <espressione> [AS<nome>] }[,…]

- Permette di specificare gli attributi che devono comparire nel risultato della query
- Può contenere anche espressioni (numeriche, su stringhe, ecc.)

Esempio:

SELECT Nome, StipendioAnnuale, PremioProduzione, PremioProduzione+StipendioAnnuale
FROM Impiegati **WHERE** Mansione = 'Sviluppatore';

Risultato query:

Nome	StipendioAnnuale	PremioProduzione	
Marco	40000	5000	45000

OSS:

- L' ultima colonna (quella calcolata) non ha un nome

Clausola SELECT: Alias per attributi

SELECT[ALL | DISTINCT]{* | <espressione> [AS <nome>] }[,…]

Ad ogni elemento della Target list (lista di attributi) è possibile associare un nome a piacere.

- la keyword **AS** può anche essere omessa

Esempio:

SELECT Nome, StipendioAnnuale, PremioProduzione, PremioProduzione+StipendioAnnuale **AS** Reddito
FROM Impiegati **WHERE** Mansione = 'Sviluppatore';

oppure

SELECT Nome, StipendioAnnuale, PremioProduzione, PremioProduzione+StipendioAnnuale Reddito
FROM Impiegati **WHERE** Mansione = 'Sviluppatore';

Risultato query:

Nome	StipendioAnnuale	PremioProduzione	Reddito
Marco	40000	5000	45000

Clausola FROM: Alias per tabelle

FROM <nome tabella> [[AS]<alias>]

- Per abbreviare la scrittura si può introdurre uno pseudonimo (*alias*) per la tabella
- Inoltre per referenziare una colonna si può anche fare uso della forma estesa
  - **<nome tabella>.<nome colonna>**

Esempio:

SELECT Imp.matricola **FROM** impiegati **AS** Imp **WHERE** Imp.Nome = 'Paola';

oppure

SELECT impiegati.matricola **FROM** impiegati **WHERE** impiegati.Nome = 'Paola';

OSS: L'Alias non è obbligatorio

Operatori sulle stringhe

Concatenazione di stringhe: **Operatore ||**

L'utilizzo dell' operatore || all'interno di un'espressione nella clausola SELECT permette di concatenare campi di tipo Carattere

Esempio:

SELECT cognome || ' ' || nome **AS** CognomeNome **FROM** impiegati;

Cognome Nome
Rossi Giulia
Bianchi Mario
Verdi Paola
Rossi Marco
Neri Luca

Operatori sulle stringhe

Confronto tra stringhe: **Operatore LIKE**

- L' **operatore LIKE** permette di trovare stringhe che soddisfano un certo "pattern" mediante le "wildcard"
  - _ (corrisponde a un carattere arbitrario)
  - % (corrisponde a una stringa arbitraria)
- L'espressione è vera se il pattern fà match con la stringa
- Esempi:
  - 'M%A' fà match con 'MA', 'MAMMA', ecc.
  - '%MA%' fà match con qualsiasi stringa che contiene la sottostringa 'MA', ecc
  - "M_R%' fà match con MARIO, MARCO ma **NON** con Mario e Marco (case sensitive fa distinzione tra le lettere maiuscole e quelle minuscole)

Esempio:

Impiegati il cui nome finisce con 'o'

SELECT * **FROM** Impiegati **WHERE** Nome **LIKE** '%o';

Impiegati il cui nome inizia per 'M' e hanno il carattere r in terza posizione

SELECT * **FROM** Impiegati **WHERE** Nome **LIKE** 'M_r%';

Operatori di confronto

Operatore BETWEEN

- L'operatore **BETWEEN** permette di esprimere condizioni di appartenenza ad un intervallo (estremi inclusi)

Esempio:

Impiegati il cui stipendio annuale è compreso tra 40000 e 65000:

SELECT * **FROM** impiegati **WHERE** stipendioannuale **BETWEEN** 40000 **AND** 65000**;**

SELECT * **FROM** impiegati **WHERE** stipendioannuale **>=** 40000 **AND** stipendioannuale <= 65000**;**

Matricola	Cognome	Nome	Mansione	IdReparto	StipendioAnnuale	PremioProduzione	DataAssunzione
89	Neri	Luca	Dirigente	3	65000	6000	2001-10-27
129	Rossi	Marco	Sviluppatore	2	40000	5000	2004-05-03
201	Verdi	Paola	Dirigente	3	60000	8000	2006-01-10

