**Corso di Lab. Di Basi di Dati A.A. 2009/2010**

**A cura di Francesco Cavarretta (WiLD)**

**Database:** Il termine Database indica l'insieme delle informazioni presenti in uno specifico sistema informativo.

E' composto da due diversi tipi di informazione:

- **Dati:** che rappresentano le entità dei sistemi da modellare.
  Le proprietà di tali entità sono descritte in termini di **valori**.
  I dati sono classificati in categorie in **base alla loro struttura** comune.
- **Le strutture(metadati):** che **descrivono le caratteristiche comuni** delle varie categorie di dati, quali i nomi e i tipi dei valori delle proprietà.

Un database deve rappresentare i diversi aspetti della realtà deve rappresentare anche le relazioni tra i dati.

Deve avere i seguenti requisiti:

- I dati devono essere organizzati **con ridondanza minima**.
- I dati devono essere **utilizzabili contemporaneamente da più utenti**.
  **Ciascun tipo di utenza** deve avere una specifica visione dei dati e **specifici diritti di accesso** ai dati stessi. Bisogna utilizzare tecniche che **evitino** che l'attività dei vari utenti generi **conflitti per l'uso contemporaneo dei dati stessi**.
- **I dati devono essere permanenti.**

Il **Database Management System (DBMS)**, o sistema per la gestione di basi di dati, è il componente del software di base che **consente di gestire uno o più database**.

Ha i seguenti scopi:

- **Indipendenza dei dati dall'applicazione**
- **Riservatezza nell'accesso ai dati**
- **Gestione dell'integrità fisica dei dati**
- **Gestione dell'integrità logica dei dati**
- **Sicurezza e ottimizzazione nell'uso dei dati**

**Linguaggi del DBMS:**

- **DDL(Data DescriptionLanguage),** per la definizione **dello schema logico** del database

- **DML(Data ManipulationLanguage),** per le operazioni di **interrogazione**, di **aggiornamento** dei dati, quali inserimento, modifica, cancellazione, ecc

- **DCL(Data ControlLanguage),** per le operazioni di **controllo dei dati, gestione degli utenti**, assegnazione dei diritti di accesso, **ottimizzazione** del funzionamento **del DBMS**

SQL contiene i linguaggi DDL e DML.

Tre classi di utenza del DBMS:

- **Utenti** finali
- **Programmatori**
- **Amministratori** (Database Administrator detti DBA)

**Struttura dei database**

Un database contiene uno o più tabelle e ognuna ha un nome.
Ogni tabelle contiene righe (record) e colonne.
Ogni riga contiene le informazioni relative ad una singola entità.
La colonna ha un nome che ci indica il tipo di informazione che contiene.
Ogni cella può contenere vari tipi di dati ed essere settata **NULL** cioè che non contiene alcun valore**. Due valori NULL sono sempre diversi** l'uno dall'altro.

**Primary Keys**
Una chiave primaria (primary key) è una colonna, o un gruppo di colonne (ma è conveniente avere come chiave primaria una sola colonna) i cui valori identificano univocamente una riga. Consente il collegamento frà due tabelle quindi di non avere dati ridondanti e permettere la sincronizzazione frà tabelle diverse.

**Integrità**
Le tabelle devono seguire certe regole di integrità:

- **Non ci devono essere due righe identiche**
- **Una colonna chiave primaria o parte di una chiave primari non può essere NULL**

**Domini**

**I domini sono i tipi di dato** che possono assumere le colonne.

Essi possono essere:

- Domini **elementari** (predefiniti)
- Domini **definiti dall'utente** (semplici, ma riutilizzabili)

Per i caratteri:

- **character (n) (o char(n))** definisce una stringa di n caratteri. Se manca il parametro n, il default è n=1.
- **character varying (n)(o varchar(n))**, è il formato in grado di contenere stringhe fino a lunghezza n, destinando a quel particolare valore lo spazio che effettivamente utilizza, purchè non ecceda la taglia n. Il parametro n è in questo caso obbligatorio. In Oracle esiste varchar2.

Per numerici esatti:

- **Numeric(precisione, scala)**
- **Dec(precisione, scala)**
- **Int**
- **Smallint**
-

**Precisione indica il numero massimo di cifre tra decimali e non.**
**Scala indica il numero di cifre dopo la virgola.**
Deve essere sempre scala < precisione.

Dec e Numeric in alcuni DBMS sono sinonimi ma in alcuni casi Dec può avere precisione maggiore di Numeri.
Tipi numerici approssimanti:

- **Float(precisione)**
- **Real**
- **Double**

Sono rappresentati attraverso **mantissa** (m) ed **esponente**(n).
**Precisione indica il massimo numero di cifre della mantissa.**

**In Oracle abbiamo solo il tipo Number** che può essere specificati in tre forme:

- **Number(precisione, scala) è un float**
- **Numeric(precisione) è un intero**
- **Numeric è un reale**

Oppure il tipo **Date** dove possiamo specificare un valore date come una stringa scritte nel formato 'aaaa-mm-gg', il tipo **Time** il cui format string è 'hh:mm:ss' o **Timestamp che include** tutte le informazioni sia di **Time che di Date**.

