**Linguaggio SQL**  (Structury Query Language

Linguaggio per basi di dati basate sul modello relazionale

3 versioni : SQL 1, SQL 2, SQL 3

* Si parla di tabelle (non relazioni)
* Il risultato di un’operazione sui dati può restituire una tabella con righe duplicate
* Il sistema dei vincoli è più espressivo
* Il vincolo di integrità referenziale (chiave esterna) è meno stringente

Costituito da due componenti principali:

* DDL (Data Definition Language): contiene i costrutti necessari per la creazione/modifica dello schema della base di dati
* DML (Data Manipulation Language): contiene i costrutti per le interrogazioni e l’inserimento/eliminazione/modifica di dati

Tramite ‘create table’ è possibile costruire una tabella all’interno dello schema, poi per ciascun attributo è possibile specificare, oltre al nome e dominio, un valore di default e i vincoli

È possibile associare i domini (elementari) agli attributi di uno schema:

1. **Caratteri**

Il dominio character consente di rappresentare singoli caratteri o stringhe di lunghezza masssima fissa

character / char [varying][(Lunghezza)]

*character varying (20)*

1. **Numerici esatti**

Consentono di rappresentare valori esatti, interi o con parte decimale di una lunghezza prefissata

numeric [(Precisione[,Scala])])

decimal()

integer()

smallint()

*numeric(4,2)*

La keyword auto\_increment consente di creare dei campi numerici che si auto-incrementano ad ogni nuovo inserimento nella tabella

*integer auto\_increment*

1. **Numerici approssimati**

Consentono di rappresentare valori reali con rappresentazione in virgola mobile

float [(Precisione)]

real

double precision

*float(5)*

1. **Domini temporali**

Consentono di rappresentare informazioni temporali o intervalli di tempo

date [(Precisione)]

time

timestamp

interval [PrimaUnità] [to UltimaUnità]

*time(2)*

*interval month to second*

1. **Domini blob e cblob**

Consentono di rappresentare oggetti di grandi dimensioni come sequenza di valori binari(blob) o di caratteri (cblob)

La dimensione massima del file dipende dalle specifiche implementazioni

Non è possibile specificare interrogazioni sui dati in base al valore del dominio

Tramite ‘domain’ si possono costruire domini di dati a partire da domini elementari

create domain NomeDominio as TipoDati [Valore di default] [Vincolo]

*create domain Voto as smalllint*

*default null*

*check(value >= 18 and valure <= 30)*

Per ciascun dominio o attributo è possibile specificare un valore di default attraverso il costrutto default

*default [valore | user | null]*

valore indica un valore del dominio

user è l’id dell’utente che segue il comando

null è il valore di null

Per ciascun dominio o attributo è possibile definire dei vincoli che devono essere rispettati da tutte le istanze di quel dominio o attributo

Vincoli intra-relazionali:

* **Vincoli generici di ennupla (check)**

Mediante la clausola check è possibile esprime vincoli di ennupla arbitrari

* **Vincolo not null**

Indica che il valore ‘null’ non è ammesso

In caso di inserimento, l’attributo deve essere specificato, a me no che non sia stato specifica un valore di default diverso da ‘null’

* **Vincolo unique**

Impone che l’attributo/i non presenti valori comuni in righe diverse (l’attributo è una **superchiave** della tabella)

*codice smallint unique* per 1 solo attributo

*unique(codice, ufficio)* per più attributi

* **Vincolo primary key**

Impone che l’attributo/i non presenti valori comuni in righe diverse e non assuma valori ‘null’ (l’attributo è una **chiave primaria** della tabella)

*Attributo Dominio [ValDefault] primary key*

*primary key(Attributo1, Attributo2, …)*

A differenza di unique e not null, il vincolo primary key deve apparire una sola volta per tabella

…

Operazioni di interrogazione (read) vengono implementate dal costrutto di ‘**select from where**’

Select (Lista Attributi) From (Lista Tabelle) Where (Condizione)

