Progettazione Concettuale: Il modello Entità-Relazioni

- Un Sistema Informativo gestisce informazioni strutturate.
- Le Basi di Dati (Database) sono sistemi per gestire dati in modo integrato e flessibile, riducendo ridondanza e incoerenza.
- Il DBMS (DataBase Management System) è un software che gestisce collezioni di dati in modo affidabile e persistente.
- Il Modello dei dati è l'insieme di elementi per organizzare e strutturare i dati, includendo modelli relazionali, gerarchici, reticolari e NoSQL.

Metodologia di Progetto e Schema Concettuale

- La metodologia di progetto di un database include diverse fasi, iniziando dai "Requisiti del DB".
- La "Progettazione Concettuale" è una rappresentazione di alto livello dei requisiti sui dati raccolti nel documento URD (User Requirements Document).
- Lo Schema Concettuale descrive dettagliatamente dati, relazioni e vincoli senza includere dettagli implementativi.
- Tipicamente, lo Schema Concettuale viene definito utilizzando il modello Entità-Relazioni (Entity-Relationship).

Elementi del Modello Entità-Relazioni (E-R)

- L'Entità rappresenta una classe di oggetti (astratti o tangibili) distinguibili nel "mini mondo" di interesse.
- Gli Attributi sono le proprietà specifiche che caratterizzano ciascuna entità.
- Un'occorrenza di un'entità è un singolo esemplare della classe, caratterizzato dai valori assunti dagli attributi.

Tipi di Attributi Particolari

- Un Attributo Composto può essere suddiviso in parti più piccole, ciascuna con una propria specificità (es., un indirizzo può includere Via, Civico, CAP, Città, Stato).
- Un Attributo Multivalore può assumere più valori per ciascuna occorrenza di un'entità (es., "Telefoni" o "titoli di studio").
- Un Attributo Derivato può essere determinato a partire da altri attributi memorizzati (es., l'età derivata dalla data di nascita).
- Il Dominio di un Attributo è l'insieme dei valori che possono essere assegnati a un attributo di un'entità (es., età tra 16 e 65 anni).

- Un attributo A di un'entità E con insieme di valori V è definito come A : E -> P(V), dove P(V) è l'insieme dei sottoinsiemi di V.
- I Valori Null vengono creati quando non è possibile determinare un valore per un attributo in un'occorrenza di entità, il loro significato può essere "Ignoto" (valore esistente ma sconosciuto) o "Non applicabile".

Chiavi di un'Entità

- Gli Attributi chiave permettono di distinguere tra occorrenze della stessa entità, assumendo un vincolo di unicità sugli attributi.
- Il Vincolo di chiave implica l'esistenza di un sottoinsieme di attributi (anche uno solo) di un'entità, la cui combinazione di valori è unica per ogni occorrenza.
- Questo sottoinsieme è chiamato Chiave.
- La proprietà di unicità è un vincolo sullo schema dell'entità e vale sempre, indipendentemente dal particolare insieme di occorrenze.
- La chiave viene definita in base alle proprietà del "mini mondo" rappresentato dalla base di dati.
- Un'entità può avere più di un attributo che verifica il vincolo di chiave (es., Numero di telaio e Numero di targa per un'automobile).
- La proprietà di un attributo di essere chiave dipende dal contesto (es., un numero di matricola può non essere sufficiente a identificare un esame superato se lo studente ha sostenuto più volte lo stesso esame).

Relazioni nel Modello E-R

- Una Relazione è un'associazione o un legame logico tra due o più entità.
- Il Grado di una relazione si riferisce al numero di entità che vi partecipano (es., "Impiegato-Lavora-Dipartimento" ha grado 2, "Fornitore-Consegna-Prodotto-Magazzino" ha grado 3).
- Le Relazioni Ricorsive si verificano quando un'entità è in relazione con se stessa, e in questi casi si utilizzano nomi di ruolo espliciti per chiarire la partecipazione (es., un impiegato può essere "responsabile" di un altro impiegato "subordinato").