In **time** e **timestamp** esiste il concetto di **precisione che indica le frazioni di secondo**.

Esempio di Definizione di domini in Oracle

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

**Valori di default**
**Se non viene esplicitamente assegnato il valore assume NULL di default.**

**Vincoli di integrità**

Descrivono le condizioni che ciascuna istanza di una relazione deve sempre soddisfare

I vincoli di integrità consentono di **limitare i valori ammissibili per** una determinata colonna della tabella in base a specifici criteri.

- **intra-relazionali che operano su una relazione (tabella)**
- **inter-relazionali** in cui il predicato **opera su più di una relazione (tabella)**

**Vincoli intrarelazionali**
- **NOT NULL**
- **UNIQUE**
- **PRIMARY KEY (equivale a UNIQUE + NOT NULL)**
- **CHECK**

**PRIMARY KEY, UNIQUE e CHECK si possono usare su insiemi di attributi.**

**Modifica di tabelle**

**Rimozione di una tabella**

DROP TABLE <nome>

**Modifica colonne**

ALTER TABLE <nometab>

ADD COLUMN <nome> <tipo> <vincoli>

Quando si aggiunge una colonna bisogna stare **attenti al vincolo NOT NULL**. Potrebbe essere **necessario aggiungerlo successivamente oppure settare il vincolo default**.

ALTER TABLE <nometab>

DROP COLUMN <nome> {RESTRICT / CASCADE}

- **RESTRICT (è quella di default)** evita l'eliminazione
- **CASCADE** elimina le dipendenze logiche

ALTER TABLE <nometab> MODIFY

<nome> <tipo> <vincoli>

Alcune modifiche si possono fare sempre:

- Aumentare la precisione di NUMBER o il numero di caratteri in char o varchar2
- Aumentare o diminuire la scale in NUMBER

Solo se tutte le righe hanno il valore di quella colonna NULL:

- Diminuire la precisione di NUMBER o il numero di caratteri in char o varchar2
- Cambiare il tipo di dato

ALTER TABLE <nometab>

ALTER COLUMN <nome>

SET DEFAULT <valore>

ALTER TABLE <nometab>

ALTER COLUMN <nome>

DROP DEFAULT

Aggiunge i vincoli di tabella

ALTER TABLE <nometab>

ADD CONSTRAINT <nome vincolo>  <vincolo di tabella>

ALTER TABLE <nometab>

DROP CONSTRAINT <nome vincolo> {RESTRICT/CASCADE}

**FOREIGN KEY e REFERENCES**

E' un vincolo utilizzato quando la colonna fa riferimento ad una colonna con vincolo Primary Key o Unique anche qui possiamo fissare il vincolo per più colonne o per una colonna.

**Reazione a violazione di vincolo**

Vengono definite insieme alla dichiarazione del vincolo foreign key.

… REFERENCES (…) ON UPDATE / ON DELETE <reazione>.

La reazione può essere

- **NO ACTION (è quella di default)**: in caso di violazione non termina l'operazione
- **CASCADE**: cancella le righe che fanno riferimento al record cancellato o la aggiorna nel caso di update
- **SET NULL**: assegna NULL alla cella riferente
- **SET DEFAULT**: setta al valore di default la cella riferente

**Nel DML rientrano le operazioni SELECT, INSERT UPDATE e DELETE.**

In insert possiamo esplicitare i campi che vogliamo inserire.

**Le query**
La target list ci dice quali campo vogliamo visualizzare (come l'operazione di proiezione dell'algebra relazionale).

- **Select distinct** … restituisce i valori di quella target list tutti distinti fra loro
- **Select *** … seleziona tutti gli attributi
- **Select ALL** … ALL è di default

Il **where** implementa l'operatori di **selezione** come operatori da lui ammessi abbiamo (<, >, >=, <=, <>, and, or, not).

Altre condizioni della clausola where:

- **BETWEEN val0 AND val1**
- **IN**
- **IS NULL**
- **LIKE:** è usato per lo string matching **%** si usa per più caratteri _ per un solo carattere # è il carattere di escape.

Abbiamo anche i negati: **NOT BETWEEN, NOT IN, IS NOT NULL, NOT LIKE.**

Operatori aritmetici sono *, +, -, %, /.

**Alias**

Serve a rinominare una colonna nella target list.

**… <nome col> as <nuovo nome> …**

**Ordinamento**

Si può ordinare una select in base ad uno o più attributi.

**… ORDER BY <nomeattr0> {ASC/DESC}, <nomeattr1> {ASC/DESC}**

**Operatori aggregate**

- **COUNT**
- **MIN**
- **MAX**
- **AVG (media dei valori non null)**
- **SUM (agisce solo sui non null)**

**Raggruppamenti**

**… GROUP BY <attr0>, <attr1>**

**… HAVING ..** è una clausola in cui si possono utilizzare contrariamente al where gli operatori aggregate.

**Cross-join**

E’ come il prodotto cartesiano frà tabelle.

