Linguaggio SQL

Il *modello relazionale* definisce i **concetti generali** e i **vincoli** per modellare e strutturare i dati di una certa applicazione. Tuttavia, tramite lo studio del solo modello relazionale, non sarebbe possibile costruire lo *schema* del DB e manipolare le *istanze* associate. Questa richiesta è soddisfatta tramite l'impiego di **linguaggi data-oriented**, i quali possono essere suddivisi in tre macro-categorie rispetto alla compatibilità o meno con un RDBMS, ossia:

- *Linguaggi rappresentativi tramite interfacce grafiche*
- *Linguaggi basati su proprietà algebrico/logiche*
- *Linguaggio SQL*

Tra i tre citati senza alcun dubbio, in associazione ad un modello relazionale, **SQL** rappresenta il *linguaggio di riferimento*.

Introduzione

SQL è un linguaggio per la costruzione di modelli relazionali. Caratterizzato dagli stessi concetti generali su cui si fonda un qualsiasi *modello relazionale*, ma pone alcune differenze, in riferimento:

- La denominazione data alle *relazioni* in questo contesto prende il nome di **tabelle**
- Il risultato di un'operazione sui dati può restituire una tabella con **righe duplicate**
- Il sistema dei **vincoli** è più **espressivo** o **informativo**
- Il vincolo di **integrità referenziale**, ossia in relazione alla *foreign key*, è **meno stringente**. Si ricordi la regola generale, la quale richiede che tra tabella referenziante e referenziata, le colonne che compongano la referenza siano corrispettivamente qualsiasi dominio per la prima relazione, ma che ricada esattamente sulla chiave primaria della seconda relazione; ciò rispetto ad una manipolazione di dati potrebbe aumentare di molto la complessità gestionale.

La grande diffusione di SQL è dovuta da un duro lavoro di standardizzazione che ha portato ad innumerevoli versioni e modifiche del paradigma, impegno portato avanti sin dagli anni ottanta. Tuttavia ancora oggi, differenti pubblicazioni sono ancora lontane dall'essere comunemente adottate. Per questa ragione, spesso si fa riferimento a *SQL-2*, il quale pone sottili caratteristiche legate alla sintattica del linguaggio.

SQL non è solamente adottato per esprimere interrogazioni rispetto alla base di dati di riferimento. Contiene infatti al suo interno funzionalità dedicate al **DDL**, *Data Definition Language*, il quale permette la definizione dello *schema* di una base di dati, e al **DML**, *Data Manipulation Language*, ossia un insieme di comandi per la modifica e l'interrogazione delle istanze di una base di dati.

Data Definition Language

La prima parte trattata riguarda le funzionalità adoperate per la definizione dello *schema* della base di dati. Tramite il costrutto *create database*, è possibile costruire uno **schema** di

una base di dati, da non confondere con lo *schema* di una tabella, ossia dall'insieme posto dal nome della relazione e degli attributi associati. I comandi dedicati sono:

create database [if not exists] nameDataBase

implementato per la creazione del collettore successivo di *tabelle*, *viste* e *interrogazioni* alle istanze di riferimento, e di seguito è riportato il corrispondente comando per l'eliminazione del database:

drop database [if exists] nameDataBase

si osserva come in entrambe le funzioni siano descritti comportamenti *opzionali*, dovuti dall'uso della parentesi quadrate *[if not exists]* e *[if exists]*, le quali permettono di verificare, in questo contesto, dell'esistenza o meno del DB indicato nella nomenclatura.

Tramite il costrutto *create table*, è possibile costruire una **tabella** all'interno dello *schema* del DB. Il comando dedicato è:

create table nameTable (

nameAttribute1 Domain [ValDefault][Constraint]
nameAttribute2 Domain [ValDefault][Constraint]
...