Es. “ select nome from impiegati where (ufficio = “A”) ”

* Si ottiene una tabella in risultato

**Where** specifica quali righe della tabella devono comparire nel risultato finale (non è necessario). Può contenere un’espressione booleana o una combinazione di espressioni mediante gli operatori and, or, not. È possibile fare confronti fra stringhe usando l’operatore like e l’utilizzo di wildcard:

\_ = carattere arbitrario

% = sequenza di caratteri arbitraria

Così è possibile trovare tutte le stringhe che rispettano un certo pattern

Sempre dentro where, l’operatore ‘***between***’ consente di verificare l’appartenenza ad un certo insieme di valori

I valori NULL non vengono inseriti nel risultato finale ma bisogna usare ***IS NULL*** o ***IS NOT NULL***

**Select** specifica quali colonne delle righe selezionate devono comparire nel risultato finale

L’asterisco ( ***\**** ) indica tutte le colonne della tabella

È possibile ridenominare le colonne del risultato di una query tramite il costrutto ‘***as***’

Es. “ select nome as name, cognome as lastname from impiegati where(nome = “Marco”) “

È possible utilizzare espressioni aritmetiche (semplici) sui valori degli attribute di una select

**From** specifica la lista delle tabelle su cui si deve accedere. Se ci sono più tabelle allora viene effettuato il prodotto cartesiano delle stesse)

È possibile specificare degli alias per i nomi delle tabelle, tramite il costrutto ‘***as***’

Se più tabelle hanno attributi con nomi uguali, si può utilizzare *NomeTabella.NomeAttributo* per far riferimento ad un attributo in maniera non ambigua

Il risultato di una query SQL potrebbe avere righe duplicate e si può usare il costrutto ‘***distinct***’ (nella select) per rimuovere i duplicati nel risultato.

Il costrutto ‘***all***’ (nella select) non rimuovi i duplicati

Il costrutto ‘***order by***’ consente di ordinare le righe del risultato di un’interrogazione in base al valore di un attributo specificato

Gli operatori aggregati si applicano a gruppi di tuple e producono come risultato un solo valore. Vengono inseriti nella select e valutati dopo la clausola where e from

sum / avg / min / max / count

Es. select sum(Attributo) from ListaTabelle Where(Condizione)

Operatore ***groupby*** prende in input una o più colonne, e divide la tabella in gruppi, ognuno caratterizzato da un valore comune dell’attributo specificato nell’operatore

Es. select Dipartimento as DIP, count(\*) as Numero from Strutturati groupby Dipartimento

Costrutto ***having*** per filtrare i gruppi in base a determinate condizioni

Operazioni insiemistiche tra tabelle o tra i risultati di select:

* ***Union***
* ***Intersect***
* ***Except***

Inserire una riga esplicitando i valori degli attributi oppure estraendo le righe da altre tabelle, tramite ***insert***

Es. insert into NomeTabella [ListaAttributi] values (ListaValori)

Rimuovere tutte le righe che soddisfano una condizione tramite ***delete***

Es. delete from Tabella where Condizione

Aggiornare il contenuto di uno o più attributi di una tabella che rispettano una certa condizione, ***update***

Es. update NomeTabella set attributo = expr|select|null|default [where Condizione]

È possibile implementare il join tra tabelle in due modi distinti:

* Inserendo le condizioni del join direttamente nella clausola del where
* Attraverso l’operatore di inner join nella clausola from

Left join = risultato dell’inner join + righe della tabella di sinistra che non hanno un corrispettivo a destra (completate con valori null)

Right join = risultato dell’inner join + righe della tabella di destra che non hanno un corrispettivo a destra (completate con valori null)

Full join = risultato dell’inner join + righe della tabella di sinistra/destra che non hanno un corrispettivo a destra/sinistra (completate con valori null)

Esercizio 6 per compito

Gestisci connessioni

Nuova connessione

Hostname 130.136.2.70 Port 3306

Username: studente

Password: basididatiinfoman2020

…..

