1- Descrivi sinteticamente le funzionalità dei DBMS orientati agli oggetti.

Gli OODBMS (Object Oriented Data Base Management System) aggiungono alle caratteristiche dei classici RDBMS (Relational Data Base Management System), le carattistiche dei DBMS orientati agli oggetti. Le caratteristiche dei RDBMS sono:

- gestione della concorrenza;
- transazioni ACID (atomicità, consistenza, isolamento, durabilità (persistenza));
- integrità dei dati (tramite vincoli che permettono di mantenere i dati memorizzati in un database consistenti: puoi allungare il brodo parlando di log, politiche di commit o di modifica);
- efficienza (ottimizzatori delle query utente);
- interrogazioni;
- sicurezza (gestione dei permessi).

Le nuove caratteristiche introdotte invece dagli OODBMS sono:

- tipi di dati astratti;
- ereditarietà;
- identità degli oggetti.

2- I concetti dell'OO (Object Oriented) nei linguaggi e nelle basi di dati: **oggetto**, **stato**, **funzione**, **metodo** e **messaggio**. Il **tipo di oggetto**: interfaccia e implementazione. Incapsulamento e l'indipendenza fisica nel DBMS.

- **Oggetto**: entità software costituita da
  - **stato**: un insieme di valori di attributi chiamati anche variabili di istanza ;
  - comportamento: un insieme di metodi che restituiscono un valore al chiamante e modificano eventualmente lo stato dell'oggetto;
  - identità: un valore che identifica univocamente un oggetto detto OID (Object Identifier). L'identità è immutabile ed indipendente dallo stato e rende un oggetto unico (permette dunque di distinguere gli oggetti e risolvere i referenziamenti).

  Un oggetto è dunque un insieme di valori ed azioni che modificano lo stato di un oggetto .
- Un oggetto mette a disposizione alcuni metodi verso tutti gli oggetti, e questo costituisce l'**interfaccia** di un oggetto. E' un'esibizione di **funzione** nella programmazione OO, mentre è un tipo che specifica sia le funzioni che gli attributi di un oggetto. L'interfaccia è una concretizzazione dell'**incapsulamento**, che è basato sul concetto dell'information hiding (separazione di aspetti esterni da quelli implementativi – nelle base di dati sono visibili, oltre ai metodi, anche lo stato logico dell'oggetto. Tuttavia, l'**implementazione** dei metodi è nascosta).
- **Tipo oggetto**: definisce la struttura (Tipo variabili d'istanza) e il comportamento con le funzioni. Il corpo dei metodi agisce tramite messaggi. Il Tipo è un'insieme statico di valori.
- Un oggetto può inviare un **messaggio** ad un altro oggetto (o a sé stesso): ciò comporta l'esecuzione di un metodo. Quando un oggetto invia un messaggio ad un altro oggetto deve farlo riferendosi ad un metodo specifico che il ricevente deve eseguire. Quindi, il messaggio è il richiamo di una funzione esposta nell'interfaccia di un oggetto.
- I **metodi** sono sostanzialmente alternative alle funzioni offerte dai linguaggi funzionali e imperativi.
- **Indipendenza fisica**: Lo schema logico deve rimanere invariato se varia lo schema fisico ossia il modo di memorizzare i dati. Se non varia lo schema logico allora non variano neanche le applicazioni che accedono ai database siccome esse vi accedono tramite viste o schema logico. **Indipendenza logica**: Le applicazioni che utilizzano i database non devono variare al variare dello schema logico se questo preserva le informazioni originarie. Questo è possibile grazie alle viste che sono tabelle virtuali composte tramite interrogazioni quindi

basta cambiare le interrogazioni per fornire lo stesso schema anche se varia quello logico.

3- La nozione di **classe** nelle basi di dati OO. I quattro aspetti dell'**ereditarietà** che caratterizzano una sottoclasse. Confrontare la nozione di classe in ingegneria del software con quella delle basi di dati.

Classe: Nelle basi di dati la classe ha un aspetto prevalentemente estensionale. Un'estensione di una classe in un determinato istante è l'insieme di tutte le istanze di quella classe che non sono ancora state rimosse. Nelle basi di dati la classe è dunque un insieme di istanze sul quale si possono eseguire determinate operazioni come aggiunta di una nuova istanza, rimozione ecc.

L'ereditarietà è un meccanismo mediante il quale una classe eredita la struttura ed il comportamento di un'altra classe (sopraclasse). Un'istanza di una classe è dunque anche istanza di tutte le eventuali sopraclassi. La classe che eredita le caratteristiche di un'altra classe è detta sottoclasse. Esistono vari approcci all'ereditarietà:

- **Strutturale**: una classe eredita tutti gli attributi della sopraclasse e può aggiungerne di nuovi oppure modificare il tipo di attributi esistenti purchè i nuovi tipi siano sottotipi di quelli originali.
- **Comportamentale**: una classe eredita tutti i metodi di una sopraclasse e può aggiungerne di nuovi .
- **Overloading procedurale**: una classe eredita tutti i metodi di una sopraclasse, può aggiungerne di nuovi oppure ridefinire metodi esistenti. Questi metodi hanno la stessa signature (nome metodo + tipo di risultato restituito + parametro metodo) ma possono avere diverse implementazioni diverse.
- **Framework procedurale**: una classe può contenere richiami a metodi non implementati dalla classe stessa, all'interno dell'implementazione di metodi implementati dalla classe, e le sottoclassi possono fornire l'implementazione mancante.

è la dichiarazione in un opportuno linguaggio della struttura e del comportamento di un oggetto (attributi + metodi).

4- Classi e sottoclassi. **Overloading**. **Overriding**. **Dynamic binding**. Funzioni, metodi e relative asserzioni. Discutere come tali nozioni possono essere usate nella specifica dei **vincoli** nelle basi di dati.

- **Overloading**: metodi ereditati aventi con lo stesso nome del metodo della sopraclasse, ma semantica e implementazione differente.
- **Overriding**: metodi ridefiniti nelle sottoclassi, riscrivendone l'implementazione.
- **Dynamic Binding**: selezione della versione di un metodo a tempo di esecuzione, in base ai parametri passati.
- I vincoli, come quelli dell'algebra relazionale (chiave, integrità referenziale, not null, ecc.) favoriscono un controllo aggiuntivo per garantire la correttezza delle applicazioni e quindi l'integrità dei dati. Esiste anche un approccio basato su pre-condizioni e post-condizioni. Le pre-condizioni sono asserzioni che stabiliscono l'insieme dei valori che può assumere un input per un dato metodo mentre le post-condizioni stabiliscono l'insieme di valori che può assumere l'output. Gli invarianti sono invece o asserzioni che devono essere sempre rispettate e possono mettere in relazione anche più oggetti. Un applicazione ben progettata non deve mai violare gli invarianti, che vengono stabiliti in fase di progettazione. In ogni caso, può succedere che per cause impreviste anche se un input soddisfa le pre-condizioni si abbia una situazione anomala per cui non si può soddisfare gli invarianti oppure le post-condizioni. In questi casi è necessario ricorrere alle exception, ossia eccezioni che terminano l'esecuzione del metodo e vanno sempre gestite dal chiamante del metodo. Questa

metodologia di sviluppo, che arricchisce il paradigma OO con invarianti, pre-condizioni e post-condizioni è detta **Design by contract**.