Vincoli Strutturali sulle Relazioni

- Le relazioni possono avere vincoli che limitano le combinazioni delle entità partecipanti.
- Questi vincoli dipendono dal contesto e dal "mini mondo" che la relazione rappresenta.
- La Cardinalità specifica il numero di occorrenze di relazione a cui le occorrenze di entità possono partecipare.
- I tipi comuni di cardinalità sono 1:1 (uno a uno), 1:N (uno a molti), e M:N (molti a molti).

- La Partecipazione specifica se l'esistenza di un'occorrenza di entità dipende dalla sua relazione con un'altra occorrenza di entità.
- La Partecipazione Totale implica una dipendenza esistenziale, dove ogni occorrenza di entità partecipa alla relazione (es., un progetto DEVE essere gestito da un dipartimento).
- La Partecipazione Parziale implica che un'occorrenza di entità PUÒ partecipare alla relazione ma non è obbligatorio (es., un impiegato può essere direttore di un dipartimento, ma non necessariamente).
- Esistono notazioni alternative per rappresentare i vincoli, inclusi i numeri minimo e massimo di entità partecipanti (min=0 per partecipazione parziale, min>0 per totale).

Progettazione del Modello E-R e Relazioni con Grado Superiore a 2

- La progettazione di un modello E-R per un database aziendale richiede l'analisi dei requisiti.
- I requisiti di esempio includono l'organizzazione in dipartimenti, la gestione di attività, il tracciamento delle informazioni anagrafiche degli impiegati, i rapporti gerarchici, l'assegnazione ai dipartimenti e l'lavoro su progetti, oltre al tracciamento dei familiari.
- Possono esistere relazioni ternarie (grado > 2), ma sono molto improbabili relazioni con grado > 3.
- Molti sistemi reali non consentono di "mappare" relazioni con grado > 2, rendendo necessario rappresentare le relazioni ternarie utilizzando relazioni binarie.
- Quest'operazione può causare perdita di informazioni se non eseguita con attenzione.

Gerarchie ISA

- Le gerarchie ISA ("is-a" o "è un") rappresentano legami logici tra un'entità padre (più generale) e una o più entità figlie (specializzazioni).
- La Generalizzazione Totale significa che ogni occorrenza della classe padre è anche un'occorrenza di almeno una delle classi figlie (altrimenti è parziale).
- La Generalizzazione Esclusiva significa che ogni occorrenza della classe padre è al più una sola occorrenza di una delle classi figlie (altrimenti è sovrapposta).

Considerazioni Finali sul Modello E-R

- Il modello E-R deve essere costruito dopo un'adeguata raccolta dei requisiti.
- Permette una descrizione di alto livello dei dati e aiuta a chiarire ulteriormente i requisiti.
- Permette di esplicitare numerosi vincoli.
- La progettazione del modello E-R è soggettiva, con molteplici scelte possibili che devono essere ponderate adeguatamente.

- Un modello E-R ben costruito consente una mappatura immediata nel modello logico relazionale.
- Sono comunque necessari raffinamenti e verifiche, come la normalizzazione, per assicurare la qualità del database.

Introduzione al Modello Relazionale

- Il modello relazionale si basa sul lavoro di Codd (~1970).
- È attualmente il modello più usato nei database (es., Oracle, Informix, IBM, Microsoft).
- Si basa su una struttura dati semplice e uniforme: la relazione.
- Il modello relazionale ha solide basi teoriche.

Concetti Base del Modello Relazionale

- Una base di dati è una collezione di relazioni.
- Informalmente, una relazione può essere considerata una tabella.
- Ogni riga di una tabella rappresenta una collezione di valori di dati collegati.
- Il nome della relazione e dei suoi attributi (colonne) aiutano a comprendere il significato dei valori nelle righe.
- Tutti i valori in una colonna appartengono allo stesso tipo e dominio.
- Un Dominio è un insieme di valori atomici (indivisibili).
- Ad ogni dominio è associato un tipo di dato o formato.
- Lo Schema di una relazione (o componente intensionale) è definito da un nome R e una lista di attributi (A1, A2, ..., An), dove ogni attributo corrisponde a un dominio.
- Il Grado di una relazione è il numero dei suoi attributi.
- L'Istanza di una relazione (o componente estensionale) è un insieme di tuple (riga) (t1, t2, ..., tm), dove ogni tupla è una lista ordinata di valori che appartengono al dominio dell'attributo o a NULL.
- La Cardinalità di una relazione è il numero di tuple nella sua istanza.