QUERY ANNIDATE

Nel where inserisco un’altra query

Es. select codice from strutturati where (stipendio = select max(stipendio) from strutturati)

* Viene eseguita per prima la query più interna

Gli operatori di confronto <, =, > non si possono usare ma si usano ***any***, ***all***

Il costrutto ***in*** restituisce true se un certo valore è contenuto nel risultato

Il costrutto ***exists*** restituisce true se la query interna restituisce un risultato non vuoto

Le query annidate/nidificate possono essere:

* Semplici se le interrogazioni vengono valutate dalla più interna alla più esterna
* Complesse se c’è passaggio di binding tra le interrogazioni, quelle più interne vengono valutate su ogni tupa

VISTE

Rappresentano tabelle virtuali ottenute da dati contenuti in altre tabelle, ogni vista ha un nome, una lista di attributi e si ottiene dal risultato di una select

I dati delle viste non sono fisicamente memorizzati a parte, in quanto dipendono da altre tabelle

Esistono a livello di schema ma non hanno istanze proprie

Le operazioni di aggiornamento di viste potrebbero non essere consentite in alcuni DBMS

Utilizzi:

* Implementare meccanismi di indipendenza tra il livello logico e quello esterno
* Scrivere interrogazioni complesse, semplificandone la sintassi
* Garantire la retro-compatibilità con precedenti versioni dello schema del DB, in caso di ristrutturazione dello stesso (si può mantenere la visione originaria del DB)

CTE (Common Table Expression) rappresentano viste temporanee che possono essere usate in una query come se fossero una vista a tutti gli effetti ma non esistono a livello di schema del DB

Asserzioni (SQL2) sono un costrutto per definire vincoli generici a livello di schema

Es. create assertion NomeAsserzione check Condizione

Consentono di definire vincoli non altrimenti definibili con i costrutti visti fino ad ora

Il vincolo può essere immediato o differito (verificato al termine di una transazione)

Costrutti avanzati SQL3

STORED PROCEDURES

Frammenti di codice SQL che possono specificare un nome, dei parametri in input e dei valori di ritorno

Es. Procedure ModificaStipendio (MatricolaN: varchar(20), StipendioNew: smallint)

update Impiegati  
set Stipendio=StipendioNew   
where Matricola=MatricolaN

Ogni DBMS offre estensioni procedurali diverse

Le basi di dati contengono le procedure per lavorare sulle tabelle -> aumentano efficienza e espressività

Le estensioni procedurali consentono di:

* Aggiungere strutture di controllo al linguaggio SQL (cicli, strutture condizionali if then else,..)
* Dichiarare variabili e tipi di dato user-defined
* Definire funzioni avanzate e ottimizzate che sono ritenute ‘sicure’ dal DBMS

TRIGGER

Procedure automatiche, senza che vengano invocate dall’utente. Meccanismi di gestione della base di dati basati sul paradigma ECA (Evento / Condizione / Azione)

* Evento: manipolazione de dati (insert, delete, update)
* Condizione: predicato booleano
* Azione: sequenza di istruzioni SQL, talvolta procedure SQL specifiche del DBMS

Utilizzo:

* Garantire il soddisfacimento di vincoli di integrità referenziale e/o specificare meccanismi di reazione ad hoc in caso di violazione dei vincoli

Es. un valore nella tabella esterna viene cancellato o modificato

* Specificare regole aziendali (business rules), ossia vincoli generici sullo schema della base di dati

Es. Un impiegato non può avere un aumento di stipendio superiore al 10%, pena annullamento della transazione

Sintassi SQL3

* Modo 🡪 before/after
* Evento 🡪 insert/delete/update
* Referencing 🡪 qui possono essere inserite variabili globali per aumentare l’espressività del trigger
* Livello 🡪 row (Il trigger agisce a livello di righe) statement (il trigger agisce globalmente a livello di tabella)