- Funzioni e metodi possono essere usati per specificare vincoli al momento delle operazioni
- Il sistema ti tipi e sottotipi dovuti allo schema generale dell'ereditarietà costituiscono anch'esse vincoli d'integrità. Tale meccanismo fa sì che la sottoclasse sia una specializzazione di oggetti.

5- Sottoclassi e sottotipi. Definire la nozione di sottotipo richiamando gli enunciati del principio di sostituibilità. Analisi del sottotipo tupla e set.

T' è sottotipo di T se ha le stesse caratteristiche di T e ne aggiunge alcune. Si indica con T'<=T e ha le proprietà di:

- riflessiva: T è sottotipo di sé stesso;
- antisimmetrica: se T' è sottotipo di T, allora T non è sottotipo di T';
- transitiva: se T' è sottotipo di T e T è sottotipo di T1, allora T'<=T1.

- Gli approcci all'ereditarietà sono conformi al principio di sostituibilità: dato T' <= T è possibile utilizzare un'istanza di T' ovunque se richiesto T. Per mostrarlo suddividiamo una classe in due aspetti: strutturale (attributi) e comportamentale (metodi). Consideriamo ora due classi A e B dove B è sottoclasse di A. Se un attributo può avere come tipo un insieme di elementi di un tipo t, set(t), un tipo primitivo oppure una tupla rappresentiamo proprio come tupla [t1 A1 , ..., tn An] l'insieme delle variabili di istanza di un oggetto dove t1 , ..., tn sono tipi. Otteniamo che, se t1 , ..., tn sono tali che ti ≤ ti con 1 ≤ i ≤ n allora possiamo dire che :

e che

$$[t'_1\ A_1, ..., t'_n\ A_n, t_{n+1}\ A_{n+1}, ..., t_{n+k}\ A_{n+k}] \leq [t_1\ A_1, ..., t_n\ A_n]$$

$$set(t') \leq set(t)$$

6- L'aspetto comportamentale di una sottoclasse: regole che governano l'ereditarietà delle funzioni abbinate alle relative asserzioni.

siano A e B due classi con B sottoclasse di A (B<=A), e m un metodo definito in entrambe le classi relativamente con specifiche pinA , poutA , invA e pinB , poutB , invB . Per preservare il principio di sostituibilità abbiamo che se per un dato input le specifiche di input pinA sono soddisfatte anche quelle di B, pinB devono esserlo. In questo modo si può utilizzare un'istanza di B al posto di una di A potendo richiamare m con gli stessi valori:

$$pinA \rightarrow pinB$$

Analogamente per gli invarianti e i predicati di output:

$$poutA \rightarrow poutB$$
$$invA \rightarrow invB$$

7- Discutere l'ereditarietà classe-sottoclasse nelle basi di dati OO: **aspetto strutturale**, **comportamentale**, **ereditarietà multipla**, **estensione semplice** e **in profondità**, **interfacce comuni a più classi**.

- **Aspetto strutturale: class B** può essere definita sottoclasse di **class A** se **tipo(B)** è sottotipo di **tipo(A).**
- **Aspetto comportamentale:** Le funzioni di (B) conformi alle funzioni di (A) dove per funzioni conformi si intende funzioni con stessa segnatura(nome funz e tipo argomenti) ma implementazione diversa.
- **Ereditarietà multipla:** più sottoclassi ereditano da una singola superclasse. Si crea il problema del conflitto tra proprietà. Esistono più modi per risolverla.
  1. Il sistema controlla la presenza di ambiguità quando viene creato il sottotipo. In questo caso si lascia all'utente la scelta di quale funzione deve essere ereditata;
  2. C'è una scelta di default del sistema;
  3. Non si permette l'ereditarietà multipla e si obbliga l'utente a modificare il nome di una delle funzioni in uno dei supertipi (ridefinizione obbligatoria);
  4. Nome della proprietà preceduto dal nome della classe di provenienza ;
  5. ordinamento dei genitori (cioè?!).
- **Estensione di una classe:** è l'insieme delle istanze proprie della classe, cioè non appartenenti a sue sottoclassi
- **Estensione in profondità:** è l'unione dell'estensione della classe con l'estensione in profondità delle sue sottoclassi
- **Classificazione**: gerarchia tra classi instanziabili
- **Generalizzazione**: gerarchia tra classi istanziabili e interfacce (che non sono istanziabili)

8- Spazio degli oggetti: i principali modelli di riferimento CODL, FAD e ibridi. Discutere con definizioni e analisi dei pro e contro le varie proposte chiudendo le argomentazioni con esempi.

Un oggetto semi-strutturato è un oggetto che non è dotato di una particolare struttura rigida come quella dei dati dei DB relazionali (con vincoli, ecc.) , ma le informazioni sono organizzate in strutture quali insiemi, tuple, array, ecc.

Il modello Complex Object Database Language (CODL) consiste nell'organizzare gli oggetti basandosi puramente sui valori, quindi non è possibile referenziare altri oggetti tramite OID. Formalmente abbiamo:
Sia A un insieme di nomi, D un insieme di domini di base (numeri, stringhe, ecc.) unito a {null}, un oggetto può essere:

• un valore v □ D, ossia un valore atomico
• una tupla (oggetto tupla) [a1 : Q1 , ..., an : Qn] dove □i ai □ A □ Qi oggetto
• un insieme (oggetto set) {Q1 , ..., Qn } dove Q1 , ..., Qn sono oggetti

[cambia gli esempi!]
Ad esempio, supponiamo di dover memorizzare i dati di alcune persone.
Possiamo avere ad esempio il seguente spazio degli oggetti complessi
[nome : Mario, cognome : Rossi, indirizzo : [via : Roma, numero : 1], figli : {Luca, Paolo}]
[nome : Paolo, cognome : Rossi, indirizzo : [via : Roma, numero : 1], figli : null]
[nome : Maria, cognome : Verdi, indirizzo : [via : Garibaldi, numero : 3, figli : {Marco}]]

il concetto di uguaglianza è dato dalle seguenti regole :

• Due oggetti atomici sono uguali se è uguale il loro valore (es. 1 = 1, Mario = Mario, ecc.) ;
• Due oggetti tupla sono uguali se hanno nomi uguali e i relativi oggetti sono uguali;
• Due oggetti set (insiemi) A, B sono uguali se contengono gli stessi elementi.