Operatori di confronto

Operatori IN e NOT IN

- L'operatore **IN** permette di esprimere condizioni di appartenenza ad un insieme di valori
- L'operatore **NOT IN** permette di esprimere condizioni di NON appartenenza ad un insieme di valori

Esempio:

Impiegati che hanno uno stipendio annuale di 40000 e 65000

SELECT * **FROM** impiegati **WHERE** stipendioannuale **IN** (40000,65000);

è equivalente a:

SELECT * **FROM** impiegati **WHERE** stipendioannuale **=** 40000 **OR** stipendioannuale **=** 65000;

SELECT * **FROM** impiegati **WHERE** stipendioannuale **NOT IN** (40000,65000);

è equivalente a:

SELECT * **FROM** impiegati **WHERE** stipendioannuale **!=** 40000 **AND** stipendioannuale **!=** 65000;

Operatori sui valori NULL

Operatori IS NULL e IS NOT NULL

Il valore NULL rappresenta assenza di informazione.

Esempio:

SELECT * **FROM** Impiegati **WHERE** IdReparto <3 **AND** PremioProduzione **> 6000**;

Risultato:

Nessuna tupla

SELECT * **FROM** Impiegati **WHERE** PremioProduzione **IS NULL**;

Dà come risultato:

Matricola	Cognome	Nome	Mansione	IdReparto	StipendioAnnuale	PremioProduzione	DataAssunzione
350	Bianchi	Andrea	Segretario	1	25000	NULL	2001-03-14

Ordinamento del risultato di una query

[ORDER BY <espressione> [ASC | DESC] [,...]]

La clausola **ORDER BY** si usa per ordinare il risultato di una query secondo i valori di una o più colonne

- Per ogni colonna si specifica se l'ordinamento è per valori ascendenti (operatore **ASC**) o discendenti (operatore **DESC**)
- Valore di default è **ASC**

Esempio:

SELECT Cognome, Nome **FROM** Impiegati **ORDER BY** Cognome **ASC**, Nome **DESC**;

è equivalente a:

SELECT Cognome, Nome **FROM** Impiegati **ORDER BY** Cognome, Nome **DESC**;

Nota: ordina prima rispetto a Cognome in modo crescente, poi sulle tuple che hanno Valori uguali per Cognome ordina rispetto a Nome in modo decrescente

Funzioni di aggregazione

Le funzioni di aggregazione permettono di calcolare valori aggregando più tuple Secondo particolari criteri:

- Il numero di elementi (**COUNT**)
- Il minimo (**MIN**)
- Il massimo (**MAX**)
- La media (**AVG**)
- Il totale (**SUM**)

Sintassi:

Funzione ([**DISTINCT**] EspressioneSuAttributi)

Funzioni di aggregazione

COUNT

Sintassi:

COUNT(*)

- conta il numero di tuple

COUNT(Attributo)

- conta i valori di un attributo (considerando i duplicati)

COUNT(**DISTINCT** Attributo)

- conta i valori distinti di un attributo

COUNT e valori NULL

SELECT COUNT(*) **FROM** Impiegati;

Risultato: numero di tuple

SELECT COUNT(PremioProduzione) **FROM** Impiegati;

Risultato: numero di valori diversi da NULL

SELECT COUNT(**DISTINCT** PremioProduzione) **FROM** Impiegati;

Risultato: numero di valori distinti (esclusi i NULL)

Funzioni di aggregazione

SUM, AVG, MAX e MIN

- ammettono come argomento un attributo o un'espressione (ma non "*")
- **SUM** e **AVG**: argomenti numerici o tempo
- **MAX** e **MIN**: argomenti su cui è definito un ordinamento

Esempi:

- Trovare dei premi produzione il più basso, il più alto, la media e il totale

SELECT MIN(PremioProd) **AS** MinorPremio, **MAX**(PremioProd) **AS** MaggiorPremio,
AVG(PremioProd) **AS** MediaPremi, **SUM**(PremioProd) **AS** TotalePremi

FROM Impiegati;

- Trovare il premio produzione più alto del reparto 'Sviluppo'

SELECT MAX(PremioProd) **AS** MaxPremioSviluppo

FROM Impiegati, Reparti

WHERE Reparti.Nome = 'Sviluppo' **AND** Impiegati.IdReparto = Reparti.IdReparto;

Raggruppamento di tuple

Clausola GROUP BY

La clausola GROUP BY consente di realizzare l'operazione di aggregazione o Raggruppamento.