Due modalità di esecuzione: immediata o differita

PERMESSI

Su ogni risorsa sono definita dei pribiligi (grant). SQL2/SQL3 prevede meccanismi di controllo di accesso alle risorse dello schema del DB

* ***Grant*** consente di assegnare privilegi su una certa risorsa ad utenti specifici
* ***With grant option*** consente di estendere il privilegio di assegnare il grant ad altri utenti
* ***Revoke*** consente di revocare i permessi
* ***Cascade*** revoca i permessi a tutti quelli a cui l’utente aveva concesso i permessi

MySQL non ha limiti espliciti sulla dimensione massima di un database e sul numero di tabelle (ovviamente viene però intaccata l’efficienza)

MySQL ha un’architettura Client-Server della comunicazione

Creare un nuovo utente (locale)

mysql> create user nome@localhost identified by ‘password’

mysql> create database [if not exist] nome\_db;

mysql> drop database [if exists] nome\_db;

create [temporary] table nome\_tabella | nome\_db.nome\_tabella [definizione attributi] [opzioni] [select]

\*\* è possibile generare una tabella valida solo per la sessione corrente (opzione temporary)

\*\* è possibile popolare la tabella con il risultato di una query select da altre tabelle

MySQL supporta diversi tipi di “storage engine” (tipi di tabelle), tra cui:

* INNODBM
* MyISAM

Truncate per eliminare i dati di una tabella

Delimiter = comando per compilatore su come eseguire un pezzo di codice (nei trigger è obbligatorio

Per affrontare problemi di affidabilità (crash) e di concorrenza, si usane le TRANSAZIONI: unità di lavoro elementare che modificano il contenuto di una base di dati

Sono comprese tra ‘start transiction’ e ‘commit work’ / ‘rollback work’

Proprietà delle transazioni:

* Atomicità: la transazione deve essere eseguita con la regola del “tutto o niente”
* Consistenza: la transazione deve lasciare il DB in uno stato consistente, eventuali vincoli di integrità non devono essere violate
* Per verificarla basta simulare le operazioni
* Isolamento: l’esecuzione di una transazione deve essere indipendente dalle altre
* Persistenza: l’effetto di una transazione che ha fatto ‘commit work’ non deve essere perso

Schedule = sequenza di operazioni di lettura/scrittura di tutte le transazioni così come eseguite sulle basi di dati

Schedule seriale = se le azioni di ciascuna transazione appaiono in sequenza, senza essere inframezzate da azioni di altre transazioni

In un sistema reale, le transazioni vengono eseguite in concorrenza per ragioni di efficienza/scalabilità

Schedule serializzabile = se produce lo stesso risultato di un qualunque scheduler seriale delle stesse transazioni

Sistemi NOSQL

Proprietà:

* Strumenti solitamente open-source
* Non hanno uno schema
* Non supportano operazioni di join
* Non implementano le proprietà ACID delle transazioni
* Scalabili orizzontalmente
* Possono gestire grandi insiemi di dati
* Dati replicabili

Motivazioni legate alla diffusione dei database NOSQL

**Big Data**: grandi quantità di dati, difficili da gestire tramite le tecnologie tradizionali (il termine è molto vago)

È accompagnato da quattro caratteristiche (4V): Volume, Velocità, Varietà, Valore

[…]

**Limitazioni del modello relazionale**

Teorema di Brewer (CAP Theorem): un sistema distribuito può soddisfare al massimo solo due delle tre proprietà:

* Consistency = tutti i nodi della rete vedono gli stessi dati
* Availability = il servizio è sempre disponibile
* Partion tolerance = il servizio continua a funzionare correttamente anche in presenza di perdita di messaggi o di partizionamenti della rete

Caso1: Consistency + Availability 🡪 il sistema non funziona correttamente in caso di perdita di messaggi

Caso 2: Availability + Partion Tolerance 🡪 repliche del dato non aggiornate

Caso 3: Consistency + Partion Tolerance 🡪 la query non produce risposta

DBMS NoSQL:

* Chiave/Valore: dati di un DB come liste di coppie chiave/valore
* Document-oriented: per gestione di dati eterogenei e complessi (semi-strutturati)
* Column-oriented: dati in sequenze di colonne invece che righe, più flessibile e efficiente nello storage
* Graph-oriented: dati sotto forma di grafi (nodi=attributi/righe; archi=relazioni tra attributi/righe)

Documenti: strutture dati modellate attraverso il linguaggio JSON (chiave/valore)

Modello relazionale: tabelle,righe,colonne

MongoDB: collezioni, documento json, campi

**MODELLO E-R**

**Entità:** classe di oggetti della realtà di interesse, con proprietà comuni e esistenza autonoma. Viene rappresentata con un rettangolo

**Relazione**: legame logico fra due o più entità, rilevante nel sistema che si sta modellando

Graficamente viene rappresentata attraverso un rombo collegato ad entità

Ad ogni relazione è associato un nome che la identifica nello schema, per convenzione si usano nomi al singolare (non verbi, se possibile)

L’istanza di una relazione è una combinazione di istanze dell’entità che prendono parte all’associazione

Relazione ricorsiva: quando coinvolge più istanze della stessa entità, in questo caso si può definire un ruolo per ciascun ramo della relazione

**Attributo**: proprietà elementare di un’entità o di una relazione del modello, ognuno viene definito su un dominio specifico

È possibile definire attributi composti come unione di attributi affini di una certa entità/relazione, sono rappresentati da un cerchio

**Cardinalità delle relazioni**: coppia di valori (min,max) che specificano il numero di occorrenze della relazione a cui ogni occorrenza di entità può partecipare

Si usano solo due valori per il minimo:

* ‘0’ per partecipazione opzionale dell’entità
* ‘1’ per partecipazione obbligatoria dell’entità

Si usano solo due valori per il massimo:

* ‘1’ se viene coinvolta al massimo 1 istanza
* ‘N’ se non c’è un limite massimo

In base al valore della cardinalità massima delle entità E1 e E2 coinvolte in una relazione R, si distinguono tre casi:

* Relazioni uno-ad-uno: cardMax(E1)=1, cardMax(E2)=1
* Relazioni uno-a-molti: cardMax(E1)=1, cardMax(E2)=N oppure cardMax(E1)=N, cardMax(E2)=1
* Relazioni molti-a-molti: cardMax(E1)=N, cardMax(E2)=N

**Cardinalità degli attributi**

**Identificatori**: strumenti per identificare in maniera univoca le istanze delle entità

Ogni entità deve avere due identificatori:

* Interno: composto da attributi dell’entità

Gli attributi che formano l’identificatore devono avere cardinalità (1,1)

* Esterno: composto da attributi dell’entità + entità esterna

Può comprendere anche attributi dell’entità corrente e l’entità esterna deve essere in relazione(1,1) con l’entità corrente

* Servono a modellare le situazioni in cui un’istanza di un’entità ha valori univoci solo all’interno di un certo contesto

**Generalizzazione**: definisce una gerarchia tra entità basata sul concetto di ereditarietà

Tutti gli attributi di un entità sono anche attributi delle sue generalizzazioni e le generalizzazioni partecipano a tutte le relazioni dell’entità

Non tutti i vincoli sono esprimibili nel modello E-R 🡪 tabelle delle business rules

La costruzione di uno schema concettuale deve tener conto di alcune proprietà:

* Correttezza: utilizzo corretto dei costrutti
* Completezza: rappresentazione di tutti i dati di interesse, descritti nel documento di specifica

**Strategie di progettazione**

* Strategia top-down: raffinamenti successivi a partire da uno schema iniziale molto astratto
* Strategia bottom-up: le specifiche iniziali vengono divise in componenti sempre più piccole e successivamente vengono integrati gli schemi fra loro
* Strategia inside-out: si individuano i concetti importanti e poi, da questi, si procede verso i correlati (macchia a d’olio)
* Strategia mista: combinazione delle precedenti

