

Il linguaggio SQL

Introduzione e costrutti di interrogazione

c.vallati@iet.unipi.it

Sommario

- Introduzione al linguaggio SQL
- I costrutti di interrogazione del linguaggio SQL

Il Linguaggio SQL

- Il linguaggio SQL (Structured Query Language) è di fatto lo standard tra i linguaggi per la gestione di basi di dati relazionali
- Standardizzato attraverso le attività di comitati di standardizzazione internazionali quali l'ANSI e l'ISO, SQL è oggi adottato dalla maggior parte dei sistemi DBMS commerciali e open-source
- Una serie successive di versioni del linguaggio sono state standardizzate nel tempo, ciascuna di queste nuove versioni ha introdotto nuovi costrutti di utilità in maniera incrementale mantenendo la retrocompatibilità
- Ciascuna singola implementazione può includere delle funzioni del linguaggio custom che introduce delle funzionalità particolari non standard (non compatibili quindi con altri prodotti)

Il Linguaggio SQL

- SQL è un linguaggio **dichiarativo** (non-procedurale), ovvero non specifica la sequenza di operazioni da compiere per ottenere il risultato
- SQL si dice essere “relazionalmente completo”, perché ogni espressione dell'algebra relazionale può essere tradotta in una corrispondente espressione in SQL
- Il linguaggio SQL contiene funzionalità di :
 - **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
 - **Data Definition Language DDL** - Linguaggio di definizione dei dati, è l'insieme di istruzioni dedicate alla definizione di dati e tabelle
 - **Data Control Language DCL** – Linguaggio di controllo degli accessi, è l'insieme di istruzioni che permette di definire i permessi necessari per accedere alle tabelle della base di dati

Tipi di dato

- Alla base del modello relazionale vi sono le tabelle con i relativi attributi. Ogni attributo appartiene ad un dominio che deve essere ben definito
- Al fine di definire i domini degli attributi per la creazione di tabelle (gli elementi che vengono manipolati dall' SQL appartengono tutti ad un tipo di dato e questo deve essere dichiarato) e per la loro interrogazione, SQL definisce vari tipi di dato
- Un **tipo di dato** è costituito:
 - da un insieme di valori
 - da un insieme di operazioni definite sul tipo

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
 - **Carattere** (singoli caratteri o stringhe, anche di lunghezza variabile)
 - **Temporal**
 - **Bit** (singoli booleani o stringhe)
 - **Booleani** (introdotti in SQL1999)
 - **Tipi di dato definiti dall'utente a partire dai domini elementari**

Tipi numerici esatti

- Questa famiglia contiene i domini numerici che permettono di rappresentare valori numerici esatti, es. numeri interi o numeri con parte decimale di lunghezza prefissata (e.g. valori monetari)
 - **NUMERIC(precisione, scala)**: rappresenta i valori decimali (interi o decimali, positivi e negativi):
 - **Precisione** è il numero totale di cifre
 - **Scala** è il numero di cifre alla destra del separatore decimale
 - Sia dimensione che precisione sono parametri facoltativi: se non vengono esplicitate, sono impostati automaticamente
 - **DECIMAL(precisione, scala)**: identico a NUMERIC, con l'unica differenza che la precisione rappresenta il valore minimo con cui poi i numeri verranno memorizzati
 - **INTEGER**: Rappresenta i valori interi (senza cifre decimali). Non è possibile dimensionarlo (cioè specificarne la lunghezza in byte).
 - **SMALLINT**: Rappresenta i valori interi. I valori di questo tipo sono usati per eventuali ottimizzazioni poichè richiedono minore spazio di memorizzazione.