);

nuovamente, tutto ciò che è circondato da parentesi quadrate indica un comportamento opzionale, ad eccezione del nome e del dominio dell'attributo specifico. In questo esempio la denominazione *[ValDefault]* stabilisce quale sia la variabile di default associata all'istanza immessa qualora non sia specificato il valore, mentre *[Constraint]* stabilisce un vincolo al quale un qualsiasi record immesso dovrà sottostare, di seguito verranno riportati i *vincoli* principali.

Domini elementari

In SQL possono essere associati differenti **domini elementari** agli attributi di uno schema; *schema* inteso a livello di tabella, per cui l'insieme dettato dal nominativo della relazione e dal nome degli attributi. *Character*
Il dominio *character* permette di rappresentare singoli caratteri oppure stringhe. La lunghezza di una stringa può essere sia fissa che variabile, nonostante in entrambi casi occorra specificare la lunghezza massima. La sintassi è:

character [varying] [(Length)] [character set Language]

quindi qualora si voglia inserire una stringa di 20 caratteri occorre specificare *character (20)*, oppure in relazione ad una stringa con un massimo di 1000 caratteri, adottando uno specifico alfabeto occorre indicare *character varying (1000) character set Greek*. Se la lunghezza non è specificata, il dominio dell'attributo indica un unico carattere. SQL ammette anche delle varianti compatte, spesso più utilizzate rispetto alle prime citate, le quali rispettivamente

sono *char* e *varchar*, per stabilire una dimensione fissa piuttosto che una dimensione variabile.

Tipi numerici esatti
Questa famiglia consente di rappresentare valori esatti, interi o con una parte decimale di lunghezza prefissata. SQL mette a disposizione differenti soluzioni, quali:

numeric [(Precision [, Scale])]

decimal [(Precision [, Scale])]

integer

smallint

I domini *numeric* e *decimal* rappresentano numeri in base *decimale*. Il parametro *Precision* indica l'accurattezza del numero, mentre *Scale* indica quante cifre sono dedicate alla parte frazionaria.

numeric (4, 2) : Intervallo[-99.99 : 99.99]

Nei casi in cui non sia zona di interesse la rappresentazione della parte frazionaria, allora diventa possibile usare i domini predefiniti *int* e *smallint*. Rispetto a questi ultimi due formati è possibile combinare il campo *auto−increment*, mediante (*integer auto−increment*), consentendo di creare dei campi numerici che si auto-incrementano ad ogni nuovo inserimento nella tabella.

Tipi numerici approssimati
I tipi **numerici approssimati** consentono di rappresentare valori reali con rappresentazione in virgola mobile. SQL fornisce i seguenti tipi:

float [(Precision)]

real

doule precision

I domini in questione permettono la rappresentazione di numeri approssimati mediante l'utilizzo di una virgola mobile, suddividendo il numero in una coppia di valori: la mantissa e l'esponente (*0.17E16 −> 1,7∗10*^{15}*, dove 1,7 indica la mantissa mentre 15 l'esponente*).

Domini Temporali
I **domini temporali** consentono di rappresentare informazioni temporali, intervalli di tempo o istanti di tempo. SQL-2 mette a disposizione tre diverse forme:

date

time [(Precision)] [with time zone]

timestamp [(Precision)] [with time zone]

Ciascuno di questi domini è strutturato e suddivisibile in un insieme di campi. Come ad esempio *date* ammette i campi *year*, *month* e *day*, il dominio *time* ammette *hour*, *minute* e *second*, infine *timestamp* ammette tutti i campi, da *year* a *second*. Sia per *time* che per *timestamp* è possibile specificare una precisione, che rappresenta il numero di cifre decimali

che si devono utilizzare nella rappresentazione delle frazioni di secondo.

Domini elementari
I **domini elementari blob** e **cblob** consentono di rappresentare oggetti di grandi dimensioni come sequenza di valori binari o di caratteri. La *dimensione massima* del file dipende dalla specifiche implementazioni del linguaggio SQL. Tuttavia tramite tale formato non è possibile specificare interrogazioni sui record accomunati da tale attributo.