Pattern di progettazione

Bisogna attenersi alle regole concettuali:

1. Se un concetto ha proprietà significative e descrive oggetti con esistenza autonoma 🡪 entità
2. Se un concetto correla due o più entità 🡪 relazioni
3. Se un concetto è un caso particolare dell’altro 🡪 generalizzazioni

Analisi di progettazione

È importante analizzare l’efficienza dal punto di vista prestazionale

Indici di prestazione:

* Costo operazionale: numero di entità/associazioni mediamente visitate per implementare una certa operazione sui dati
* Occupazione di memoria: spazio di memoria necessario per memorizzare i dati

La tavola dei volumi fornisce una stima del numero di occorrenze di entità/relazioni presenti nel modello ER

La tavola delle operazioni definisce:

* L’insieme delle operazioni che devono essere implementate sui dati
* La tipologia delle operazioni (interattive/batch)
* La frequenza delle operazioni

Data un operazione O di tipo T, definiamo il suo costo c(Ot) come:

**c(OT )=f (OT )⋅wT ⋅(α⋅NCwrite +NCread )**

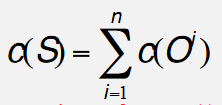
**f (OT ):** frequenza dell’operazione

**NCread:** numero di accessi in lettura a componenti dello schema

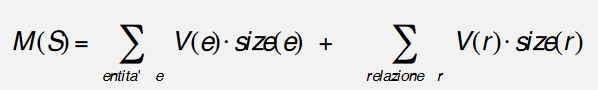
**NCwrite:** numero di accessi in scrittura a componenti dello schema

**wT:** peso dell’operazione

**α:** coefficiente moltiplicativo delle operazioni in scrittura

Dato uno schema S e un insieme di operazioni sui dati O1, O2, …, On, con costi c(O1),c(O2),…,c(On), il costo dello schema è definito come: 

L’obbiettivo del progettista è determinare lo schema E-R di costo minimo

È possibile stimare l’occupazione di memoria dello schema: 

V(e), size(e) = tabella dei volumi e dimensioni in termini di occupazione di memoria dell’entità e

**Progettazione logica**

**Eliminazione delle generalizzazioni**

Soluzioni:

1. Accorpamento delle entità figlie nell’entità genitore (con relativi attributi/relazione)

* Introduce valori nulli ed un attributo aggiuntivo, è conveniente quando non ci sono troppe distinzioni tra entità figlie e genitore

1. Accorpamento delle entità genitore nelle entità figlie

* È possibile solo se la generalizzazione è totale, introduce valori nulli, è conveniente quando ci sono operazioni che coinvolgono solo le entità figlie e non la genitore

1. Sostituzione delle generalizzazione con relazioni tra entità genitore ed entità figlie

* Non introduce valori nulli, è conveniente quando ci sono operazioni che si riferiscono solo ad entità. Ma si presenta la necessità di introdurre vincoli

**Eliminazione degli attributi multi-valore**

Gli attributi multivalore non sono presenti nel modello logico, ma possono essere sotituiti introducendo una relazione uno-a-molti

Partizionamento/accorpamento di concetti

È possibile ridurre il numero di accessi:

* Separando attributi di un concetto che vengono acceduti separatamente 🡪 partizionamenti
* Raggruppando attributi di concetti diversi acceduti insieme 🡪 accorpamenti
* È necessario avere indicazioni sul volume dei dati per effettuare partizionamenti/accorpamenti

**Scelta degli identificatori**

Nei casi di entità con più identificatori, è necessario sceglierne uno tramite alcuni criteri:

* Evitare attributi con valori nulli
* Scegliere l’identificatore minimale
* Preferire identificatori interni ad esterni che coinvolgono molte entità
* Preferire identificatori utilizzati da molte operazioni per l’accesso all’entità