Tipi numerici approssimati

- Per le grandezze reali approssimate invece (es. grandezze fisiche) SQL definisce:
 - **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 caratteri

- SQL definisce diversi tipi carattere 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 carattere

- **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 è 1
 - 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) 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 temporali

- SQL definisce dei tipi temporali per permettere la gestione di date e ore
 - Il formato usato è quello anglosassone:
 - Data 'anno-mese-giorno'
 - 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 booleani

- SQL definisce un tipo BOOLEAN per rappresentare appositamente i valori booleani
 - Generalmente un dato booleano occupa soltanto un bit.
 - Esempi di valori booleani:
 - TRUE, FALSE, UNKNOWN, '1', '0'

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
- Confronti con NULL:
 - Per verificare se un valore è NULL si deve usare l'operatore IS NULL o IS NOT NULL e non l'operatore di uguaglianza
 - se A vale NULL
 - $A = 0$ è equivalente a falsa
 - $A > 0$ è equivalente a falsa
 - $A < 0$ è equivalente a falsa
 - $A \neq 0$ è equivalente a falsa
 - **$A = \text{NULL}$ è equivalente a falsa**
 - **$A \text{ IS NULL}$ è vera**

Tipi di dato definiti dall'utente

- 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);**

Interrogazioni in SQL

- Le **interrogazioni** o query di selezione, sono la funzionalità principale di SQL. L'SQL esprime le interrogazioni in modo dichiarativo, cioè specifica l'obiettivo dell'interrogazione e non il modo con cui ottenerlo
- I dati estratti tramite una 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
- La struttura generale di una query SQL può essere espressa per mezzo dell'**espressione algebrica**:

$\Pi_{(colonna1, colonna2, \dots)} (\sigma_{condizione} (tabella1 \times tabella2))$

Query Optimizer

- Le interrogazioni espresse dall'utente programmatore prima di essere eseguite vengono analizzate dal DBMS e ottimizzate
- In particolare un componente, il Query Optimizer, analizza l'interrogazione e formula un'interrogazione equivalente nel linguaggio procedurale interno al sistema (che rimane comunque nascosto all'utente)
- In questo processo il query optimizer ottimizza la richiesta applicando tecniche di ottimizzazione standard
- Esistono in generale molti modi diversi per esprimere la stessa interrogazione in SQL, ma grazie all'azione del query optimizer l'utente deve solo preoccuparsi della leggibilità e modificabilità della query senza preoccuparsi della sua ottimizzazione

Interrogazione – Sintassi Base

- La sintassi di base di una interrogazione è la seguente:

```
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

Sintassi Base – 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

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

- Definisce le colonne nel risultato della query, ciascuna colonna può essere rinominata. In alcuni casi particolari permette anche di generarne di nuove
- Corrisponde ad un'operazione di proiezione e ridenominazione
- L'operatore * permette di recuperare tutti gli attributi della tabella specificata nella clausola FROM.
- La parola chiave DISTINCT e ALL regola come si comporterà il sistema nel caso di tuple duplicate nel risultato, nel caso ALL (comportamento di default) i duplicati vengono inclusi, nel caso di DISTINCT i duplicati vengono eliminati

Esempio:

- Tabella: **persona(IdPersona, CodiceFiscale, Nome,Cognome, DataNascita)**
- **La seguente query permette di recuperare l'elenco dei codici fiscali**

SELECT CodiceFiscale **FROM** persona;

- 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 SELECT

- La clausola AS permette di *rinominare* un attributo
- Nella clausola SELECT possono comparire anche *generiche espressioni* sul valore degli attributi al fine di manipolarne il valore

Esempio:

- Tabella **impiegato(Nome, Cognome, CodiceFiscale, StipendioAnnuale)**
- La seguente query permette di ottenere una tabella che elenca i codici fiscali e rinominando la colonna in CF

SELECT CodiceFiscale AS CF FROM Impiegato;

- La seguente query permette di calcolare lo stipendio mensile di ciascun impiegato

**SELECT Nome, Cognome, StipendioAnnuale/12 AS StipendioMensile
FROM Impiegato;**

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
- Tramite la clausola AS si può specificare un alias della tabella per convenienza (l'alias non è obbligatorio)

Clausola FROM

Voli(Codice, Data, Comandante)

Codice	Data	Comandante
AZ427	21/07/2001	Bianchi
AZ427	23/07/2001	Rossi
TW056	21/07/2001	Smith

Linee(Codice, Partenza, Arrivo)

Codice	Partenza	Arrivo
AZ427	FCO	JFK
TW056	LAX	FCO

SELECT *
FROM Voli AS V, Linee AS L;

V.Codice	Data	Comandante	L.Codice	Partenza	Arrivo
AZ427	21/07/2001	Bianchi	AZ427	FCO	JFK
AZ427	23/07/2001	Rossi	AZ427	FCO	JFK
TW056	21/07/2001	Smith	AZ427	FCO	JFK
AZ427	21/07/2001	Bianchi	TW056	LAX	FCO
AZ427	23/07/2001	Rossi	TW056	LAX	FCO
TW056	21/07/2001	Smith	TW056	LAX	FCO

Per evitare ambiguità gli attributi con lo stesso nome vengono identificati nella tabella risultante attraverso *NomeTabella.Attributo* (oppure usando l'alias definito)

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 predicati a tre valori

- SQL ricorre alla logica dei predicati a **tre valori**: vero (V), falso (F), sconosciuto (*unknown*, indicato con U)
- Qualsiasi confronto logico con un valore che è NULL porta a UNKNOWN come risultato
- 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
- Condizioni di selezione per cui il valore è UNKNOWN vengono comunque scartate

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 se almeno uno tra A e B è vero

(F, false; U, unknown; T, true)

NOT(A)		AND(A, B)				OR(A, B)					
		A ∧ B		B		A ∨ B		B			
A	¬A	A	A ∧ B	F	U	T	A	A ∨ B	F	U	T
F	T			F	F	F			F	F	
U	U			U	F	U			U	U	
T	F			T	F	U			T	T	

Clausola WHERE

Impiegati(Nome, Cognome, CodiceImpiegato, Reparto)

Nome	Cognome	CodiceImpiegato	Reparto
Carlo	Rossi	0210	Amministrazione
Mario	Bindi	0231	Produzione
Michele	Sassi	0214	Spedizioni