Tramite il costrutto **domain** è possibile costruire il proprio **dominio di dati** a partire dai domini elementari. La sintassi risulta essere cosi descritta:

create domain NameDomain as TypeDate

[Default value]
[Constraint]

create domain NameDomain as TypeDate Default (27)

inoltre rispetto a quanto detto, è possibile specificare un **valore di default** attraverso il costrutto *default*, ossia:

default [value | user | null]

Vincoli intra-relazionali

Sia nella definizione dei domini che delle tabelle, possono essere definiti alcuni vincoli, ovvero delle proprietà che devono essere rispettate da ogni record della base di dati. Il costrutto più potente per specificare vincoli generici, è il costrutto *check*, che richiede di formulare delle interrogazioni alla base di dati.

nameAttribute smallint check((attribute >= number1) and (attribute <= number2))

vincolo applicato per ogni *ennupla*. I più semplici vincoli adottabili sono di tipo *intra-relazionale*, quali *not null*, *unique* e *primary key*.

Not Null
Il valore *NULL* è un particolare valore adottato in assenza di informazioni. Tuttavia il vincolo stabilito non ammette il valore *nullo* come valore dell'attributo, quindi il dato dovrà essere sempre specificato, indipendentemente dall'istanza che si voglia inserire. Invece se l'attributo in questione è combinato rispetto ad un valore di *default* diverso dal valore nullo, allora diventa possibile inserire l'istanza anche in assenza di specifica a livello di inserimento. Il vincolo è specificato attribuendo alla definizione dell'attributo la dichiarazione *not null*, come segue:

nameAttribute varchar(20) not null

Unique
Il vincolo *unique* impone che l'attributo o l'insieme di attributi su cui si applica non possa/possano avere duplicati, ossia stessi valori in istanze differenti, pertanto gli attributi in questione segueno regole definite dal concetto di *superchiave*, per cui essi stessi sono delle superchiavi. Il vincolo può essere espresso in due sintassi, dove la prima delle due descrive:

nameAttribute Domain [ValDefault] unique

qualora la superchiave della relazione sia un unico attributo, oppure quando è composta da più attributi si adotta solitamente:

unique(nameAttribute1, nameAttribute2, ...)

concludendo, quest'ultima sintassi è riportata al termine del costrutto per la creazione della tabella. Tuttavia, le due sintassi non hanno la stessa valenza, poichè specificare il vincolo per ogni attributo piuttosto che raggruppare il tutto come la seconda sintassi non persegue nello stesso obiettivo. Infatti, rispettivamente, il primo citato indica che la specifica colonna non possa avere duplicati al suo interno, mentre la definizione *unique(nameAttribute1, nameAttribute2, ...)* indica che l'insieme dei domini riportati per tutte le istanze seguenti non possano avere valori combinati duplicati.

Primary key
Il vincolo *primary key* può essere ugualmente posto come il vincolo *unique*, ma con alcune differenze a livello di implementazione. Tale proprietà ammette che gli attributi associati non possano mai assumere il valore nullo, inoltre l'uso di questo vincolo può avvennire una sola volta, al contrario di unique e not null. In relazione a quanto detto, la sintassi del costrutto è:

create table nameTable(

nameAttribute1 Domain [ValDefault]
nameAttribute2 Domain [ValDefault]
primary key (nameAttribute1, nameAttribute2)

);

Vincoli inter-relazionali

I vincoli inter-relazionali più diffusi sono i *vincoli di integrità referenziale*. In SQL spesso sono denominati come *vincoli* di *foreign key*, ossia di *chiave esterna*. Questa proprietà crea un collegamento tra una tabella *referenziante* e una tabella *referenziata*. Il vincolo impone che ogni riga che compare nella tabella referenziata di un attributo specificato, compaia anche nelle righe della tabella referenziante in un totale di attributi di simile dominio. L'unico requisito imposto richiede che l'attributo o l'insieme di attributi di riferimento, posti internamente alla tabella referenziata, siano soggetti al vincolo *unique*, ovvero sia un identificatore della relazione. Il vincolo può essere posto sia come vincolo *unique* oppure come vincolo *primary key*. Per cui seguono due costrutti di differente sintassi:

create table nameTable1(

nameAttribute1 Domain [ValDefault] references nameTable2(nameAttribute1)
nameAttribute2 Domain [ValDefault]
primary key (nameAttribute1, nameAttribute2)