-->sorge il problema di come identificare gli oggetti (per nome ci sarebbero sinonimi, per path sarebbe troppo esoso e per key dovremmo ricorrere ad operazioni di join per unire informazioni provenienti da più tuple)

9- Modello FAD

Il modello FAD consiste nell'organizzare gli oggetti basandosi sui loro OID., risolvendo il problema dell'identificazione degli oggetti (vedi su). Formalmente , dato un insieme di attributi A, un dominio di base D contenente l'elemento {null} ed un insieme di identificatori ID un oggetto è nella forma (OID, stato) dove lo stato è nella forma (tipo, valore) e il valore può essere:

• un valore v □ D, ossia un valore atomico se il tipo è atomico ;
• una tupla (oggetto tupla) [a1 : i1 , ..., an : in] dove □j aj □ A □ ij □ ID se il tipo è tupla
• un insieme (oggetto set) {i1 , ..., in } dove □j ij □ ID ossia un insieme di indirizzi se il tipo è set

Le tuple e i set contengono quindi indirizzi e non valori come nel modello CODL.

Riprendendo l'esempio di prima (nel modello CODL), i tre oggetti presenti nello spazio degli oggetti sono quì rappresentati come segue [cambia l'esempio!]:

(i1 , (tupla, [nome : i4 , cognome : i5 , indirizzo : i6 , f igli : i7]))
(i4 , (atomico, Mario))
(i5 , (atomico, Rossi))
(i6 , (tupla, [via : i8 , numero : i9]))
(i8 , (atomico, Roma))
(i9 , (atomico, 1))
(i7 , (set, {i10 , i11 }))
(i12 , (atomico, Luca))
(i13 , (atomico, P aolo))

(i14 , (tupla, [nome : i12 , cognome : i5 , indirizzo : i6 , figli : inull]))

(i15 , (tupla, [nome : i16 , cognome : i17 , indirizzo : i18 , f igli : i21))
(i16 , (atomico, M aria))
(i17 , (atomico, Verdi))
(i18 , (tupla, [via : i19 , numero : i20]))
(i19 , (atomico, Garibaldi))
(i20 , (atomico, 3))
(i21 , (atomico, Paolo))
(inull , (atomico, null))

Il concetto di uguaglianza è dato da una sola regola: due oggetti atomici (i1 , s1), (i2 , s2) sono uguali se è uguale il loro indentificatore, cioè se i1 = i2

MODELLO IBRIDO

Il modello ibrido unisce gli aspetti del modello CODL con quelli del modello FAD. Formalmente Sia A un insieme di nomi, D un insieme di valori atomici compreso null, ID un insieme di identificatori un oggetto è nella seguente forma (OID, stato) dove lo stato è nella seguente forma (tipo, valore) e il valore può essere :

• un valore qualunque del modello CODL (quindi una tupla contenente valori, un insieme, ecc.) se il

tipo è valore_strutturato ;
• un identificatore se il tipo è Ref ;
• un insieme di identificatori se il tipo è SetRef ;
• una tupla [a1 : s1 , ..., an : sn] dove ai □ A per ogni i e si è un stato per ogni i se il tipo è tupla .

Riprendendo il solito esempio, questo può essere ristrutturato come segue

(i1 , (tupla, [nome : (valoreStrutturato, Mario), cognome : (valoreStrutturato, Rossi)
, indirizzo : (Ref, i2), f igli : (Ref, i3)]))
(i2 , (valoreStrutturato, [via : Roma, numero : 1]))
(i3 , (valoreStrutturato, {Luca, Paolo}))

(i4 , (tupla, [nome : (valoreStrutturato, P aolo), cognome : (valoreStrutturato, Rossi),
indirizzo : (Ref, i2), figli : (valoreStrutturato, null)]))

(i5 , (tupla, [nome : (valoreStrutturato, M aria), cognome : (valoreStrutturato, Verdi),
indirizzo : (Ref, i6), f igli : (Ref, i7)]))
(i6 , (valoreStrutturato, [via : Garibaldi, numero : 3]))
(i7 , ({Marco}))

Il concetto di uguaglianza di stati è dato dalle seguenti regole :
• Due stati di tipo valoreStrutturato sono uguali in base alle regole definite per il modello CODL;
• Due stati di tipo Ref sono uguali se hanno lo stesso valore (che indica un OID);
• Due stati di tipo SetRef sono uguali in base al criterio di uguaglianza insiemistica (per gli indirizzi) dato nel modello FAD .

Spazio di oggetti consistenti(anche FAD)
- Due oggetti sono distinti se e solo se hanno due OID diversi.
- Ogni OID presente in un set (tupla) appare come identificatore in uno ed uno solo oggetto

Persistenza(anche FAD)
- Ogni oggetto raggiungibile da database è persistente
- Se o è un insieme raggiungibie da un database, lo sono anche i suoi elementi
- Se o è una tupla raggiungibile da database, lo sono anche i suoi attributi

10 – Le operazioni di interesse in uno spazio di oggetti conformi al modello ibrido con particolare attenzione ai confronti

CONFRONTO:

- **Identità: (x==y)** due oggetti in uno spazio di oggetti sono identici se hanno OID uguale;
- **Uguaglianza debole/ shallow-equal (x=y)** se i loro stati sono uguali nello spazio degli oggetti;
- Deep equals: x è uguale a y in profondità se hanno la stessa struttura, ovvero se tupla, ref e setRef sono uguali in profondità. Ovvero, case tipo:
  - **Ref**: i due oggeti sono uguali in profondità
  - **SetRef:** ogni oggetto di un insieme è uguale in profondità ad uno e un solo oggetto dell'altro insieme;
  - **tupla**: le tuple sono uguali in profondità se sono compatibili e gli attributi omonimi **o** sono abbinati a Valori Strutturati uguali **o** sono abbinati a oggetti uguali in profondità **o** sono abbinati a set di oggetti uguali in profondità (cioè ogni oggetto di un insieme è uguale in profondità ad uno e un solo oggetto dell'altro insieme) **o** sono tuple uguali in

profondità.

ASSEGNAMENTO:

siano x1 e x2 variabili e x2 inizializzata con OID
Assegnamento: x1:=x2 → x1==x2. Copia l'OID di x2 in x1.
Copia debole(shallow copy): x1:=Sx2 → x1=x2 and not x1==x. Copio lo stato di x2 in x1 ma con OID diverso.
Copia in profondità(deep copy): x1:=Dx2 → x1=inprof a x2 and not x1==x2. Crea un nuovo oggetto con uno stato equivalente in profondità ma in cui gli OID referenziati sono tutti diversi.
Merging: Dati due oggetti O1 ed O2, effettuo uno shallow equals. Fatto ciò, creo un nuovo OID, con lo stato delle variabili istanza di O1 ed O2, cancellando quindi O1 ed O2.