Esempio: **select A.*, B.* from A, B**

**Equi-Join o Natural Join**

E’ un cross-join in cui nel Where indichiamo quali campi devono essere uguali.

**Self-join**

Si mette in relazione una tabella con se stessa e si deve quindi utilizzare l’alias di tabella.

**Clausola using e on**

Si usa per il natural join e specifica quali attributi usare.

Per on invece dobbiamo specificare l’espressione.

**Outer join**

E’ un join in cui abbiamo delle righe che di una delle due tabelle che non hanno corrispondenza ma che vengono visualizzate nei risultati.

Le clausole rimangono identiche.

**… LEFT JOIN …**

**… FULL JOIN …**

**… RIGHT JOIN …**

**Interrogazioni nidificate**

**Una subquery è una query inclusa in un’altra** che ritorna valori utili alla soluzione della query di livello superiore.

Sono di 3 tipi:

- **Scalare**: ritorna un solo valore
- **Colonna**: ritorna una colonna
- **Tabella**: ritorna una tabella

Le subquery:

- **Non** possono contenere clausole **having e group by**
- Vanno inserite sempre dopo **l’operatore di confronto**
- **Non possiamo confrontarle con altre subquery**

Le subquery possono essere molto usate per avere su un’unica riga funzioni di gruppo e non o per eseguire confronti del tipo “con qualche” o “con tutti” i valori di una qualche colonna.

Abbiamo per questo due parole chiave:

- **ALL**

- **ANY**

Che devono sempre nidificare la select di subquery.

Fare un <> ALL è uguale a NOT IN o = ANY è come fare IN.

**EXISTS** invece è un operatore booleano che controlla se **esiste un dato in una subquery**. Esiste il **NOT EXISTS.**

Regole di visibilità delle subquery:

- Le **query interne** possono richiamare un dato da una **query più esterna** e mai il viceversa
- visibilità se abbiamo due **alias** uguali ci riferiamo a quello **piu vicino**

Una **subquery** viene eseguita **una volta per ciascuna ennupla** e quindi viene **memorizzata** in una **tabella temporanea**. Inoltre **non** possono **contenere operatori insiemistici**.

**Funzioni in SQL**

**Funzioni numeriche**

**ABS(val)**

**CEIL(val)**

**FLOOR(val)**

**LN(val)**

**LOG(val, base)**

**EXP(val)**

**POWER(val, esp)**

**SQRT(val)**

**MOD(val, divisor)**

**NVL(val0, val1)** se il val0 è nullo restituisce val1

**ROUND(val, ncifreprec)**

**TRUNC(val, ncifreprec)**

Queste funzioni numeriche lavorano su elenchi:

**Coalescence(…)** restituisce il primo valore non nullo nell'elenco

**Greatest(…)** il valore più grande se sono tutti non nulli

**Least(…)** il valore più piccolo se sono tutti non nulli

**Funzioni su stringhe**

**Concat(str1, str2)** (si può anche fare **str1 || str2**) si applica ai nomi di colonna

**ASCII(str)**

**Chr(val)**

**Length(str)**

**Lower(str)**

**Upper(str)**

**Initcap(str)** rende maiuscolo il primo carattere

**Substr(str, init[, length])** length può essere negativo, significa che i caratteri si prendono al contrario

**Instr(str, chr[, inizio, [, occorrenza]])** occorrenza ci indica quale occorrenza cerchiamo (prima, seconda, terza, etc…)

**LPAD(str, length[, padding])**

**RPAD(str, length[, padding])**

Il numero di simboli aggiunti è length - length(str). Il carattere di default è lo spazio.

**LTRIM(str, chr)**

**RTRIM(str, chr)**

**TRIM(str, chr)**

**Funzioni sulle date**
**Add_Months (n, data)** somma n mesi a data
**Current_Date** Data corrente nel fuso orario della sessione
**Current_Timestamp** Indicatore orario nel fuso orario della sessione
**Extract (UT FROM data)** Estrae una porzione di data (UT=unità di tempo) da un valore data
**Greatest(data1,data2,…)** La più recente fra le date elencate
**Least(data1, data2…)** La meno recente fra le date elencate
**Last_day(data)** La data dell'ultimo giorno del mese a cui appartiene la data
**Months_Between(data2,data1)** Fornisce data2-data1 in mesi
**Next_Day(data, 'giorno')** Data del giorno successivo a date dove giorno='Monday', 'Tuesday' etc
**Round(data, 'formato')** Senza un formato specificato arrotonda una data a 12 A.M. se l'ora della data è prima di mezzogiorno, altrimenti alle 12 A.M del giorno successivo.
**Sysdate** ritorna la data corrente

**Systimestamp** ritorna data e ora correnti

**Differenza frà date (data1-data2)** ritorna la differenza in giorni

**To_date(str)** converte una stringa in una data

**Round(data)** arrotonda alla mezzanotte della data precedente se siamo prima di mezzogiorno viceversa alla mezzanotte del giorno corrente.