);

dove si osserva la nuova semantica *references* utilizzabile solo quando un singolo attributo risulta essere coinvolto per la costruzione della referenza, mentre qualora sia composta da più di un singolo attributo si adotta:

create table nameTable1(

nameAttribute1 Domain [ValDefault] primary key
nameAttribute2 Domain [ValDefault]
foreign key (nameAttribute1) references nameTable2(nameAttribute1)

);

uso del costrutto *foreign key* per la costruzione è posto sempre al termine della definizione degli attributi, inoltre qualora siano specificati tra le parentesi un numero superiore di attributi per la referenza, SQL segue lo stesso ordine imposto sia per la tabella referenziante e sia per la tabella referenziata.

Tuttavia la modifica o l'eliminazione di un valore della tabella referenziante, potrebbe causare l'invalidità del *vincolo di integrità referenziale* per la relazione referenziata. In queste casistiche, spesso si adottano degli strumenti associativi alla tabella referenziata per monitorare violazioni della *foreign key*, gestiti mediante lo stesso costrutto *on*, il quale è definito come:

on (delete | update) (cascade | set null | set default | no action)

per cui, tramite la sintassi introdotta, si evince la possibilità di poter manipolare violazioni del vincolo a causa di variazioni sui domini che compongono la *foreign key*. Generalmente, qualora si dovessero verificare azioni di modifica e cancellazione, saranno imposte anche alle colonne che compongano la referenza tra le tabelle in questione.
Successivamente alla sintassi *(delete | update)* sono collocati ulteriori strutture intercambiabili, le quali illustrano il comportamento consecutivo all'azione imposta che si ingiunge sui domini interessati.

Data Manipulation Language

La seconda parte trattata riguarda funzionalità adoperate per modificare e interrogare la base di dati. In una qualsiasi applicazione data, si cerca di emulare le quattro funzionalità CRUD, ossia *create*, *read*, *update* e *delete*. Per rispondere alla necessità dell'*operazioni di lettura*, o meglio definite di *interrogazione*, è utilizzato un singolare costrutto, così attuato:

Select nameAttribute1, ..., nameAttributeN

From nameTable1, ..., nameTableM

Where Condition

a livello di operatività, si effettua il *prodotto cartesiano* delle tabelle prese in considerazione, da cui verranno estratte le *righe* che rispettino la *condizione* posta nell'ultimo blocco sintattico. Di quest'ultime saranno prelevate solo le *colonne* corrispondenti alla selezione posta in primo piano.

Prima di poter distinguere le differenti azioni conseguibili attraverso le *query*, occorre definire

al meglio quale sia l'approccio di ogni *blocco sintattico*, con l'obiettivo di illustrare operatori che possano approcciarsi al problema posto dinanzi.

Select
La clausola **Select** specifica quali *colonne* debbano essere prese in considerazione per poter produrre il risultato voluto. Per cui, differentemente rispetto a *From* e *Where*, opera solamente a livello di *domini*, senza considerare le *righe* della tabella. Spesso, tale sintassi, è accomunata da un discreto livello di *ambiguità*; ossia, possono essere prese in considerazioni tabelle che costituiscano il risultato voluto, le cui colonne hanno nominativi simili. Per ovviare a tale problematica, è attuato un certo costrutto definito, *As*. Questo strumento permette di ridenominare i *domini* selezionati dalla *query*, come segue:

Select nameAttribute As Name

From
La clausola **From** illustra l'insieme di tabelle che dovranno essere interrogate, pur di ottenere il risultato voluto. La molteplicità di *relazioni* al suo interno permette di descrivere query sempre più avanzate e articolate, adeguandosi rispetto al *prodotto cartesiano*. Come avviene per il costrutto *Select*, è possibile utilizzare *As* affinchè si possano ridenominare le tabelle appartenenti all'insieme di interesse. Ciò avviene per imporre una distinzione referenziale rispetto ai *domini* adoperati.