```
SELECT Nome, Cognome  
FROM Impiegati  
WHERE Reparto = 'Amministrazione  
OR Reparto = 'Produzione'';
```

Nome	Cognome
Carlo	Rossi
Mario	Bindi

DB di esempio

Consideriamo i seguenti schemi di una base di dati relazionale:

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

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

SELECT - Espressioni

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

Nota: L'ultima colonna non ha un nome

SELECT - Alias

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

FROM - Alias

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';

NOTA: L'Alias non è obbligatorio

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 ha un match con la stringa
- Esempi:
 - 'M%A' farà match con 'MA', 'MAMMA', ecc.
 - '%MA%' farà match con qualsiasi stringa che contiene la sottostringa 'MA', ecc
 - "M_R%" farà 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%';
```

Confronto numerico

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

Appartenenza

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;
```

Controllo valori NULL

Operatori IS NULL e IS NOT NULL

Il valore NULL rappresenta assenza di informazione.

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

Dà come risultato:

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

Ordinamento dei risultati

[**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:

SELECT Funzione ([DISTINCT] EspressioneSuAttributi)

COUNT

- Permette di contare il numero di tuple della tabella risultante

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 (i duplicati non sono considerati)

COUNT non conta i 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)

SUM, AVG, MAX e MIN

- Permettono di aggregare i dati delle tuple
- 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 su sottoinsiemi delle tuple
- Le tuple vengono raggruppate secondo un criterio dato, e poi su ciascun sottoinsieme viene calcolato il valore aggregato

[**GROUP BY** <ListaAttributiDiRaggruppamento>]

Esempio:

Fornire la somma degli stipendi degli impiegati di ciascun Reparto.

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**.

La seguente query invece è ERRATA!

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

Le funzioni di aggregazione senza la clausola GROUP BY non possono essere usate insieme agli attributi normali!

Raggruppamento di Tuple - Selezione

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. (Nota, la condizione è una condizione sul gruppo!)

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

- L'operatore di **Join** permette di effettuare la ricerca di informazioni mettendo in relazione i dati presenti in tabelle diverse
- L'algebra razionale definisce una serie di varianti dell'operatore derivato JOIN: Join Naturale, Join Esterno, Equi-Join, Theta-Join
- Nonostante i vari tipi di operazioni di JOIN sono stati standardizzati in SQL nelle varie versioni, non tutti i DBMS supportano esplicitamente le operazioni di JOIN **tramite una singola operazione atomica con costrutti definiti ad hoc**
- **Essendo comunque l'operazione di JOIN un'operazione derivata questa può essere effettuata tramite una composizione delle operazioni di base**
- **Le varie operazioni di Join possono essere implementate con particolari configurazioni delle interrogazioni di base:**
 - Nella clausola **FROM** si genera il **prodotto Cartesiano** delle tabelle coinvolte
 - Nella clausola **WHERE** si applicano i **predicati** dell'operazione di JOIN
 - Nella clausola **SELECT** si estraggono le colonne

Join Naturale/Natural Join

- Il Join naturale presuppone che alcuni degli attributi delle due tabelle abbiano lo stesso nome e dominio coerente
- Si realizza imponendo nella WHERE le condizioni di uguaglianza tra tutti gli attributi che hanno lo stesso nome tra le due tabelle:

```
SELECT * FROM impiegati AS imp, reparti AS rep WHERE  
imp.IdReparto = rep.IdReparto;
```

- La sintassi del metodo diretto è la seguente:

```
SELECT * FROM impiegati NATURAL JOIN reparti;
```

Join Interno/Inner Join

- In alcune implementazioni il Natural Join non è *direttamente* supportato in quanto la comparazione automatica degli attributi (senza controllo da parte dell'utente) può produrre risultati a volte indesiderati
- Solitamente è invece supportata un costrutto join generico chiamato **inner join** (in contrapposizione al join esterno o outer join) attraverso il quale possono essere implementate tutte le varianti (theta-join, equi-join e anche il natural join)

Join Interno/Inner Join

- L'inner join è un join generico interno basato sull'applicazione di una selezione sulla tabella frutto di un prodotto cartesiano
- La condizione di selezione non deve essere necessariamente d'uguaglianza
- In algebra relazionale è l'equivalente del theta-join, se la condizione della selezione è d'uguaglianza allora l'inner join si riduce all'equi-join
- E' equivalente a aggiungere una WHERE con le condizioni della selezione da applicare

Esempio:

Selezionare gli impiegati che lavorano a Milano:

```
SELECT imp.*
```

```
FROM impiegati imp, reparti rep
```

```
WHERE imp.IdReparto = rep.IdReparto AND citta = 'Milano';
```

Join Interno/Inner Join

- La sintassi esplicita per l'Inner Join è la seguente:

SELECT * FROM t1 INNER JOIN t2 ON condizione

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;
```

NOTA: il costrutto INNER JOIN può essere riferito direttamente usando JOIN

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

Equi-Join

- Nel caso la condizione sia di uguaglianza la sintassi esplicita può essere :

SELECT * FROM t1 INNER JOIN t2 USING(attributi)

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 USING(IdReparto)
```

Join Esterno/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

Join Esterno/Outer Join

- In SQL l'operatore di OUTER JOIN può essere applicato usando la seguente sintassi :
 - **{LEFT|RIGHT|FULL} [OUTER] JOIN + ON <predicato>**
 - **{ LEFT|RIGHT|FULL} [OUTER] JOIN + USING <colonne>**
- La parola *outer* è opzionale

Esempio:

```
SELECT * FROM impiegati RIGHT JOIN reparti USING(IdReparto);
```

Anche se in questo caso non cambia nulla, visto che non ci sono tuple appese

Self Join

- E' un inner join di una tabella con se stessa
 - Occorre rinominare le tabelle in **FROM**

Esempio:

- trovare per ogni impiegato il nome del suo responsabile (supponiamo che la tabella impiegati dell'esempio abbia anche un attributo responsabile, che è la matricola dell'impiegato che è il suo responsabile)
 - il risultato “non appartiene” allo schema della tabella, ma al prodotto della tabella con se stessa

```
SELECT imp1.nome AS Impiegato, imp2.nome AS Responsabile  
FROM impiegati imp1, impiegati imp2  
WHERE imp1.Responsabile = imp2.Matricola;
```

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** (la select di default invece lascia i duplicati). Per ottenerli occorre aggiungere l'opzione **ALL**.

Esempio: Recuperare i nomi degli impiegati e dei reparti insieme con i relativi codici