OPERAZIONI INSIEMISTICHE:

- Sono operazioni effettuate sulla base di confronti insiemistici (unione, intersezione, differenza, …).
- **Selezione**: operatore di selezione di tuple data una condizione: è un'espressione condizionale che indica la condizione che devono rispettare le tuple per appartenere all'insieme restituito ed il loro formato (proiezione). Restituisce quindi un sottoinsieme di S (dove S è un insieme di tuple).
- **Nest**: è l'operazione di partizione e raggruppamento per attributi. crea un insieme di tuple dove tutti i valori di ti presenti in tutte le tuple i cui valori degli attributi A − {ai } che coincidono sono raggruppati in un'unica tupla. Si tratta dunque di un'operazione simile a group by.
- **Un-nest**: è l'operazione inversa al nest. Tuttavia, ciò non è sempre vero.
- **Flatten**: operazione iterata di un nest.

11 - Persistenza per raggiungibilità e persistenza quale proprietà di una classe. Spiegare i pro e i contro alla luce delle conseguenze del modello di persistenza sull'azione di cancellazione di oggetti. La persistenza in ODMG

Estenzioni persistenti delle classi: la classe gestisce le persistenze della sua estenzione. Ogni oggetto creato è persistente attraverso costruttore o metodi.

Persistenza per raggiungibilità: si creano uno o più oggetti database che sono tuple dove ogni attributo contiene un valore set dove sono presenti tutti gli oggetti persistenti.

Cancellazione automatica: quando un oggetto non è più referenziato viene cancellato.
Cancellazione esplicita: può violare vincoli d'integrità (eccezioni o contatore)

12 – Il modello delle relazioni annidate ed estensione dell'algebra relazionale: definizione, operatori estesi e operatori di ristrutturazione

Il modello relazionale a oggetti è un'estensione del modello relazionale dove le relazioni non sono in prima forma normale. Queste relazioni prendono il nome di relazioni annidate . Quello che viola la prima forma normale è l'esistenza di attributi composti. Una relazione è dunque una regola nella seguente forma : R = (R1 , ..., Rn) dove R, R1 , ..., Rn sono nomi di attributi . I nomi che non appaiono sulla sinistra di una regola sono gli usuali attributi, ognuno abbinato ad un dominio di valori atomici. Tali nomi vengono detti nomi di ordine zero . I nomi che appaiono sulla sinistra devono essere tutti distinti nello schema e vengono detti nomi di ordine maggiore di zero. Le regole

non devono essere ricorsive.

Gli operatori algebrici di base (unione, differenza, selezione, proiezione, prodotto cartesiano) operano, in prima approssimazione, esattamente nello stesso modo dell'algebra relazionale in 1°NF, con l'unica differenza che tra le costanti ammissibili occorre includere gli insiemi di tuple. Inoltre occorre aggiungere i confronti tra insiemi e l'appartenenza di un elemento ad un insieme (membership) .

Nest: simile al group by;
Un-nest: Unnest(r2; B) = {t|(□x) (x □ r2 AND t[a1 a2 .....,an]=x[a1,a2, ...,an] AND t[b1,b2,...,bm] □ x[B])} . dato un attributo composto di un'istanza di relazione, crea dunque un insieme di tuple formate da attributi semplici al posto di quello composto.
Flatten: ripete un-nest fino ad ottenere una relazione in forma normale.
Proiezione: unione estesa delle tuple proiettate.

13 – PNF

Affinchè nest sia l'operazione inversa di unnest, occorre che le tuple si presentino in una forma particolare: la PNF.
Dato uno schema relazionale R = (a1 , ..., an , X1 , ..., Xm) con a1 , ..., an nomi di ordine zero e X1 , ..., Xm nomi di ordine maggiore di zero, l'istanza r di R è in Partition Normal Form (PNF) se e solo se :
• a1 , ..., an è una superchiave ;
• □t □ R, e □Xj si ha che t[Xj] è in PNF.

Da questo segue che Nest(UnNest(r, B), B) = r, dove B è un nome di ordine maggiore di zero di R, r è istanza di R e r è in PNF.

Siano r1 , r2 due istanze di relazioni . r1 □r2 = Ricost(Flatten(r1) □ Flatten(r2)), Dove Ricost è la ricostruzione dell'insieme ottenuto con l'unione classica del flatten delle due istanze, ossia l'annidamento consecutivo del risultato fino a riottenere un'istanza nella stessa forma di r1 e r2.
Altre estensioni sono r1 −r2 = Ricost(Flatten(r1) − Flatten(r2)) , r1x r2 = Come algebra relazionale standard. Si ricorda che gli schemi devono essere disgiunti; πY (r) = Unione estesa delle tuple proiettate: per ogni t □ πY (r), □ (t) ; σP (r) = Come algebra relazionale standard con l'aggiunta di insiemi di tuple costanti, confronti tra insiemi e operatore di appartenenza.

14 – **CLOB** E **BLOG**: a cosa servono, limitazioni, implementazione, azioni di CAST, SUBSTRING, POSITION.

Lo standard SQL99 aggiunge nuove funzionalit` ai DBMS tra cui nuovi tipi di dato non strutturati detti Large OBject (LOB). Questi oggetti sono trattati come stringhe di bit o di caratteri e il DBMS non assegna alcuna semantica a questi dati: è compito dell'utente e delle applicazioni manipolarli. Dal nome (large object) è evidente che questi tipi vengano utilizzati per memorizzare grosse quantità di dati come contenuti multimediali (immagini, video, musica, ecc.). I tipi di dati non strutturati forniti dall'SQL99 sono :

• **BLOB**: Binary Large Object - è una stringa di bit. Si dichiarano come BLOB(n) dove n è la dimensione del dato ;
• **CLOB**: Character Large Object - è una stringa di caratteri. Si dichiarano come CLOB(n) dove n è la dimensione del dato .

Per quanto riguarda la dimensione n, questa può essere espressa in byte , kilobyte, megabyte o

gigabye.

Non si possono eseguire le comuni operazioni su questi tipi di dato: si possono solo eseguire test di uguaglianza (=, diverso). SQL99 aggiunge poi alcune operazioni per manipolare i BLOB (utilizzate anche per manipolare le stringhe ed estese ai blob) tra cui :
• POSITION(str1 IN str2) che restituisce la posizione della prima occorrenza di str1 in str2 (stringhe di bit), contando che il primo carattere della stringa è in posizione 1;
• str1 — — str2 concatena le due stringhe ;
• SUBSTRING(str FROM pos [FOR n]) estrae n caratteri a partire dal carattere in posizione pos della stringa str ;
• CAST espressione AS tipo-target : CAST è la funzione che trasforma il valore con tipo specificato dall'espressione in un valore con tipo tipo-target .

15 – Tipi di dati distinti: a cosa servono. Descrivere **costruttori**, **distruttori**. **Tipi di dati opaco**.