Where
La clausola **Where** indica quali *righe* delle tabelle considerate, debbano comparire nel risultato richiesto. All'interno della *condition* sono poste molteplici espressioni, le quali coinvolgono differenti settori, quali:

- Operatori booelani, come *And*, *Or* e *Not*, dove il loro utilizzo è simile a quanto adottato in linguaggi di programmazione
- Operatori di confronto, i quali avvengono tramite il costrutto *Like*, implementato principalmente per comparare variabili di tipo stringa
- Operatori di insiemistica, un esempio è dato dallo strumento *Between* che permette, come da denominazione, di verificare se un dato, di tipo numerico, appartenga ad un certo intervallo

L'insieme dei termini sintattici da poco descritti, permette la costruzione di *espressisoni regolari*. Un'*espressione regolare* è definita come una sequenza di simboli che identifica univocamente un *insieme di stringhe*.

Una nota di interesse ricade nella considerazione di *valori speciali*, come *NULL*, i quali possono comparire nel blocco sintattico *Where*. Valorizzando una query simile a quanto visto fino ad ora, immetendo il valore *NULL* come se fosse un *dominio elementare*, non permetterebbe la loro inclusione nel risultato finale. Per cui sono implementati operatori designati per questo specifico dato. Si basano sulla *logica a tre fattori*, idealizzata per comprendere *connettori booleani*, distinti in:

Where nameAttribute NOT NULL

Where nameAttribute IS NOT NULL

Join
Come già accennato, il *costrutto* generale adottato per *interrogare* la base di dati può comprendere un numero crescente di *relazioni*. In certi contesti, pur di poter risalire ad un dato ricercato, occorre combinare valori appartenenti a tabelle differenti. Si riscontra solitamente una sintassi di tale ordine:

Select nameAttribute As name

From nameTable1, nameTable2

Where (primaryKeyTable1 = foreignKeyTable2) And Condition

E' bene soffermarsi sul processo esecutivo di una query simile, poichè sono presenti passaggi atomici che contraddistinguono la totalità di codice scritto in SQL, i quali possono essere riassunti come di seguito:

- Si effettua il *prodotto cartesiano* delle tabelle, ossia l'insieme delle tuple ordinate appartenenti a tutte le relazioni in questione, generandone una comprendente tutti i domini
- Si seleziona l'insieme di righe che abbiano valori comuni e che rispettino la condizione posta nel costrutto *Where*
- Si seleziona il dominio specificato inizialmente nel primo costrutto sintattico
- Concludendo, stabiliti i passaggi, si articolano fra loro pur di ottenere il risultato finale

Notazioni aggiuntive
Una query in genere, se non imposto, non attua filtri a livello di ricerca. Per cui la costruzione di un'interrogazione potrebbe produrre come risultato finale una moltitudine di *rows duplicate*, le quali non favoriscono quanto richiesto. Una soluzione prevede l'uso di costrutti specifici, articolati nella prima parte sintattica del codice SQL. I quali, brevemente, sono posti come segue:

Select distinct(nameAttribute1), ..., nameAttributeN

ciò consente di rimuovere tutti i duplicati, che abbiano medesima valorizzazione rispetto ai domini presi in considerazione, all'interno del risultato.

Dato un approccio che mitiga su un certo rigore, un'altra espressione che opera sullo stesso layer è data da:

Where condition

Order by nameAttribute1, ..., nameAttributeN

Order by permette di imporre un criterio d'ordine rispetto agli attributi che formulano i domini selezionati; si evince che la massima utilità del costrutto avviene qualora le colonne interessate siano *domini elementari* di carattere *numerico*.

Esempi particolari
...