Note sulla Definizione del Modello Relazionale

- Le tuple sono definite come insiemi, implicando l'assenza di un ordinamento particolare sulle tuple stesse.
- I valori all'interno di una tupla sono definiti come una "lista ordinata", quindi esiste un ordinamento.
- Le relazioni sono definite a livello logico, pertanto non hanno un ordinamento predefinito sulle tuple.
- Si può considerare una tupla come un valore associato direttamente al nome dell'attributo corrispondente.
- Uno schema di relazione può essere interpretato come una dichiarazione o un'asserzione, dove ogni tupla è un esempio di asserzione.
- Uno schema di relazione può anche essere interpretato come un predicato; i valori di ciascuna tupla soddisfano il predicato.

Vincoli del Modello Relazionale

- Vincolo di dominio: i valori di ciascun attributo devono essere atomici e appartenere al dominio corrispondente.
- Vincolo di chiave: per ogni relazione R, esiste un sottoinsieme di attributi (sk) tale che i valori di sk siano distinti per ogni tupla.
- Il sottoinsieme sk è chiamato superchiave.
- Una chiave è un sottoinsieme di attributi che è una superchiave minimale (non si può rimuovere alcun attributo mantenendo l'unicità).
- Una relazione può avere più sottoinsiemi di attributi che verificano le proprietà di chiave candidata.
- La chiave designata è chiamata chiave primaria (PK).
- Integrità di entità: nessuna chiave primaria può assumere un valore nullo.
- Integrità referenziale: una tupla in una relazione R1 che si riferisce a un'altra relazione R2 deve riferirsi a una tupla esistente in R2.
- Questo vincolo è espresso tramite il concetto di chiave esterna (foreign key - FK).
- Una FK in R1 è un sottoinsieme di attributi che ha lo stesso dominio della PK di un'altra relazione R2.
- Un valore di FK in una tupla t1 di R1 deve essere identico al valore di PK di una tupla t2 in R2, oppure nullo.

Operazioni nell'Algebra Relazionale

- L'algebra relazionale è un linguaggio procedurale in cui le operazioni sono specificate descrivendo la procedura da seguire.
- È un'algebra chiusa, ovvero gli operatori sono definiti su relazioni e producono relazioni, consentendo l'ulteriore manipolazione dei risultati.
- Gli operatori principali includono:
    - Unari: selezione, proiezione, ridenominazione.
    - Binari: join.
- Esistono anche operazioni tipiche dell'insiemistica: unione, intersezione, differenza e prodotto cartesiano.

Selezione e Proiezione

- La Selezione consente di scegliere un sottoinsieme di tuple di una relazione che soddisfano una condizione.
- La condizione di selezione può includere confronti tra attributi o con costanti, e può combinare condizioni logiche (AND, OR, NOT).
- L'operazione di selezione viene applicata individualmente a ogni tupla e mantiene lo schema della relazione originale, lasciando il grado invariato.
- La cardinalità risultante della selezione è minore o uguale a quella della relazione di partenza.
- La selezione è commutativa.
- La Proiezione consente di selezionare un sottoinsieme di colonne (attributi) di una relazione.
- La proiezione generalmente modifica il grado (numero di attributi) e la cardinalità della relazione.
- La proiezione non è commutativa.
- È possibile concatenare operazioni, come una proiezione su una selezione, per ottenere risultati specifici.

Ridenominazione e Operazioni Insiemistiche

- La Ridenominazione consente di rinominare uno o più attributi di una relazione, utile per facilitare le operazioni insiemistiche.
- Modifica solo i nomi degli attributi, non i valori.
- **Unione (R∪S)**: unisce tutte le tuple presenti in R, S o entrambe.
    - È applicabile a relazioni compatibili (tuple omogenee con attributi identici o appartenenti allo stesso dominio).