**Trunc(data)** arrotonda alla mezzanotte della data precedente

**Alter session set NLS_DATE_FORMAT = "nuovo_formato"** altera il formato predefinito

**Funzioni di conversione**

**To_char(number)**

**To_char(data[, formato])**

**To_Number(str)**

Conversioni automatiche

- Un numero o una data si possono sempre convertire automaticamente in stringa
- Una stringa si può convertire in automatica in numero o data se è nel formato corretto (ciò avviene ad esempio ne between)

**Decode(var, caso0, valoreret0, …)** al massimo può contenere fino a 255 condizioni

**Case attributo**
**when valore1 then1**
**when valore2 then2**
**...**
**when valoren thenn**
**else …**
**end**

Case e Decode si possono sempre annidare.

**Operazioni booleane**

- **UNION** unione
- **INTERSECT** intersezione
- **EXCEPT** differenza

Se seguite dalla clausola **ALL include anche i duplicati**.

La notazione di questi operatori è **posizionale** cioè si tiene conto dell'ordine della target list ma non in quanto nomi di colonna ma in quanto tipi.

**Le viste**

Le Viste Logiche o Viste o View possono essere definite come delle tabelle virtuali, i cui dati sono riaggregazioni dei dati contenuti nelle tabelle "fisiche".
Forniscono una diversa visione, dinamicamente aggiornata.
Utilizzate per vari scopi:

- Semplificazione
- Protezione dati
- Scomposizione query complesse
- Riorganizzazione dati secondo nuovi schemi
- convenienti per svolgere una serie di query molto complesse

Le Viste hanno delle limitazioni:

- **INSERT** si può utilizzare solo se tutti i campi **NOT NULL** sono inclusi nella Vista
- **DELETE** non si può utilizzare su una vista di tabelle multiple
- In caso di aggiornamento di una Vista basata su una combinazione di più tabelle tutto ciò che viene aggiornato deve appartenere alla stessa tabella
- Se la Vista utilizza una clausola DISTINCT non si può aggiornare
- Sono **aggiornabili se non hanno query nidificate**

**Rimozione di una Vista**
DROP VIEW <nome> {RESTRICT/CASCADE}

- **RESTRICT** evita la rimozione della Vista se ad essa si riferiscono altra Viste
- **CASCADE** invece elimina anche le Viste che le si riferiscono

**Clausole "with {local/cascaded}"**
Cascaded è la clausola di default e la sola permessa in Oracle.

Se definisco una Vista V1 in funzione di un'altra V2 in **caso di modifica di dati su V1**

- **LOCAL** fa si che il controllo avviene solo in V1
- **CASCADED** fa si che avvenga anche su V2

**With check option**

In caso la Vista contiene un Select con un Where questa clausola fa si che in caso di **inserimento i dati verifichino la condizione imposta nel where.**

Le Viste aggiornabili sono Viste in cui sono permesse operazione di inserimento, aggiornamento ed eliminazione.

Le Viste di gruppo contengono al loro interno operatori aggregati e non sono mai aggiornabili.

**Schemi e indici**

**L'indice è la struttura primaria per l'accesso efficiente ai dati.**

- **Indice primario** è quello che viene normalmente definito di tipo **unique** sulla chiave primaria (**primary key**) di ciascuna tabella. Si osservi che chiave ed indice primario sono concetti differenti, il primo fa riferimento ad una proprietà astratta dello schema ed il secondo ad una proprietà della implementazione fisica della base di dati
- **Indici secondari** sono quelli che possono essere ti tipo **unique** e **multiple**; nel secondo caso ad ogni valore di una chiave dell'indice possono corrispondere varie tuple.

Le operazioni più delicate in una base di dati relazionale sono quelle di **selezione** e di **join** ciascuna delle due operazioni può essere eseguita in modo molto più efficiente se sui campi interessati è definito un indice, che permette un accesso diretto.
Possono essere definiti ulteriori indici su altri campi su cui vengono effettuate operazioni di selezione oppure su cui è richiesto un ordinamento in uscita (perché un indice ordina logicamente i record di un file, rendendo nullo il costo di un ordinamento).

**L'ordine in cui compaiono gli attributi nella lista è significativo**: le chiavi dell'indice vengono ordinate prima in base ai valori del primo attributo poi a partire dai valori del secondo e così via.

**CREATE [unique] INDEX NomeIndice on NomeTabella**

**DROP INDEX NomeIndice**

Gli **indici** se da un lato **accelerano le operazioni di ricerca**, dall'altro **occupano memoria** e **rallentano le operazioni di aggiornamento.**

Gli indici permettono ad un DBMS di accedere ai dati più rapidamente.