Oltre alle funzioni e ai tipi di dato primitivi aggiunti(**tipi distinti**), SQL99 permette di inserire tipi di dato definiti dall'utente. Esistono tre approcci per definire un tipo di dato. Il primo metodo consiste nel creare tipi distinti, ossia tipi con nome diverso da quelli esistenti ma che sono equivalenti.

La sintassi è: CREATE TYPE t i p o AS t i p o −d e f i n i t o ; dove 'tipo-definito' è un tipo che è già stato definito dall'utente oppure un tipo primitivo. Ecco un esempio di creazione di tipo e conseguente definizione di tabella.

```
CREATE TYPE TESTO AS VARCHAR2( 1 0 0 0 ) ;
CREATE TABLE ALBERGO
(
NOME VARCHAR2( 3 0 ) ,
INDIRIZZO VARCHAR2( 1 0 0 0 ) ,
DESCRIZIONE TESTO,
NOTE TESTO
);
```

Notare che l'attributo INDIRIZZO e l'attributo DESCRIZIONE, pur rappresentando lo stesso tipo di dato non sono confrontabili. L'idea che conduce a questa scelta è quella di avere un sistema di tipi forte, per cui si tenta di localizzare subito gli errori di tipo e non durante l'esecuzione dell'applicazione.

In generale, dato un tipo T definito con CREATE TYPE, il **costruttore** è: nomeTipoUtente(nomeTipoBase) e l'operazione tipoBase(nomeTipoUtente) è detta invece **distruttore**. L'istruzione TESTO('stringa') richiama dunque il costruttore di TESTO fornendo un valore alfanumerico di tipo TESTO. Il confronto tra TESTO e INDIRIZZO è così possibile tramite il test ALBERGO.DESCRIZIONE = TESTO(ALBERGO.IN DIRIZZO)

Altri tipi di dato definibili sono i tipi opachi, ossia tipi non strutturati di cui si dichiara solo la dimensione occupata. La sintassi è la seguente : CREATE TYPE nome (INTERNALLENGTH = size)

15 – Tipo di dato riga con nome e uso come UDT: come si definisce in una tavola e si utilizza nelle interrogazioni e nelle azioni di update

Tipi di dati che contengono all'interno più attributi. E' un tipo di dati astratto utilizzabile come tipi

di attributi nella definizione di una tabella, dove si possono dichiarare anche tipi riga anonimi . ROW è il costruttore per i tipi di dato anonimi .

CREATE TYPE nome AS (
a t t r 1 TIPO1 ,
a t t r n TIPOn
)NOT FINAL;

```
CREATE TYPE INDIRIZZO AS
(
    VIA VARCHAR(30),
    NUMERO INTEGER
);
CREATE TABLE PERSONA
(
    NOME VARCHAR(30),
    COGNOME VARCHAR(30),
    ABITAZIONE INDIRIZZO,
    OCCUPAZIONE ROW(TIPO VARCHAR(30),
                    STIPENDIO INTEGER)
);
```

L'attributo OCCUPAZIONE è stato tipato con un tipo riga anonimo. Un tipo così costruito ha associato un costruttore implicito i cui parametri sono tutti gli attributi presenti nella dichiarazione di tipo. Ecco dunque come inserire i dati nella tabella PERSONA

```
INSERT INTO PERSONA VALUES(
        'Mario','Rossi', INDIRIZZO('Roma', 1),
        ROW('Programmatore', 1500));
```

16 – Le tavole tipate: creazione dei tipi e delle tavole. Struttura della CREATE TYPE (attributi, metodi, …) e della CREATE TABLE (Vincoli, Scope). Le varie forme dell'OID ("derived" e "system generated"). Il referenziamento (clausula REF): uso e limiti.

Per creare tipi tipi che contengono all'interno più attributi (i quali a loro volta possono essere di tipo

riga), la sintassi è la seguente :

CREATE TYPE nome AS (
a t t r 1 TIPO1 ,
a t t r n TIPOn
)NOT FINAL;

Mentre per creare una tabella come insieme di oggetti tupla, uso questa sintassi:

CREATE TABLE nome OF t i p o
(vincoli);

Posso aggiungere vincoli, chiave primaria, chiavi esterne, indici. Se aggiungo i metodi allora diventa una dichiarazione di classe. Una tabella definita in questo modo è una tavola tipata. Le tavole tipate hanno dunque lo stesso ruolo delle classi, dove le loro tuple sono oggetti, ossia istanze della tavola tipata. I vincoli vanno specificati nella forma attributo WITH OPTIONS . **REF(T)** indica il riferimento ad una tupla di tipo T tramite OID. L'associazione definita da un attributo di tipo REF è esclusivamente monodirezionale (almeno al momento). L'associazione tra tavole deve essere eseguita dal progettista usando la chiave primaria oppure REF. La cancellazione di un attributo non aggiorna automaticamente REF. Tuttavia, se usato, REF semplifica il join. Per specificare esattamente la tabella a cui un tipo REF si deve riferire occorre inserire il vincolo WITH OPTION SCOPE nome-tabella-tipo-T. Il vincolo IS SYSTEM GENERATED applicato a campi di tipo REF, indica invece che l'OID degli oggetti referenziati viene generato automaticamente dal sistema (è consigliato inserirlo sempre).

18- Tavole Tipate in gerarchia: come si definiscono, a quale principio si ispirano. Precisare l'ereditarietà dei vincoli. Interrogazioni con semantica in profondità e clausule ONLY. Illustrare i vari argomenti con semplici esempi

CREATE TYPE classe AS (...)
CREATE TYPE sottoclasse **UNDER** classe AS(...)
CREATE TABLE tavola OF classe
CREATE TABLE tavola2 OF sottoclasse UNDER tavola

Ereditarietà semplice. UNDER si usa per specificare quale classe è sottoclasse dell'altra. Si ereditano anche i vincoli. E' obbligatorio NOT FINAL nella dichiarazione del tipo classe, mentre è facoltativo nella dichiarazione del sottotipo. Supporta il binding dinamico: i parametri dei metodi possono essere istanziati con valori appartenenti a sottotipi . Le interrogazioni sono intese per estensione in profondità: ogni tupla di una sottoclasse è anche una tupla della sopraclasse. La clausula ONLY serve per ottenere le tuple solo di una classe (esclusivamente della sopraclasse).

Esempio:
sopraclasse: Donne
sottoclasse: Donne prosperose

SELECT *
FROM Donne ---> torna ANCHE le tuple delle donne prosperose.

SELECT *
FROM ONLY (Donne) ---> torna SOLO le tuple delle donne [anche di quelle non prosperose]

19 – Alcune caratteristiche dello standard SQL99: il SAVEPOINT e la problematica delle

transazioni lunghe; vari livelli di isolamento.