[GROUP BY <ListaAttributiDiRaggruppamento**>]**

Esempio:

Fornire la somma degli stipendi degli impiegati di ciascun Reparto.

SELECT IdReparto**, SUM(**StipAnnuale**) FROM** impiegati**;** **è ERRATA!**

Query corretta:

SELECT IdReparto**, SUM(**StipAnnuale**) FROM** impiegati **GROUP BY** IdReparto**;**

NOTA: Le colonne usate per il raggruppamento sono le sole che possono apparire anche nella clausola **SELECT** e **ORDER BY**.

Raggruppamento di tuple

Clausola HAVING

La clausola HAVING può essere usata correttamente solo in combinazione con la Clausola GROUP BY.

- Permette di specificare condizioni sui gruppi di tuple generati dalla clausola GROUP BY in modo da selezionare solo i gruppi che rispettano certi criteri.

Esempio:

Determinare i reparti che spendono più di 4000 in premi produzione

```
SELECT IdReparto, SUM(PremioProduzione) AS SommaPremi
FROM impiegati
GROUP BY IdReparto
HAVING SUM(PremioProduzione) > 4000;
```

Join tra tabelle

L'operatore di **Join** permette di effettuare la ricerca di informazioni mettendo in relazione i dati presenti in tabelle diverse;

- consideriamo due tabelle che abbiano colonne contenenti dati in comune. Se due tabelle soddisfano questa condizione possono essere correlate tramite una operazione di join.

- le colonne devono rappresentare lo stesso insieme di entità
  - In termini algebrici, **i valori delle colonne delle due tabelle devono appartenere allo stesso dominio.**

Join tra tabelle

L'operazione di Join permette di correlare dati rappresentati da relazioni diverse.

- è espressa in SQL tramite un prodotto cartesiano a cui sono applicati uno o più predicati di join
- il **predicato di join** esprime una relazione che deve essere verificata dalle tuple risultato della query

L'operazione di join si **interpreta** nel seguente modo:

- Nella clausola **FROM** si genera il prodotto Cartesiano delle tabelle coinvolte
- Nella clausola **WHERE** si applicano i predicati
- Nella clausola **SELECT** si estraggono le colonne specificate nella Target List

Equi-Join

La forma più utile e immediata della operazione di join è la **equi-join** tra due tabelle.

Questa operazione realizza l'operazione di giunzione naturale definita nell'algebra relazionale

- restituisce una terza tabella le cui righe sono tutte e sole quelle ottenute dalle righe delle due tabelle di partenza in cui i valori delle colonne (attributi) in comune sono uguali.

Esempio:

Selezionare di ogni impiegato il Nome, Cognome e la città in cui lavora:

SELECT imp.Nome, imp.Cognome, rep.Citta

FROM impiegati imp, reparti rep

WHERE imp.IdReparto = rep.IdReparto;

- imp.IdReparto = rep.IdReparto è il predicato di join

Inner Join

Per la inner-join la condizione di join non deve necessariamente essere una condizione di uguaglianza.
E' possibilire effettuare join tra colonne espresse sullo stesso dominio, seppure utilizzate per rappresentare informazioni non esplicitamente correlate.

NOTA: la equi-join è un caso particolare di Inner Join

Esempio:

Selezionare gli impiegati che lavorano a Milano:

SELECT imp.*
FROM impiegati imp, reparti rep
WHERE imp.IdReparto = rep.IdReparto **AND** citta = 'Milano';

Self Join

Il Self Join

- è il join di una tabella con se stessa
  - occorre rinominare le tabelle in **FROM**

Esempio:

trovare per ogni impiegato il nome del suo responsabile

- il risultato "non appartiene" allo schema della tabella, ma al prodotto della tabella con se stessa

SELECT imp1.nome || ' works for ' || imp2.nome "Impiegati e loro Responsabili"
FROM impiegati imp1, impiegati imp2
WHERE imp1.Responsabile = imp2.Matricola**;**