- Il **sistema crea della strutture dati** interne (gli indici) per la selezione veloce di determinate righe
- La struttura indica al DBMS dove si trova una certa riga su una tabella con colonne indicizzate, più o meno come il glossario di un libro indica la pagina in cui appare una determinata parola.

Due importanti fattori da considerare sono:

- Un indice su un attributo accellera notevolmente le query in cui quell'attributo è specificato
- Le inserzioni, cancellazioni e modifiche sulle tabelle sulle quali sono stati creati degli indici sono piu' complesse e richiedono maggior tempo
- Se su una tabella vengono **effettuate query più frequentemente di modifiche**, allora specificare degli indici su questa tabella ha senso
- Se le modifiche sono predominanti rispetto alle query, dovremo porre molta attenzione alla creazione degli indici

**Transazioni**
Una transazione è **una sequenza finita di istruzioni**.
E' legata all'attività di **aggiornamento** di un database ed è atomica.
Si può chiedere di una transazione **l'annullamento** o il **salvataggio**.
Ogni volta che facciamo una operazione su un DBMS sarà terminata nel seguente modo:
- L'utente chiede il **salvataggio (Commit)**
- L'utente chiede l'**annullamento (Rollback)**
- L'utente per qualche motivo **perdere la connessione** quindi la **transazione và annullata**

Le transazioni sono quindi unità di lavoro che devono essere svolte in un certo ordine logico e devono essere completate con successo.

L'insieme di azioni che portano il database da uno stato consistente ad un altro stato aggiornato e consistente può essere il risultato di **più utenti "simultanei"** che lo manipolano, in questo caso **sono presenti più transazioni contemporanee**.

La necessità di **delimitare ogni transazione** è rinforzata dall'esigenza di **mantenere distinto l'effetto di ogni transazione** da quello di altre transazioni **interlacciate rispetto al tempo** e ai dati condivisi su cui si opera.

Il **lock** è l'uso **temporaneo esclusivo di alcuni dati da parte** di un'applicazione o **di un utente**.
Si dice che il lock sia **granulare** nel senso che il suo **campo d'azione può essere** esteso ad un **record** o ad una **tabella**.

**Affidabilità**
- **protezione da malfunzionamenti** (hardware, e del software di base con perdita dei dati nelle memorie centrali)
- **Impedire** che programmi possano **effettuare "modifiche parziali"** su una basedati dovute, ad esempio, **a vincoli di integrità non verificati**, ad errori durante l'esecuzione dei programmi stessi

**Controllo della concorrenza**
- **Impedire l'uso di dati non completamente elaborati o** incerti dovuti all'esecuzione concorrente di più programmi su una basedati

**ACID Property**
- **Atomicity (atomicità):** unità indivisibile di esecuzione
- **Consistency (consistenza) :** non violazione dei vincoli di integrità
- **Isolation (isolamento) :** l'esecuzione sia indipendente dalla contemporanea esecuzione di altre transazioni

- **Durability (persistenza):** l'effetto di una transazione che ha eseguito il commit non venga più perso

**Atomicità**
Una transazione non può lasciare il database in uno stato intermedio:
- Un errore prima del commit causa l'UNDO del lavoro fatto
- Un errore dopo il commit può richiedere il REDO del lavoro fatto: gli effetti sullo stato del database non sono garantiti
-

**Consistenza**
La transazione non deve violare un vincolo di integrità:
- **Immediato: verifica nel corso della transazione**
- **Differito: alla fine di qualcosa** (se alcuni vincoli di integrità sono violati, l'intera transazione viene annullata)

**Integrità dei dati in caso di accesso concorrente**
- **Perdita di aggiornamenti:** una transazione T2 aggiorna un dato precedentemente aggiornato da un'altra transazione T1, l'aborto di T1 causa la perdita dell'aggiornamento fatto da T2 poiché il database viene riportato allo stato precedente a T1
- **Letture non riproducibili:** se una transazione T1 legge un dato che viene successivamente aggiornato da T2 che termina con successo e quindi T1 legge nuovamente il dato: i valori letti da T1 per lo stesso dato sono differenti poiché la seconda lettura risente dell'aggiornamento fatto da T2
- **Letture improprie o fantasma:** se una transazione T2 legge un dato scritto da T1 e successivamente T1 abortisce, il dato letto da T2 è scorretto poiché non era definitivo nel database

**Bisogna sequenzializzare correttamente le operazioni di più transazioni concorrenti.**

- **Lock dei dati: prima di utilizzare** un dato, **ciascuna transazione deve farne richiesta al sistema** mediante una operazione di lock e restare in attesa finché il sistema non lo renda disponibile.