    - Il grado dell'unione è pari al grado di R e S. La cardinalità è minore o uguale alla somma delle cardinalità di R e S.
- **Intersezione (R∩S)**: produce una relazione che include solo le tuple presenti sia in R che in S.
    - Il grado dell'intersezione è pari al grado di R e S. La cardinalità è minore o uguale al minimo delle cardinalità di R e S.
- **Differenza (R-S)**: produce una relazione che include solo le tuple presenti in R ma non in S.
    - Il grado della differenza è pari al grado di R e S. La cardinalità è minore o uguale alla cardinalità di R.

Prodotto Cartesiano e Join

- Il **Prodotto Cartesiano** (R x S) è un'operazione binaria che non richiede la compatibilità delle relazioni partecipanti.
    - Il grado della relazione risultante è la somma dei gradi delle due relazioni.
    - La cardinalità è pari al prodotto delle cardinalità delle due relazioni.
- Il **Join** è un'operazione per combinare coppie di tuple provenienti da relazioni collegate in singole tuple.
    - Permette di elaborare le "associazioni" tra relazioni.
    - Un join con una condizione generica è chiamato *theta join*.
    - Quando la condizione è un'uguaglianza, è chiamato *equi join*; in questo caso l'attributo usato per la condizione è incluso due volte.

Tipi di Join

- Il **Natural Join** è un tipo di join che riporta una sola volta l'attributo di comparazione nella tabella risultante.
    - Utilizza una condizione di uguaglianza su attributi con lo stesso nome.
- L'**Outer Join** inserisce nella relazione risultante le tuple che contribuiscono al risultato, estendendole con valori NULL se non ci sono controparti appropriate.
    - **LEFT Outer Join**: include tutte le tuple della relazione a sinistra.
    - **RIGHT Outer Join**: include tutte le tuple della relazione a destra.
    - **FULL Outer Join**: include tutte le tuple da entrambe le relazioni.
- Il **Self Join** è l'operazione di join di una tabella con una copia di se stessa, spesso utilizzando alias per distinguere le copie.
    - È utile per relazioni ricorsive o per confrontare elementi all'interno della stessa entità.

Raffinamento e Normalizzazione degli Schemi Relazionali

- La modellazione E-R consente di descrivere schemi relazionali, ma è necessario ricorrere a metodi formali per garantire la scelta di "buoni" schemi.
- Questi metodi sono utilizzati per verificare e raffinare lo schema.

Relazioni con Anomalie

- Una singola relazione può presentare anomalie quando rappresenta informazioni eterogenee (es. impiegati con stipendi e progetti con bilanci).
- Un esempio di anomalia è la ridondanza, dove lo stipendio di un impiegato si ripete in tutte le tuple a lui relative.
- Le anomalie di aggiornamento avvengono quando la modifica di un valore (es. stipendio) richiede l'aggiornamento di più tuple, aumentando il rischio di incoerenza.
- Le anomalie di cancellazione si verificano quando la rimozione di un'informazione (es. un impiegato che smette di partecipare a tutti i progetti) comporta la perdita involontaria di altre informazioni (es. i dati del dipartimento associato a quel progetto).
- Le anomalie di inserimento si presentano quando non è possibile inserire un nuovo dato (es. un impiegato senza progetto) a causa di vincoli sulle altre informazioni.

Linee Guida Informali per la Progettazione

- Scegliere gli attributi in modo da semplificare la descrizione del loro significato, evitando di riunire attributi di entità diverse nello stesso schema.
- Ridurre la ridondanza per minimizzare lo spazio di archiviazione e prevenire anomalie di aggiornamento, inserimento e cancellazione.
- Ridurre al minimo i valori NULL nelle tuple, poiché possono essere interpretati in diversi modi (non applicabile, sconosciuto, non inserito); in molti casi è preferibile creare nuove relazioni per gestire i valori NULL.
- Evitare le tuple spurie (informazioni non corrette) che possono essere generate da un join naturale, assicurando che i join utilizzino chiavi primarie o esterne.