Join espliciti

- Negli esempi visti finora abbiamo utilizzato la forma tradizionale di scrittura di una join mediante l'uso di predicati di join nella clausola WHERE.
- Nelle ultime revisioni allo standard sono stati introdotti, per specificare il tipo di join, dei **costrutti ad hoc**

(Nota: non sono supportati da tutti i DBMS commerciali)

Tipi di join **espliciti** sono:

- **INNER JOIN**
- **CROSS JOIN (prodotto Cartesiano)**
- **OUTER JOIN**

Inner Join

Esempio:

Selezionare di ogni impiegato il Nome, Cognome e la città in cui lavora:

SELECT imp.Nome, imp.Cognome, rep.Citta
FROM impiegati imp, reparti rep
WHERE imp.IdReparto = rep.IdReparto;

è equivalente a:

SELECT imp.Nome, imp.Cognome, rep.Citta
FROM impiegati [AS] imp **INNER JOIN** reparti [AS] rep **ON** imp.IdReparto = rep.IdReparto;

è equivalente a:

SELECT imp.Nome, imp.Cognome, rep.Citta
FROM impiegati [AS] imp **NATURAL INNER JOIN** reparti [AS] rep ;

è equivalente a:

SELECT imp.Nome, imp.Cognome, rep.Citta
FROM impiegati [AS] imp **INNER JOIN** reparti [AS] rep **USING** IdReparto

Cross Join

Operatore CROSS JOIN

- L'operazione di prodotto tra due tabelle, definita nell'algebra relazionale, si ottiene mediante una operazione di join espressa in maniera tradizionale in cui non venga specificata la condizione di join.
- Se le due tabelle sono composte rispettivamente da n e da m righe, la nuova tabella conterrà nxm righe, ottenute accodando ciascuna riga della prima tabella con ciascuna riga della seconda.

Le seguenti espressioni sono **equivalenti** in SQL:

```
1  SELECT tabella1.*, tabella2.* FROM tabella1, tabella2;
2  SELECT tabella1.*, tabella2.* FROM tabella1 CROSS JOIN tabella2;
3  SELECT tabella1.*, tabella2.* FROM tabella1 JOIN tabella2 ON TRUE;
4  SELECT tabella1.*, tabella2.* FROM tabella1 INNER JOIN tabella2 ON TRUE;
```

Outer Join

Il **join esterno (outer join)** prevede che tutte le tuple diano un contributo al risultato, estendendo con valori nulli le tuple che non hanno una corrispondenza.

La outer-join può mantenere i valori per cui non esiste corrispondenza per una, per l'altra o per entrambe le tabelle; i tre casi vengono rispettivamente definiti outer-join sinistra, outer-join destra, outer-join completa.

- Il **join sinistro** estende le tuple del primo operando
- Il **join destro** estende le tuple del secondo operando
- Il **join completo** le estende tutte

In SQL l'operatore di **OUTER JOIN** può essere introdotto esplicitamente nella clausola FROM del comando SELECT utilizzando i costrutti:

- **{LEFT|RIGHT|FULL} [OUTER] JOIN** + **ON** <predicato>
- **{ LEFT|RIGHT|FULL} [OUTER] JOIN** + **USING** <colonne>
- **NATURAL { LEFT|RIGHT|FULL} [OUTER] JOIN**

NOTA: OUTER è opzionale

Operatori insiemistici

Operatori insiemistici: UNION, INTERSECT, EXCEPT

- L'istruzione **SELECT** non permette di eseguire unione, intersezione e differenza di tabelle
- si può invece utilizzarli per combinare in modo opportuno i risultati di due istruzioni SELECT

UNION (unione)

- L'operatore **UNION** realizza l'operazione di unione definita nell'algebra relazionale.
- Gli operandi sono due tabelle risultanti da comandi SELECT e restituisce come risultato una terza tabella che contiene tutte le righe della prima tabella e tutte le righe della seconda tabella.
- per poter avere l' unione occorre che gli schemi delle due relazioni (tabelle) siano "compatibili"
  - stesso numero di argomenti
  - stesso tipo degli argomenti
  - non è richiesto che abbiano gli stessi nomi

NOTA: Il risultato è di default privo di duplicati. Per ottenerli occorre aggiungere l'opzione **ALL**.