Operatori

Operatori aggregati
Gli **operatori aggregati** sono costrutti applicati ad insiemi di *tuple* e producono un unico risultato. SQL impone che siano inseriti all'interno della sintassi della *Select*, per cui valutati dopo aver applicato le clausola del *Where* e del *From*.

Sono adoperati differenti tipologie di *esecutori*, i quali si suddividono in:
Trovare qualcosa sul libro ...

- *Count*, permette di contare il numero di righe della relazione ottenuta, mediante non solo alla selezione dei domini, ma anche in relazione alla condizione verificata e alle tabelle che compongano il prodotto cartesiano
- *Max*, restituisce il massimo della colonna selezionata
- *Sum*, restituisce la somma della colonna selezionata
- *Min*, restituisce il minimo della colonna selezionata
- *Avg*, restituisce la media della colonna selezionata

Tra gli esecutori elencati vige una diversificazione rispetto al loro utilizzo all'interno della clausola primaria. Tuttavia, la sintassi generale per poter usufruire delle caratteristiche descritte risulta:

Select Operation(nameAttribute)
From nameTable
Where Condition

Posta la sintassi generale, la diversificazione si nota dalla costruzione dell'interrogazione, dove si illustrano due tematiche principali:

Select Count(| nameAttribute | nameAttribute1, ..., nameAttributeN)*
From nameTable
Where Condition

Select Avg(nameAttribute)
From nameTable
Where Condition

da cui si nota che la caratteristica fondamentale che non eguaglia le due entità avviene all'interno della clausola *Select*. Infatti l'operatore *Count* garantisce un'esecuzione su insieme di possibili domini, si noti la presenza del carattere speciale (*); mentre esecutori come *Avg* sono adottati nei confronti di una singola colonna. E' bene sottolineare come questa sottile discrepanza non incida sul processo esecutivo delle query introdotto e illustrato fino ad ora.

Anche in questo caso è bene soffermarsi sul processo esecutivo che caratterizza query simili:

- Si considerano tutte le tabelle selezionate dalla clausola *From*, prestando attenzione a rimuovere ogni ambiguità

- Si effettua il *prodotto cartesiano* delle relazioni, generandone una comprendente tutti i domini del caso
- Si seleziona l'insieme di righe che rispettino la condizione del costrutto *Where*
- Si considerano i domini posti all'interno della *Select*, a cui si applica l'*operatore aggregato* sulla colonna richiesta
- Concludendo, stabiliti i passaggi, si articolano fra loro ottenendo una relazione comprendente una sola colonna e riga

Group by
Un *operatore aggregato* restituisce un solo valore, come indicato dal processo esecutivo. Tuttavia, in alcuni contesti, potrebbe essere richiesto di formulare espressioni che riescano a combinare *esecutori* rispetto ad un numero crescente di *record*. Ciò è garantito da un costrutto specializzato, il quale accorda l'univocità degli operatori aggregati rispetto alla selezione multipla della *Select*.

Per aggregare i due approcci, si adottano **operatori di raggruppamento**, che consentono di dividere la tabella in *gruppi*, ognuna caratterizzata da un valore comune dell'attributo specificato nell'operatore. La sintassi si dimostra come segue:

Select nameAttribute1, ..., nameAttributeN
From nameTable1, ..., nameTableM
Where Condition
Group by nameAttribute1, ..., nameAttributeM

una nota di interesse avviene nei confronti delle liste attuate nei construtti *Select* e *Group by*, in cui i domini della prima sintassi devono essere un sottoinsieme dei domini dell'*operatore di aggregazione*, poichè un *gruppo* produce una singola riga nel risultato finale. **Trovare spiegazione più esaustiva**.

Having
La sintassi **Having** permette di applicare un *filtro di ricerca* rispetto all'aggregazione precedente del costrutto *Group by*. L'*operatore di ricerca* è posto al termine delle query, da cui ne deriva:

Select nameAttribute1, ..., nameAttributeN
From nameTable1, ..., nameTableM
Where Condition
Group by nameAttribute1, ..., nameAttributeM
Having Condition

dove ciò che differenzia l'ultima condizione posta, rispetto alla precedente, consiste sull'entità su cui è applicata. *Where* valuta per ogni *row* che sia parte della selezione dei domini, mentre *Having* è valutata su ogni gruppo, per cui non restituisce singoli elementi ma un insieme oppure un sottoinsieme di elementi.