Esistono quattro livelli di isolabilità delle transazioni :
1. Read uncommitted(LV1): vengono ammesse le letture sporche, ossia se una transazione T1 aggiorna un dato A, la transazione T2 legge tale dato e successivamente T1 esegue un rollback, allora T2 ha letto un dato che non esiste in quanto il rollback di T1 comporta la riscrittura del vecchio dato presente in A. Tuttavia, se si è interessati ad esempio solo al conteggio delle tuple presenti in una tabella questo livello di isolabilità è adeguato, per cui lo si può scegliere avendo una maggiore efficienze nell'esecuzione delle query in parallelo con altri utenti.
2. Read comitted(LV2): vengono ammesse letture non ripetibili, ossia se una transazione T1 legge un dato A, una transazione T2 può successivamente scrivere sullo stesso dato A prima che T1 abbia effettuato il commit, quindi successive letture di dati A si riferiscono ad un dato diverso. Tuttavia, se T1 legge solo una volta i dati questo livello di isolabilità è adeguato.
3. Repeatable read(LV3): vengono ammesse letture fantasma, ossia se una transazione T1 legge un insieme di dati A e T2 inserisce un nuovo dato nell'insieme A, allora la successiva lettura di A comprende un dato in più.
4. Serializzable(LV4): garantisce un livello completo di serializzabilità, a discapito naturalmente dell'efficienza.

Il savepoint permette di strutturare una transazione. Una transazione può eseguire più savepoint Successivamente si può eseguire, ad esempio, ROLLBACK TO SAVEPOINT B che annulla tutte le modifiche a partire da B. Restano le modifiche effettuate dalla transazione sino al savepoint B. Supponiamo che si concluda con un COMMIT() accolto dal db. Diventeranno persistenti le modifiche sino al SAVEPOINT B e quelle effettuate dopo l'azione di ROLLBACK TO SAVEPOINT B.

20 – Le interrogazioni ricorsive in SQL99. L'algoritmo del punto fisso. Illustrare l'argomento avvalendosi di un esempio di chiusura transitiva.

La sintassi per effettuare un'interrogazione ricorsiva è:

```
WITH RECURSIVE R(A1, … , An) AS
SELECT …..
FROM ….., R
…..
```

R0(A,B) = □ ; i=0;
Repeat
Ri+1 (A B) = f(r(A; B), Ri (A; B));
(A,B) f(r(A,B,...), (A,B));
i=i+1;
Until Ri (A,B) = Ri-1 (A,B);
Dove f() è un'espressione in algebra relazionale senza differenze insiemistiche

Esempio: Data la relazione famiglia(genitore,figlio, dataNascita, ...) Si vuole calcolare la relazione discendenza D(persona,discendente) .

Simulazione delle iterazioni sino al raggiungimento della chiusura transitiva

Famiglia

G	F
a	b
a	c
b	d
d	e

D1

P	D
a	b
a	c
b	d
d	e

D2

P	D
a	b
a	c
b	d
d	e
a	d
b	e
c	e

D3

P	D
a	b
a	c
b	d
d	e
a	d
b	e
c	e
a	e

L'imput dell'esempio è un grafo aciclico.
Il metodo, tuttavia, non impone alcun vincolo:
il grafo può essere ciclico e anche non connesso.

21 – Basi di dati attive. I trigger: definizione. Dati i seguenti schemi: Dipartimento (nomeDip, direttore), Professore (nomeProf, qualifica) dove direttore referenzia (chiave esterna) "Professore", progettare un trigger sull'evento UPDATE OF direttore ON dipartimento che verifichi se la qualifica del nuovo direttore è 'ordinario'. In caso contrario il trigger deve ripristinare il direttore precedente.

I trigger sono operazioni che vengono svolte al verificarsi di determinati eventi (INSERT, UPDATE, DELETE) e seguono lo schema: evento - condizione – azione .
Nel nostro caso:

```
CREATE TRIGGER esercizio
        BEFORE UPDATE OF direttore ON Dipartimento
        REFERENCING NEW ROW neodirettore
        FOR EACH ROW
        WHEN (neodirettore.direttore==professore.nomeProf) AND professore.qualifica='ordinario'
```

[sintassi trigger]:
Lo standard SQL99 prevede la seguente sintassi:

```
CREATE TRIGGER nome_trigger
        BEFORE | AFTER <evento> ON nome_relazione
        [REFERENCING <lista_variabili>]
        [FOR EACH {ROW | STATEMENT}]  //granularità del trigger
        [WHEN condizione]
        <azione>  | BEGIN <azione> END                    //in PSM/SQL

<evento>            ::= INSERT | UPDATE [OF <lista_colonne>] | DELETE
<lista_variabili>   ::= { NEW | OLD} ROW nome_riga,….       //solo row level
                    ::= { NEW | OLD} TABLE nome_tabella,…. //solo AFTER

Ovviamente          NEW non è applicabile all'evento DELETE
                    OLD  non è applicabile all'evento INSERT
```

22 – Definire in ORACLE 2 tavole tipate con referenziamenti incrociati tali che implementino un'associazione 1:1. Ispirarsi al seguente esempio: Dipartimento (nomeDip, direttore), Professore (nomeprof, qualifica, dirige). "Direttore" ha tipo REF verso la tavola "Professore", mentre "dirige" ha tipo "REF" verso la tavola "Dipartimento". Definire, in particolare, la segnatura del metodo "assegna_direttore" della tavola "Direttore" accennando al suo body.

```
CREATE TYPE Dipartimento_T AS OBJECT (
        nomeDip IN VARCHAR(20),
        member procedure assegna_direttore (dir in VARCHAR(20));
        );

CREATE TYPE Professore_T AS OBJECT(
        nomeprof IN VARCHAR(20),
        qualifica IN VARCHAR(20),
        dirige REF Dipartimento_T);


ALTER TYPE Dipartimento_T ADD ATTRIBUTE
        direttore REF Professore_T CASCADE;

CREATE TABLE Dipartimento OF dipartimento_T(
        primary_key(nome_dip));

CREATE TABLE Professore OF Professore_T(
        primary_key(nomeoprof));

CREATE ON REPLACE TYPE BODY Dipartimento_T AS
        MEMBER PROCEDURE assegna_direttore (Dir IN VARCHAR(20)) IS
        begin
                //se l'oggetto era collegato con un dipartimento, devo spezzare il collegamento,
ottengo il riferimento da collegare e creo i due collegamenti
        end assegna_direttore
```

23 – Vincoli dichiarativi di colonna e di tavola in SQL99: (?) commentata con particolare riguardo

al vincolo REFERENCES. Dettagliare il significato di modalità e suo settaggio.