Dipendenze Funzionali (FD)

- Per studiare sistematicamente gli aspetti di integrità e ridurre le anomalie, si introduce il vincolo di dipendenza funzionale.
- Una dipendenza funzionale $X \rightarrow Y$ esiste se, per ogni istanza di una relazione R, ogni volta che i valori del sottoinsieme di attributi X coincidono in due tuple, anche i valori del sottoinsieme Y devono coincidere.
- Le anomalie sono spesso legate a FD specifiche (es., Impiegato → Stipendio, Progetto → Bilancio).

- Le anomalie sono causate dalla presenza di concetti eterogenei nello schema (es., proprietà degli impiegati, dei progetti e della chiave Impiegato Progetto).
- F+ è la chiusura di F, l'insieme delle FD implicate da F.
- Gli assiomi di Armstrong (Riflessività, Incremento, Transitività) sono regole complete e corrette per determinare le FD.
- Altre regole utili includono l'Unione ($X \rightarrow Y$, $X \rightarrow Z$ implica $X \rightarrow YZ$) e la Decomposizione ($X \rightarrow YZ$ implica $X \rightarrow Y$ e $X \rightarrow Z$).
- Un insieme F di dipendenze funzionali è minimo se ogni FD ha un solo attributo sulla destra, non è possibile rimuovere alcuna FD da F senza perdere equivalenza, e la parte sinistra di ogni FD è minima.
- L'obiettivo è ottenere schemi privi di ridondanza, anche se il calcolo di F+ è computazionalmente pesante e una certa ridondanza può essere accettata per motivi di efficienza.

Forme Normali

- Le Forme Normali sono metodi formali per analizzare le relazioni e permettono la decomposizione di relazioni in schemi con proprietà migliori.
- Per un buon progetto di database, è necessario verificare altre proprietà come il lossless join (decomposizione senza perdite) e la conservazione delle dipendenze.
- **Prima Forma Normale (1NF)**:
    - Coincide con la definizione pratica di relazione.
    - Il dominio di ogni attributo deve contenere valori atomici e ogni valore in una tupla deve essere singolo.
    - Esiste una chiave primaria che identifica univocamente ogni tupla.
    - Il metodo di decomposizione applica il mapping ER relazionale.
- **Seconda Forma Normale (2NF)**:
    - Una relazione è in 2NF se è in 1NF e ogni attributo non chiave ha una dipendenza funzionale piena dall'intera chiave primaria (non solo da una sua parte).
- **Terza Forma Normale (3NF)**:
    - Una relazione è in 3NF se è in 2NF e, per ogni FD $X \rightarrow Y$, X è una superchiave o ogni attributo in Y è un attributo primo (fa parte di una chiave).
    - Non ci sono attributi che dipendono transitivamente da altri attributi non chiave.
- **Forma Normale di Boyce-Codd (BCNF)**:
    - Una relazione è in BCNF se è in 3NF e per ogni FD $X \rightarrow Y$, X è una superchiave di R.
    - Le uniche FD non ovvie sono vincoli di chiave.

- Una relazione in BCNF è anche in 3NF, ma una relazione in 3NF potrebbe non essere in BCNF e presentare ancora ridondanza.
- 3NF è comunque accettabile e permette di decomporre una relazione in relazioni 3NF che supportano lossless-join e conservano le dipendenze.

Problemi delle Decomposizioni e Conservazione delle Dipendenze

- Le interrogazioni possono diventare più costose a causa dei join necessari.
- Può verificarsi una perdita di informazione (tuple spurie) a causa dei join.
- Il controllo delle dipendenze potrebbe richiedere l'uso di join.
- Una **Decomposizione Senza Perdite** (Lossless Join) garantisce che una relazione R(X) su un insieme di attributi X = X1 ∪ X2 possa essere ricostruita con precisione tramite il join delle sue proiezioni su X1 e X2, senza generare tuple spurie.
    - R si decompone senza perdite se l'insieme degli attributi comuni (X1 ∩ X2) è una chiave per almeno una delle relazioni decomposte.
- La **Conservazione delle Dipendenze** garantisce che sullo schema decomposto siano soddisfatti gli stessi vincoli dello schema originale, così da poterli verificare.
    - Ogni dipendenza funzionale dello schema originale deve coinvolgere attributi che compaiono tutti insieme in uno degli schemi decomposti.
    - Le relazioni decomposte mantengono la capacità di rappresentare i vincoli di integrità e di rilevare aggiornamenti non consentiti.
    - Esiste un algoritmo che garantisce il lossless join e la conservazione delle dipendenze con decomposizione in 3NF, anche se non in BCNF.