```
SELECT IdReparto, Nome FROM azienda.impiegati UNION ALL SELECT IdReparto, NomeReparto FROM  
azienda.reparti;
```

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 query che contengono al loro interno delle altre interrogazioni, dette **subquery**, o query innestate.
- Dato che risultato di una query effettuata su di un certo numero di relazioni è una relazione, una subquery può introdotta ogni volta che nella query compare una relazione.
- Nel caso di subquery il sistema prima esegue le query innestate e poi esegue quella esterna usando il risultato della query innestata.
- Nel caso in cui la subquery abbia delle dipendenze (cioè nella subquery compaiano degli attributi della query esterna) questa viene rieseguita per ogni tuple della query esterna

Subquery in WHERE

- Le subquery possono essere usate nella clausola WHERE per esprimere delle condizioni complesse
- Il caso piu' semplice e' quello in cui la subquery restituisce un valore singolo (una tabella con un solo attributo e una sola tupla)
- Questa puo' essere usata come un valore scalare ad esempio per verificare una condizione

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 e operatore IN

- Nel caso in cui la subquery restituisca una tabella con piu' di una tupla questa puo' essere usata nella clausola WHERE insieme con l'operatore IN
- **La clausola IN usata con una subquery serve per verificare se il valore di un attributo per una determinata tupla è presente nella 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 e 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 e operatori ANY e ALL

- Selezionare I nomi dei reparti che in cui esiste almeno un impiegato con stipendio maggiore di 30000

SELECT Nome

FROM reparti

WHERE IdReparto =ANY

(SELECT IdReparto FROM impiegati WHERE
StipendioAnnuale > 30000)

Subquery e operatori ANY e ALL

- Selezionare tutti gli impiegati che prendono uno stipendio annuale maggiore di tutti i lavoratori del reparto Sviluppo

SELECT Nome

FROM impiegati

WHERE StipendioAnnuale >ALL

(SELECT StipendioAnnuale FROM impiegati WHERE
IdReparto = 2)

Subquery e operatore Exist

- 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.

Subquery complesse

- Nella maggioranza dei casi le subquery sono eseguite prima di analizzare le righe dell'interrogazione esterna
- Il linguaggio SQL dà la possibilità di inserire nelle query annidate riferimenti alla query più esterna (es. il valore di un attributo specifico)
- In tal caso (query correlate) dato che il risultato della query interna dipende dai valori della tupla esterna considerata questa viene eseguita nuovamente per ogni tupla della query esterna considerata

Subquery complesse

Esempio:

Estrarre le persone che hanno degli omonimi nella tabella **Persona(Nome, Cognome, CodiceFiscale)**

```
SELECT * FROM Persona AS P  
WHERE EXISTS ( SELECT * FROM Persona AS P1  
                WHERE P1.Nome = P.Nome AND  
                P1.Cognome = P.Cognome AND  
                P1.CodiceFiscale <> P.CodiceFiscale)
```

Subquery in FROM

- Le subquery possono essere usate anche come argomento della clausola FROM
- In questo caso il risultato della subquery viene usato come tabella dalla quale vengono presi i dati per la query esterna
- E' obbligatorio in questo caso l'uso dell'alias per dare un nome alla tabella risultante subquery

Esempio:

Trovare il Nome degli impiegati che hanno un premio di produzione piu' alto di 3000 e il cognome 'Rossi'

SELECT Nome

FROM (SELECT * FROM impiegati WHERE PremioProduzione > 3000)

AS impiegatipremioalto

WHERE Cognome = 'Rossi';

Subquery in FROM

Esempio:

Trovare tutti il nome del reparto e l'indirizzo degli impiegati che hanno un premio di produzione piu' alto di 3000

SELECT DISTINCT NomeReparto, Indirizzo

FROM

(SELECT * FROM impiegati WHERE PremioProduzione > 3000) AS impiegatipremioalto

NATURAL JOIN reparti;