Tipi di Lock:
- **Lock in lettura o condiviso**: compatibile con altri lock in lettura
- **Lock in scrittura o esclusivo**: non è compatibile con alcun altro lock sul dato

**Programmazione di Transazioni**
- Una transazione è programmata attraverso una serie di istruzioni SQL, in un qualche linguaggio di programmazione (e con un certo approccio)

E' quindi necessario distinguere tra:
- il programma attraverso cui si programma(no) la(e) transazione(i)
- la singola transazione
- Ciò implica anche la distinzione del concetto di terminazione che può essere del programma della transazione

**PL / SQL**
Il PL/SQL (Procedural Language/Structured Query Language) è un **linguaggio di programmazione** procedurale di **Oracle** che costituisce un'estensione dell'SQL.

Esso consente di creare **store procedures** e **funzioni**, ma anche di creare i **triggers**.
Il PL/SQL supporta le **variabili**, **condizioni** e **gestisce le eccezioni**.

I blocchi hanno questa forma generale:
**declare** Blocco di dichiarazione (opzionale)
**begin** Codice da eseguire
**exception** Gestione eccezioni (opzionale)
**end**

La sezione **DECLARE specifica** i tipi di dato delle **variabili**, delle **costanti** e i **tipi definiti dal programmatore** e definizione di **cursori**.

Il **blocco** tra **BEGIN** ed **END** specifica il **codice da eseguire**.

Le eccezioni possono essere di due tipi:
- **eccezioni predefinite**
- **eccezioni definite dal programmatore**
-

Si possono sollevare le eccezioni definite dal programmatore in modo esplicito.

**Dichiarazioni**
I **tipi di base** per le variabili che **sono gli stessi** che possono essere **utilizzati nella** definizione delle **colonne**.

Le costanti sono inizializzate in fase di dichiarazione e dichiarate con constant.

L'assegnazione dei valori alle variabili viene fatta mediante il simbolo :=.
I **valori costanti** possono anche **essere** assegnati mediante la parola chiave **default**.

**Cursore**
Un **Cursore è una variabile di tipo tupla che varia su tutte le tuple** che risultano da qualche query.

Un cursore ha **il compito di conservare i risultati di una query** affinchè possano essere elaborati da altri comandi all'interno del blocco PL/SQL.

In PL/SQL un cursore si **dichiara** mediante l'istruzione:

**CURSOR <Nome> is <query>;**

Per **prelevare** una tupla dal cursore C si usa

**FETCH C INTO <Variabile>**

**Variabile ancorata a colonne**

La **variabile viene dichiarata** nel seguente modo:

**<nomevariabile> <tabella.attributo>%TYPE**

Vantaggi:

- **non occorre conoscere il tipo della colonna**
- **se si cambia la definizione della colonna, la variabile cambia tipo con essa**

**Variabile ancorata a righe**
E' possibile definire delle variabili di tipo tupla legate alla definizione delle righe di una tabella.
Esse costituiscono di fatto dei record, in cui le definizioni dei campi sono in corrispondenza con gli attributi di una tabella.

Tali variabili sono definite nel seguente modo:

**<nomevar> <nometabella>%ROWTYPE**

Assegna a nomevar il tipo delle tuple della tabella.
nomevar potrà essere trattata come una tupla e nomevar.a fornirà il valore dell'attributo a nella tupla nomevar.

**Variabile ancorata a cursore**
In particolare **un cursore "contiene" il risultato di una select**.
Dunque, allo stesso modo in cui il tipo di una variabile si può ancorare a una riga di una tabella, esso si può ancorare anche a un cursore.

La variabile ancorata a un cursore è dichiarata così:

**<nomevariabile> <nomecursore>%ROWTYPE**

Un cursore per essere utilizzato deve essere attivato mediante l'istruzione

**OPEN <nomecursore>**

Quando non serve più viene disattivato mediante l'istruzione

**CLOSE <nomecursore>**

Con l'istruzione open, il cursore viene inizializzato automaticamente alla prima tupla del risultato della select che la definisce.

**Cicli**
Sono tre tipi:

- **Cicli semplici**: si ripetono fino al raggiungimento di un'istruzione **exit** o **exit when**
- **Cicli FOR: si ripetono un numero fissato** di volte
- **Cicli WHILE: si ripetono fintantoché una condizione è soddisfatta**

**Cicli Semplici**

```
LOOP
    <istruzioni>
    EXIT WHEN <condizione>
END LOOP
```

**Cicli semplici a cursore**
Le condizioni per uscire dal ciclo sono spesso condizioni sullo stato del cursore.

Questo stato può essere individuato mediante i valori di certi attributi:
**%FOUND** significa che il **cursore può trasmettere** un record
**%NOT FOUND** il cursore **non può trasmettere** record
**%ISOPEN il cursore è** stato **aperto**
**%ROWCOUNT numero di righe trasmesse** dal cursore fino a questo momento

**Ciclo FOR**

```
For <Variabile> IN <estremo1>..<estremo2>
LOOP
    <istruzioni>
END LOOP;
```

**Ciclo for a cursore**
In un ciclo for a cursore i **risultati di una query servono per stabilire** in modo dinamico il **numero di esecuzioni** del ciclo che è uguale al **numero di righe** del cursore.