**Analisi delle ridondanze**  
Potrebbero essere presenti informazioni significative ma derivabili da altre già presenti nel modello E-R

Vantaggi ridondanze:

* Operazioni sui dati (spesso) più efficienti

Svantaggi ridondanze:

* Maggiore occupazione di memoria
* Maggiore complessità degli aggiornamenti

[…]

**NORMALIZZAZIONE**

Verifica dello schema che ho prodotto

Ridondanza:

* Ridondanza concettuale: non ci sono duplicazioni dello stesso dato, ma sono memorizzate informazioni che possono essere ricavate da altre già contenute nel DB
* Ridondanza logica: esistono duplicazioni sui dati che possono generare anomalie nelle operazioni sui dati

Cause:

* Traduzioni non corrette nel modello logico
* Errori durante la progettazione concettuale 🡪 meglio ristrutturare lo schema E-R partizionando l’entità ed introducendo delle relazioni uno-a-molti o molti-a-molti

Per risolvere le anomali si introduce il concetto di Dipendenza Funzionale (DF):

data una tabella su uno schema R(X) e due attributi Y e Z di X. Esiste la dipendenza funzionale Y->Z se per ogni coppia di tuple t1 e t2 di r con t1[Y] = t2[Y], si ha anche che t1[Z]=t2[Z]

Le DF sono una generalizzazione del vincolo di chiave (e di superchiave).

Data una tabella con schema R(X), con superchiave K, esiste un vincolo di dipendenza funzionale tra K e qualsiasi attributo dello schema r.

Uno schema R(X) si dice in forma normale di Boyce e Codd se per ogni dipendenza funzionale Y 🡪Z definita su di esso, Y è una superchiave di R(X)

* Se una relazione è in FNBC, non presenta le anomalie e ridondanze viste fin qui
* Se una relazione non è in FNBC, bisogna trasformarla (normalizzarla) se possibile in FNBC

Una tabella r è in terza forma normale (3FN) se per ogni dipendenza funzionale X 🡪 A dello schema, almeno una delle seguenti condizioni è verificata:

* X contiene una chiave K di r (X è una superchiave di r)
* A appartiene ad almeno una chiave K di r

Se uno schema rispetta la FNBC allora rispetta anche la 3FN, ma non viceversa. La 3FN è sempre ottenibile ma la FNBC no

Algoritmo di normalizzazione per:

* Semplificare l’insieme di dipendenze F, rimuovendo quelle non necessarie e trasformando ogni dipendenza in modo che nella parte destra compaia un singolo attributo
* Raggruppare gli attributi coinvolti nelle stesse dipendenze e costruire le relazioni corrispondenti
* Assicurarsi che almeno una delle relazioni prodotte contenga la chiave della relazione originaria

A -> Apache

M -> MySQL

P -> Php

Vengono usati per la progettazione di applicazioni web

PHP  
Non è necessario dichiarare una variabile prima del suo utilizzo, i nomi vengono preceduti da $

Es. $testVariable=5;

Array non sono necessariamente omogenei

Es. $myarray = array(“Hello”,2,3.5,”3”);

echo $myarray[1];

$myarray[0]=”Ciao”; //modificare un elemento

$myarray[] = “nuovo”; //aggiungere un elemento

If..else.. uguale al java

While uguale al java

All’interno di una funziona possono essere definite variabili locali, con l’istruzione global si può accedere a variabili globali

Si può invocare una funziona anche specificando un numero di parametri inferiore a quelli previsti dall’intestazione della funzione

Metodo $\_GET per passare parametri in ingresso, parametri inseriti in appendice alla URL dello script PHP

Metodo $\_POST come get ma i parametri sono inviati (non in forma visibile) insieme alla richiesta http

Con il simbolo 🡪 si può accedere a metodi/attributi di un oggetto

Es. $myObject = new myClass();

$myObject 🡪 a = $myObject 🡪 a+5

Supporta l’ereditarietà multipla, una classe può estendere più classi genitori