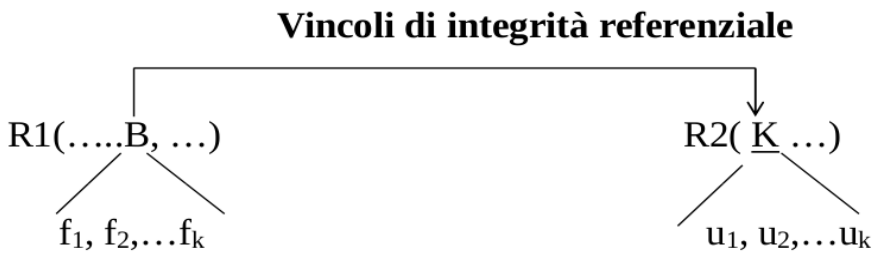
<vinc._di_colonna> ::= [CONSTRAINT <nome_vincolo>] <vincolo colonna> [<modalità>]

<vincolo_colonna> ::= NOT NULL | CHECK(<condizione>) | UNIQUE | PRIMARY KEY | <referenziamento>

<referenziamento> ::= REFERENCES <tavola>[(<colonna>)] [MATCH { SIMPLE | PARTIAL | FULL }] [<azione>]

→ La tavola è uguale, ma in vincolo c'è la lista di colonne.

Abbiamo 3 diverse modalità:



SIMPLE(DI DEFAULT):
Il match è vero se:
- almeno un fi di t1 è nullo
- oppure esiste una tupla t2 con tutti gli attributi corrispondenti uguali
NB.: la tupla referenziata t2, se esiste, è unica.

PARTIAL:
Il match è vero se:
tutti gli fi sono nulli, oppure esiste una t2 tale che in corrispondenza di tutti gli fi non nulli t1[fi] = t2[ui] . Si osservi che t2[ui] <> NULL (componente di chiave primaria)
NB.: più tuple t2 possono soddisfare il vincolo.

FULL:
Il match è vero se:
- tutti gli fi sono nulli
- oppure tutti gli fi sono diversi da nullo e t1[fi] = t2[ui] . Si osservi che t2[ui] <> NULL (componente di chiave primaria) . NB.: la tupla referenziata t2, se esiste, è unica.

Proprietà:
1) FULL più restrittivo di PARTIAL, a sua volta più restrittivo di SIMPLE
2) I tre metodi sono equivalenti se k=1 (chiavi con un solo attributo) oppure tutti gli fi sono diversi da nullo .

Modalità dichiara quando il vincolo deve essere verificato:
modalità immediata, dopo ogni statement (Consistenza a livello di statement)
modalità differita, al termine della transazione (Consistenza transazionale)

Modifica della <modalità>
In una transazione si può modificare la <modalità> richiamando il vincolo tramite il suo nome.

24 – La semantica di CODD del valore nullo: interpretazioni NULL, logica a 3 valori, tavole di verità degli operatori e comportamento della selezione. Influenza del valore nullo sulle espressioni aritmetiche e sulla funzione di aggregazione

Esistono tre interpretazioni
• Dato esistente ma non registrato. (esempio persona.datanascita: posta a NULL, significa che la data di nascita non è registrata - ma esiste perchè ogni persona può avere una data di nascita).
• Dato inesistente. Ad esempio la colonna SecondoNome può essere NULL in una tupla perchè la persona non ha il secondo nome.
• Assenza di informazioni. Ad esempio la colonna Coniuge è NULL perchè non è noto se questo esiste.
La semantica di Codd introduce tre valori di verità: true, false, unknown . (valore sconosciuto, cioè non è possibile stabilire se è vero o falso).

False	Unknown	True
F	U	T
0 < 1 < 2		

Definisce il comportamento della selezione
e quindi anche dei join:

$$\sigma_P(r(A)) = \{t \mid (t \in r(A) \wedge P(t)\}$$

Dove P(t) è un'**espressione proposizionale** con operatori AND, OR e NOT costruiti sui seguenti **atomi**:
$(t[A_i]\ \theta$ *costante)* e $(t[A_i]\ \theta\ t[A_j])$ $\theta \in \{=, <>, >, <, >=, <=\}$

Tavole di verità degli operatori logici:

AND	F	U	T
F	F	F	F
U	F	U	U
T	F	U	T

$AND(x_1, x_2) = \min\{x_1, x_2\}$

OR	F	U	T
F	F	U	T
U	U	U	T
T	T	T	T

$OR(x_1, x_2) = \max\{x_1, x_2\}$

NOT	
F	T
U	U
T	F

$NOT(x) = (2 - x)$

P(t)	C[P(t)]
F	F
U	F
T	T

Se l'espressione di partenza P(t) contiene al più operatori AND e OR il risultato finale equivale all'interpretazione canonica secondo la semantica di Codd.
Se l'espressione di partenza P(t) contiene l'operatore NOT allora la
procedura non è conforme alla semantica di Codd.
Nerlle operazioni aritmetiche, se queste contengono un termine NULL, allora il loro risultato è

NULL. Le funzione di aggregazione: COUNT conta i valori non NULL come qualsiasi altro valore. SUM e AVG ignorano i valori NULL.

25 – La clausula CHECK: come definire la condizione con un'interrogazione e a quali limitazioni è soggetta. Avvalersi di esempi

Il vincolo CHECK è inserito all'interno della definizione di tabella. Si tratta di un vincolo a livello di tabella o di colonna.

```
CREATE TABLE R
(
...
CHECK ( c o n d i z i o n e )
...
);
```

Se abbiamo un vincolo del tipo X+Y<N dove X e Y sono attributi della tavola, è possibile avere un'interrogazione come:

```
CHECK (NOT EXIST(SELECT *
                 FROM T
                 WHERE Y+X>=N))
```

Un'altro impiego utile è quello di esprimere dipendenze funzionali. La clausola CHECK ha comunque delle limitazioni. Prima di tutto, anche se una condizione pu` essere espressa tramite una query (come EXISTS(SELECT ...)) non si può fare riferimento ad altre tabelle o diverse da quella su cui è inserito il vincolo ed in ogni caso tutti le variabili fanno riferimento ai soli attributi della tabella. Inoltre, non si possono utilizzare funzioni dipendenti dal tempo (come CURRENT DATE), l'operatore UNION, le funzioni MIN, MAX, ecc. e le funzioni o procedure implementate all'esterno dei DBMS.

26 – ODMG: Obiettivi e architettura. Tipi, letterali, oggetti e rispettive specifiche con la clausula "interface", "classi", "extends", "extent". Avvalersi di esempi.

ODMG (Object Data Management Group) è un consorzio che si occupa di stabilire standard per le basi di dati Object Oriented. L'architettura si divide in tre parti:

Object Model: modello degli oggetti costituito da moduli che contengono interfacce e classi.

Linguaggi di specificazione: linguaggi che specificano la struttura dati e la loro manipolazione. Tra questi vi sono OQL (Object Query Language) per le interrogazioni ai DBMS ad oggetti, ODL (Object Definition Language) per definire la struttura degli oggetti persistenti secondo l'Object Model (è indipendente dai linguaggi di programmazione) e OIF (Object Interchange Format) che non verrà trattato in questo documento.
Vincoli Binding: prevede una serie di librerie ed eventualmente linguaggi per manipolare gli oggetti (principalmente ODL e OQL) del DBMS in modo trasparente, ossia in modo tale che si abbia la percezione di utilizzare un unico linguaggio. E' disponibile per i linguaggi C++ e Java.