In questi cicli **l'apertura**, la **trasmissione** e la **chiusura** dei cursori **sono eseguite automaticamente**.

```
For <variabile_cursore> in <nome_cursore>
```

**Cicli WHILE**

```
WHILE <condizione>
LOOP
    <istruzioni>
END LOOP
```

**Istruzioni condizionali**

```
IF <cond_1> THEN <istruzioni_1>
ELSIF <cond_2> THEN <istruzioni_2>
…
ELSIF <cond_n> THEN <istruzioni_n>
ELSE <istruzioni_(n+1)>
END IF;
```

**Gestione delle eccezioni**

Quando nell'esecuzione di un blocco PL/SQL si **trovano delle eccezioni** (errori), **il controllo passa** (se esiste) alla sezione in cui si gestiscono le eccezioni, che è collocata alla fine dei comandi eseguibili, **dopo la parola chiave exception**.

All'interno di questa sezione sono contenute una o più clausole del tipo

**When <nome_eccezione> then <soluzione>**

che permettono di dire quale soluzione dare nel caso in cui un'eccezione si verifica.
Se si, viene applicata la soluzione suggerita il programma termina.

Se si verifica un errore e l'eccezione non è contemplata nella sezione exception, potrebbero esserci degli effetti indesiderati, come per esempio **la cancellazione di tutti i dati** che sono stati **inseriti fino a quel punto dal programma (Rollback)** in una tabella.

La sezione che gestisce le eccezioni è definita frà exception ed end.

**Eccezioni predefinite**

- **ZERO_DIVIDE**
- **CASE_NOT_FOUND** (nessuno dei casi del CASE è verificato e non esiste l'else) etc.

I numeri e i messaggi di errore visualizzati dall'utente sono impostati mediante la procedura

**RAISE_APPLICATION_ERROR**

Che può essere chiamata all'interno di ogni segmento PL/SQL.

**Raise_Application_Error** è una procedura che prende in input due parametri.
Il numero dell'errore (che deve essere un numero compreso tra -20001 e -20999) che è il messaggio di errore da visualizzare.

**Trigger**

Un trigger definisce **un'azione** che il database **deve attivare automaticamente** quando si verifica nel database un **determinato evento**.

Possono essere utilizzati:
- per migliorare l'integrità referenziale dichiarativa
- per imporre regole complesse legate all'attività del database
- per effettuare revisioni sulle modifiche dei dati

**L'esecuzione** dei trigger è quindi **trasparente** all'utente.

I trigger **vengono eseguiti automaticamente** dal database quando **specifici** tipi di **comandi (DML o per esempio quelli come Create View)** (Eventi) di manipolazione dei dati vengono eseguiti **su specifiche tabelle**.

Anche gli aggiornamenti di specifiche colonne possono essere utilizzati come trigger di eventi.

**Privilegi**
**Per creare** un trigger bisogna avere **3 privilegi:**
- **ALTER ANY TABLE**
- **CREATE TRIGGER**
- **Essere proprietari della tabella**

.
**Trigger a livello di riga**
**I trigger a livello di riga vengono eseguiti una volta per ciascuna riga modificata** in una transazione; vengono spesso utilizzati in applicazioni di revisione dei dati e si rivelano utili per operazioni di audit dei dati e per mantenere sincronizzati i dati distribuiti.

**Per creare** un trigger a livello di riga occorre specificare la clausola **FOR EACH ROW** nell'istruzione create trigger.

**Trigger a livello di istruzione**
I trigger a livello di istruzione **vengono eseguiti una sola volta per ciascuna transazione**, indipendentemente dal numero di righe che vengono modificate.

Vengono utilizzati di solito per imporre misure aggiuntive di sicurezza sui tipi di transazione che possono essere eseguiti su una tabella.
E' il tipo di trigger predefinito nel comando create trigger (ossia non occorre specificare che è un trigger al livello di istruzione).

**BEFORE e AFTER**: **i trigger possono essere eseguiti prima o dopo l'utilizzo dei comandi** insert, update e delete; all'interno del trigger è possibile fare riferimento ai vecchi e nuovi valori coinvolti nella transazione. Occorre utilizzare la clausola.

BEFORE/AFTER <tipo di evento> (insert, delete, update).

**INSTEAD OF: per specificare che cosa fare invece di eseguire le azioni che hanno attivato il trigger.**

Ad esempio, è possibile utilizzare un trigger INSTEAD OF per reindirizzare le insert in una tabella verso una tabella differente o per aggiornare con update più tabelle che siano parte di una vista.