Nuovamente, il processo esecutivo subisce delle modifiche rispetto ai nuovi costrutti introdotti:

- Si effettua il *prodotto cartesiano* delle tabelle, generandone una comprendente tutti i domini del caso
- Si estraggono l'insieme di righe che rispettino la condizione del costrutto *Where*
- Avviene il partizionamento della tabella, rispetto alla funzionalità imposte
- Si selezionano i gruppi, i quali sono rappresentativi di sottoinsiemi di interesse
- Si estraggono i domini della selezione, a cui saranno successivi gli *operatori aggregati* sui gruppi modellati, componendo il risultato richiesto

Operatori insiemistici
SQL mette a disposizione, oltre ad *operatori aggregati* e *operatori di raggruppamento*, esecutori affini alla logica insiemistica. Gli **operatori insiemistici** sono adoperati affinchè attività basilari appartenenti alla teoria degli insiemi, possano essere utilizzate anche tra colonne di differenti tabelle. Le operazioni effettuabili, immesse all'interno della clausola *Select*, sono:

- *Union*, come da denominazione rappresenta l'unione di domini, rispetto alla totalità di tabelle prese in considerazione dalla clausola *From*
- *Intersect*, rappresentativa dell'*intersezione*, per cui il risultato consiste in tutte quelle colonne che abbiano valori comuni
- *Except*, individua l'*eccezione*, ossia la sottrazione tra insiemi

Esempi particolari
...

Join

Fino ad ora l'operatore *join*, il quale permette di ricavare una tabella comprendente un insieme di domini appartenenti a relazioni differenti fra loro, è stato utilizzato all'interno della clausola *Where* affinchè sia rispettata la condizione del *vincolo inter-relazionale*. Tuttavia spesso si fa uso di un'ulteriore metodologia, del tutto equivalente rispetto a quella descritta precedentemente. Si tratta dell'esecutore **Inner Join**, manifestato all'interno della clausola *From*. La sintassi esplicativa è definita come segue:

Select nameAttribute
From nameTable1 Join nameTable2 On primaryKeyTable1 = foreignKeyTable2
Where Condition

Si osserva quale sia la differenza principale che contraddistingue le due sintassi, ossia l'unione delle relazioni interessate è anticipata rispetto alla corrisposta. Nell'esempio sono riportate le colonne che costituiscano il *vincolo inter-relazionale*, le quali sono indicativamente di denominazione differente, tuttavia per ovviare ad ambiguità è obbligatorio formulare il costrutto *nameTable.nameAttribute*.

Concludendo esistono tre varianti dell'operatore, anche se poco utilizzate, le quali approcciano ad unioni di tabelle in maniera singolare. Brevemente esse sono suddivise in:

- *Left Join*, pone il risultato dell'*Inner Join* in combinazione con le righe della *tabella di sinistra*, che non hanno un corrispettivo a destra

- *Right Join*, pone il risultato dell'*Inner Join* in combinazione con le righe della *tabella di destra*, che non hanno un corrispettivo a sinistra
- *Full Join*, pone il risultato dell'*Inner Join* in combinazione con le righe della *tabella di sinistra/destra*, che non hanno un corrispettivo

Esempi particolari
...

Query annidate

Le query illustrate rappresentano un livello di complessità piuttosto basilare, in grado di rispondere ad un breve insieme di interrogazioni alla base di dati. Tramite la clausola del *Where* possono essere articolate *espressioni complesse*, in cui valori appartenenti a domini sono confrontati con il risultato derivato da un'ulteriore **query annidata**.
La sintassi adeguata risulta:

Select nameAttribute
From nameTable
Where Condition (Select ... From ... Where ...)