L'Object Model prevede che vi siano due costrutti: Interface e Class. Il costrutto Interface non può istanziare oggetti, mentre Class si. In ogni caso sia Interface che Class contengono definizioni di aspetti statici (letterali e relazioni) e aspetti dinamici (funzioni e eccezioni eventualmente sollevate).

I letterali sono attributi che contengono un valore, mentre le relazioni sono concettualmente come relazioni tra classi o tra entity nel modello ER cioè mettono in relazione uno a molti, molti a uno, uno a uno o molti a molti valori. Ad esempio in Java questo viene realizzato utilizzando gli omonimi costrutti interface (dove gli attributi sono definiti con i metodi get e set) e class. Ad ogni oggetto si possono associare uno o più nomi. Tra le caratteristiche principali abbiamo che gli oggetti persistenti e quelli transienti hanno lo stesso ciclo di vita e si possono applicare le stesse operazioni su di essi. Questa è una differenza sostanziale rispetto alle basi di dati relazionali in cui i dati persistenti sono gestiti solo tramite SQL. L'object model stabilisce anche che si specifichino delle chiavi e si attribuisca un nome alle estensioni delle classi. L'ereditarietà tra interfacce è multipla, mentre tra classi è singola. I vari binding dovranno fornire un preprocessore che, dati i file in ODL crei lo schema rispettivo nella base di dati e richiami il compilatore, che prende in input la parte di dichiarazione delle classi di ODL e l'applicativo generando il programma che comunica con il DBMS.

27 – ODMG3: le principali interfacce e classi dello standard (Collection, Iterator, Transaction, Database). Sintetizzare le funzionalità e l'uso

L'interfaccia Collection contiene le funzioni comuni per tutte le collezioni oggetto (Set, List, ecc.) L'interfaccia Iterator viene utilizzata per definire iteratori monodirezionali. (uno per ogni collezione). Un iteratore può essere anche creato come stabile, cioè non permette che accessi concorrenti (inserimenti o cancellazioni) alla stessa collezione modifichino la lettura dei dati di un processi . Metodi particolari dell'interfaccia Transaction sono join e leave. Questi metodi sono stati introdotti perchè un thread può attivare una sola transazione alla volta. Per rilasciare una transazione attiva si utilizza la funzione leave, per fare entrare un'altra transazione (nello stesso thread) precedentemente sospesa con leave si utilizza join. Notare che la funzione leave non chiude la transazione e tanto mento comporta il rilascio dei lock, ma la sospende. L'interfaccia Database consente consente di effettuare le operazioni sul Database. Questa interfaccia offre tra le altre funzionalità, la possibilità di associare ad una particolare istanza un nome o di rimuovere tale associazione. Sempre la classe Database, permette dunque di ottenere un'istanza dato un nome.

28 – La struttura dell'interrogazione OQL. La costruzione della risposta in accordo con il modello CODL. Le sottointerrogazioni. Il quantificatore esistenziale e universale. Avvalersi di esempi esplicativi.

Il linguaggio OQL (Object Query Language) è un linguaggio funzionale, molto simile ad SQL, utilizzato per effettuare le query in ODMG.

```
select [ distinct ] q
from
q1 in x1 ,
...
qn in xn
[ where p ]
```

q1, ..., qn sono query .
Il predicato p può invece utilizzare tutte le variabili. Questo, oltre a contenere operatori di confronto e booleani, può contenere anche i quantificatori universali ed esistenziali. Possiamo dunque avere un predicato nella forma

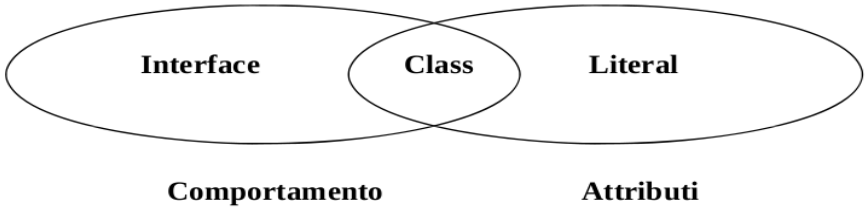
exists x in q : c

dove x è un nome nuovo di variabile, q è una query e c un'espressione condizionale che può fare

uso della variabile x. Esiste anche la versione universale del quantificatore:

forall x in q : c

I **letterali** non hanno identificatori. Sono categorizzati in tipi. Es.: interi, stringhe, tuple,... Sono dichiarati solo gli aspetti statici.
L'**oggetto** è un concetto primitivo; Gli oggetti sono categorizzati in tipi; Ciascun oggetto ha un unico identificatore nella base dati.
Il **comportamento** degli oggetti è definito da un insieme di funzioni con garanzie di completezza rispetto al dominio e consistenza della base dati;
Lo **stato** degli oggetti è definito dai valori delle rispettive proprietà suddivise in attributi e relazioni.
Interface: definisce l'oggetto tramite il suo comportamento astratto. E' un approccio funzionale: tutte le proprietà (attributi, relazioni, metodi) sono dichiarazioni di altrettante funzioni applicabili all'oggetto. NON istanzia oggetti.
Class: definisce l'oggetto avvalendosi degli aspetti sia statici sia dinamici . PUO' istanziare oggetti.



L'estenzione EXTENT è l'insieme di tutte le sue istanze correnti all'interno di una particolare base di dati. Una classe eredita (extends) da al più una classe (ereditarietà semplice). Una classe può ereditare da più interfacce (:<interface>: ereditarietà multipla , ma con il vincolo che le interfacce non contengano funzioni omonime). Le funzioni di una superclasse appaiono con la stessa segnatura (nome, tipo parametri e tipo risultato) nella sottoclasse. Il metodo (body) di una funzione ereditata può essere ridefinito.

29 – La clausula group by ed having. L'attributo standard "partition". Tipo della risposta. Esempi.

Il group by è uguale all'SQL:
select *
from q1 x1, q2 x2,..., qn xn
where p(x1, x2,..., xn)
group by att1:e1,...,atth:eh [having predicate]

Con having si mantengono vere solo le relazioni in cui il predicato è vero. Partition definisce la partizione del group by.

Tipo della risposta: set<struct(att1:Type(e1),...,atth:Type(eh),
partition: bag<struct(x1:Type(x1),..., xn:Type(xn))>)>

esempio: Partiziona gli studenti in base all'età

```
select *
from studenti x
group by incorso: x.eta < 25,
        fuoricorso:x.eta > 25 and x.eta < 28
        veterani:x.eta > 28
```

tipo della risposta:

```
set<struct( incorso:boolean,
            fuoricorso:boolean,
            veterani:boolean, partition : bag<struct(x: studente)>)>
```

[fine]