- BEFORE INSERT riga
- BEFORE INSERT istruzione
- AFTER INSERT riga
- AFTER INSERT istruzione
- BEFORE UPDATE riga
- BEFORE UPDATE istruzione
- AFTER UPDATE riga
- AFTER UPDATE istruzione
- BEFORE DELETE riga
- BEFORE DELETE istruzione
- AFTER DELETE riga
- AFTER DELETE istruzione
- INSTEAD OF riga

L'istruzione Create Trigger seguita dal nome assegnato al trigger

- Tipo di trigger, Before/After
- Evento che scatena il trigger Insert/Delete/Update
- [For each row], Se si vuole specificare trigger al livello di riga (altrimenti nulla per trigger al livello di istruzione)
- Specificare a quale tabella si applica
- Condizione che si deve verificare perchè il trigger sia eseguito
- Azione, definita dal codice da eseguire se si verifica la condizione

**create trigger <NomeTrigger>**
**Tipo di trigger Evento {, Evento} ON <TabellaTarget>**
**for each row**
**[when <Predicato SQL>]**
**Blocco PL/SQL**
**Tipo di trigger Evento: before o after**
**Evento: insert, update, delete**

**for each row specifica la granularità. In assenza di questa clausola si intende per ogni istruzione.**

**Vecchi e nuovi valori**
Poiché la maggior parte dei trigger ha a che fare con modifiche di righe, è importante poter fare riferimento ai valori della riga prima dell'inserimento, cancellazione, modifica e i valori dopo tali eventi.

In particolare se, per esempio, si tratta di **un trigger BEFORE UPDATE**, per valori **vecchi** intendiamo i **valori che sono nella tabella** e che **vogliamo modificare** e per nuovi quelli che vogliamo inserire al posto dei vecchi.
Se viceversa è un trigger **AFTER UPDATE**, per **vecchi** intendiamo **quelli che c'erano prima dell'update** e nuovi quelli presenti nella tabella alla fine della modifica.

In Oracle ci si può riferire automaticamente ai vecchi valori (ossia i valori prima dell'aggiornamento) e ai nuovi valori (ossia quelli dopo l'aggiornamento) rispettivamente mediante le parole chiave "**old**" e "**new**".

Mentre nella condizione (when) le parole chiave old e new compaiono senza nessun altro simbolo, nell'azione le parole chiave old e new sono precedute da due punti (:old, :new). Esse si comportano come due variabili di tipo record.

Variabili speciali, tra cui:

- **new**: in operazioni **insert o update**, rappresenta la **nuova riga** della tabella che **si vuole inserire o aggiornare**;
- **old**: in operazioni **delete o update**, rappresenta la **riga** della tabella **che si vuole cancellare o modificare**

Le variabili new e old sono di tipo record. I singoli attributi sono denotati: new.<nome_colonna> o :old.<nome_colonna>.

L'uso di queste variabili **ha senso solo** in funzioni invocate da trigger **definiti for each row**.

**Drop Trigger**
Un trigger si elimina mediante l'istruzione

**DROP Trigger <nome trigger>**

**E' possibile combinare diversi trigger su diversi comandi insert**, update e delete, purchè siano tutti allo stesso livello.

In tal caso l'evento può essere indicato per esempio come segue

**Before insert OR update**

**E i diversi casi vengono selezionati** mediante i diversi tipi di transazione, che sono **INSERTING, DELETING e UPDATING**.

**Trigger Attivi e Passivi**

- **Un trigger è attivo quando, in corrispondenza di certi eventi, modifica lo stato della base di dati.**
- **Un trigger è passivo se serve a provocare il fallimenti della transazione corrente sotto certe condizioni.**

Solo l'ultimo esempio fra quelli visti è un trigger passivo, mentre gli altri sono tutti attivi.

**Trigger Attivi**

- Per definire le cosiddette **business rules**, ovvero le **azioni da eseguire per garantire la corretta** evoluzione del sistema informativo
- Per **memorizzare eventi** sulla base di dati per ragioni di controllo (**auditing** e **logging**)
- **Per propagare su altre tabelle** gli effetti di certe operazioni su tabelle
- Per **mantenere** allineati **eventuali dati duplicati** quando si modifica uno di essi

**Trigger Passivi**

- Per definire vincoli di integrità non esprimibili nel modello dei dati usato
- Per fare controlli sulle operazioni ammissibili degli utenti basati sui valori dei parametri di comandi SQL Personalizzare le condizioni di

**Errore**

I numeri e i messaggi di errore visualizzati dall'utente sono impostati mediante la procedura **RAISE_APPLICATION_ERROR.**

Che **può essere chiamata all'interno di ogni trigger Raise_Application_Error** è una procedura che prende in input due parametri.
Il numero dell'errore (che deve essere un numero compreso tra -20001 e -20999).

**Instead OF**

- I trigger instead-of **possono essere definiti solo su viste** (relazionali od oggetto).
- A differenza di un trigger DML, che viene eseguito oltre all'operazione DML, un trigger instead-of **viene eseguito in luogo della dichiarazione DML** che lo ha attivato
- I trigger instead-of **devono essere a livello di riga**
- **INSTEAD OF fornisce un modo trasparente di modificare le viste che non possono essere modificate direttamente attraverso un'istruzione DML SQL (INSERT, UPDATE, and DELETE).**