Il costrutto indica due *query annidate*, in cui insiemi o singoli valori possono essere confrontati con una *query interna*, la quale potenzialmente potrebbe consistere in una relazione.

Una *query annidata* si suddivide in due entità, dove l'interrogazione compresa nella clausola *Where* è definita **query interna** mentre quella posta esternamente è chiamata **query esterna**. Come citato poco fa, la *query interna* è in grado di restituire singoli elementi oppure insiemi di elementi. In caso si dovesse verificare la seconda effettività, non sarebbe possibile utilizzare *operatori di confronto* adoperati sino a ora. Per ovviare alla problematica, SQL mette a disposizione **operatori speciali di confronto**, i quali coinvolgono medesimi meccanismi di funzionamento di operatori basilari, distinti in:

- *All*, restituisce il risultato voluto se e solo se il confronto tra il valore del dominio e *tutti* i valori della query interna soddisfino la condizione
- *Any*, restituisce il risultato voluto se e solo se il confronto tra il valore del dominio e *almeno* uno dei valori della query interna soddisfino la condizione
- *Exists*, restituisce il risultato voluto se e solo se il confronto della query annidata fornisce un *insieme non vuoto*, comprendente almeno un valore
- *In*, restituisce il risultato voluto se e solo se un *certo valore* è contenuto nel risultato della query interna

Terminando, in relazione a *query annidate*, solitamente si suddividono due aree, le quali sono caratterizzate da riferimenti interni ed esterni tra le query che compongono l'interrogazione.

Definizione informale
Una query annidata è detta **semplice** se non avviene passaggio di binding, dove la query più interna viene valutata con ciascuna riga della tabella esterna, ossia non è presente alcuna

dipendenza fra le due.

Definizione informale
Una query annidata è detta **complessa** se avviene passaggio di binding, dove le interrogazioni interne vengono valutate su ogni tupla. **Trovare migliore definizione**

Viste

Le **viste** sono tabelle *virtuali*, formulate mediante dati contenuti in ulteriori relazioni della base di dati. Ogni *vista* ha una propria denominazione e un insieme di domini che la contraddistingue, ottenuta mediante un costrutto sintattico particolare, quale:

Create view nameView (nameAttribute1, ..., nameAttributeN)
As SelectSQL
...

I valori che compongono una *vista* non sono memorizzati fisicamente, poichè dipendono da altre relazioni, per cui la costruzione avviene solamente in memoria volatile. Spesso il costrutto introdotto è utilizzato qualora una richiesta specifica non possa essere formulata mediante *query annidate*, la quale richieda l'utilizzo di *operatori aggregati* e costrutti di selezione articolati.

Sono attuate differenti realtà in cui si adoperano le *tabelle virtuali*, suddivise in:

- Implementazione di meccanismi di indipendenza tra il livello logico e il livello esterno
- Garantire la retrocompatibilità con precedenti versioni del database
- Realizzare interrogazioni complesse, semplificandone la sintassi

Esempi particolari
...

Costrutti avanzati

Fino ad ora sono stati trattati costrutti basilari, caratteristici della versione del linguaggio SQL2. Tuttavia, la versione successiva a quella citata precedentemente, ossia SQL3, mette a disposizione i **costrutti avanzati**, fortemente dipendenti dal *Data-Base-Management-System* adoperato.

Definizione informale
Le **procedure** sono frammenti di codice SQL, molto simili al comportamento esecutivo di funzioni in un linguaggio di programmazione. Al loro interno è possibile illustrare un nominativo, parametri di ingresso e valori di ritorno.

Definizione informale
Un **trigger** è un meccanismo di gestione della base di dati, basato sul paradigma *ECA*, acronimo di evento, condizione e azione.

Definizione informale

Un **permesso** è un meccanismo di controllo di accesso alle risorse mantenute dallo schema della base di dati. Su ogni risorsa sono stabiliti dei **privilegi**, i quali permettono di compiere le tipiche operazioni di accesso, lettura, modifica oppure di eliminazione.