Operatori insiemistici

INTERSECT (intersezione)

- L'operatore **INTERSECT** realizza l'operazione di intersezione definita nell'algebra relazionale.
- Gli operandi sono due tabelle risultanti da comandi SELECT e restituisce come risultato una terza tabella che contiene tutte le righe che compaiono sia nella prima tabella che nella seconda.

NOTA: Il risultato è di default privo di duplicati. Per ottenerli occorre aggiungere l'opzione **ALL**.

EXCEPT (differenza)

- L'operatore **EXCEPT** realizza l'operazione di differenza definita nell'algebra relazionale.
- Gli operandi sono due tabelle risultanti da comandi SELECT e restituisce come risultato una terza tabella che contiene tutte le righe della prima tabella che non si trovano nella seconda.

NOTA: Il risultato è di default privo di duplicati. Per ottenerli occorre aggiungere l'opzione **ALL**.

Subquery

In SQL è possibile esprimere delle condizioni che si basano sul risultato di altre interrogazioni (**subquery**, o query innestate).

- il risultato di una query (ossia di una select) effettuata su di un certo numero di relazioni è una relazione. Nel caso più semplice è il valore di uno specifico attributo.

- con dovute eccezioni, nella costruzione di uno statement SQL dove compare una relazione o una colonna si può far comparire il risultato di una select

Esempio:

Trovare Nome e Cognome degli impiegati che lavorano nel reparto di Marco

SELECT Nome, Cognome
FROM impiegati
WHERE IdReparto = (**SELECT** IdReparto **FROM** impiegati **WHERE** Nome = 'Marco');

Subquery

Esempio:

Trovare le matricole degli impiegati dei reparti con sede a Milano.

```
SELECT Matricola
FROM impiegati
WHERE IdReparto IN (SELECT IdReparto
                    FROM reparti
                    WHERE Citta = 'Milano');
```

La subquery restituisce l'insieme di reparti (1, 3)

La clausola **WHERE** equivale a:

- **WHERE** IdReparto **IN** (1,3);

Subquery: operatori ANY e ALL

Utilizzo degli operatori ANY e ALL

Gli operatori di confronto **=, <, >…** si possono usare solo se la subquery restituisce non più di una tupla.

Se la subquery restituisce più di una tupla si devono usare gli operatori:

- <op> ANY: <op> vale per almeno uno dei valori
- <op> ALL: <op> vale per tutti i valori

ANY: la tupla soddisfa la condizione se il confronto (con l'operatore specificato) tra il valore che l'attributo/i assume sulla tupla e almeno uno degli elementi restituiti dall' interrogazione nidificata risulta VERO.

ALL: la tupla soddisfa la condizione se il confronto (con l'operatore specificato) tra il valore che l'attributo/i assume sulla tupla e ciascuno degli elementi restituiti dall' interrogazione nidificata risulta VERO.

NOTE:

- <op> è un operatore di confronto
- La forma **= ANY** equivale a **IN**

Subquery: operatore EXISTS

Operatore Exists

Mediante **EXISTS** (SELECT * …) è possibile verificare se il risultato di una subquery restituisce almeno una tupla

```
SELECT IdReparto
FROM Reparti
WHERE EXISTS (SELECT *
              FROM Impiegati
              WHERE Mansione = 'Programmatore');
```

Facendo uso di **NOT EXISTS** il predicato è vero se la subquery non restituisce alcuna Tupla.

Riepilogando

Operatore di Proiezione ($\Pi_{(A1,A2,\ldots,An)}(relazione)$ **)**

L'operatore di proiezione si traduce in SQL come:

SELECT DISTINCT A1,…,An **FROM** relazione;

Operatore di Selezione ($\sigma_{(predicato)}(relazione)$ **)**

L' operatore di selezione si traduce in SQL come:

SELECT * **FROM** relazione **WHERE** predicato;

Operatore di Prodotto Cartesiano (relazione1 × relazione2)

L' operatore di prodotto cartesiano si può tradurre in SQL mediante la query:

SELECT * **FROM** relazione1, relazione2;

Operatore di Join:

Relazione1 $\bowtie_{exp}$ Relazione2 è equivalente a $\sigma_{exp}(relazione1 \times relazione2)$

e si può tradurre in SQL mediante la query:

SELECT * **FROM** relazione1, relazione2 **WHERE** exp;