Riassunto sulla Normalizzazione

- Le relazioni in BCNF sono prive di ridondanza.
- Se non è possibile ottenere una decomposizione BCNF con lossless join e conservazione delle dipendenze, ci si può accontentare di una 3NF.
- È fondamentale considerare le prestazioni e le probabili interrogazioni quando si progettano i database.

SQL come DDL e DML

- SQL (Structured Query Language) è sia un Linguaggio di Definizione Dati (DDL) che un Linguaggio di Manipolazione Dati (DML).
- Il DDL permette di definire lo schema del database (tabelle, vincoli, domini, viste, etc.).
- Il DML consente la modifica e l'interrogazione dell'istanza di una base di dati (inserimento, aggiornamento, cancellazione, interrogazione).

- Esistono diverse versioni di SQL che hanno introdotto miglioramenti, come SQL-89, SQL-92 (entry, intermediate, full) e SQL-99.

Definizione dei Domini Elementari in SQL

- SQL supporta sei gruppi di domini elementari da cui è possibile costruire nuovi domini.
- **Stringhe**:
    - **character** consente di definire singoli caratteri o stringhe.
    - **character [varying][(num_char)][character set nome_set]** per stringhe di lunghezza fissa o variabile con set di caratteri specifici.
    - Esempi: **Codice_fiscale char(13)**, **prodotto_greco varchar(100) character set Greek**.
- **Valori Booleani**:
    - Il tipo **bit**, introdotto in SQL2, assume valori 0 e 1.
    - **bit [varying][(num_bit)]** per bit singoli o stringhe di bit.
    - Esempi: **Sequenza bit(5)**, **Codice varbit(16)**.
- **Valori Numerici Esatti (1/3)**:
    - **integer** e **smallint** per numeri interi o in virgola fissa.
    - **decimal [(precision[,scale])]** e **numeric[(precision[,scale])]** per numeri in virgola fissa.
    - **precision** indica il numero di cifre significative, **scale** il numero di cifre dopo la virgola.
    - **numeric** garantisce precisione esatta, **decimal** è il minimo per tale precisione.
- **Valori Numerici Approssimati (2/3)**:
    - **Float[(precision)]**, **Real**, **Double precision** per numeri in virgola mobile.
    - **Float** consente una precisione esplicita (numero di cifre della mantissa), mentre per **Real** e **Double precision** dipende dal sistema di calcolo.
    - La precisione di **double precision** è maggiore o uguale a quella di **real**.
- **Date/Time (3/3)**:
    - **Date**, **Time**, **Timestamp** consentono di rappresentare istanti di tempo.
    - Sono strutturati: data (aaaa:mm:gg), ora (HH:MM:SS).
    - La precisione predefinita è 0 secondi per **time** e 6 microsecondi per **timestamp**.
    - Il fuso orario (**Time zone**) si riferisce all'ora di Greenwich.
- **Time Intervals (4/3)**:

- **Interval FirstTimeUnit [to LastTimeUnit]** rappresentano intervalli di tempo, valori relativi per incrementare/decrementare date o timestamp.
- Esempi: **interval year(5) to month** (fino a 99.999 anni e 11 mesi), **interval day(4) to second(6)** (fino a 9.999 giorni, 23 ore, 59 minuti e 59.999999 secondi).

Definizione di Nuovi Domini e Schemi

- Nuovi domini possono essere definiti a partire da quelli elementari (**CREATE DOMAIN**).
- **CREATE DOMAIN DomainName AS DataType [DefaultValue][Constraint]**.
- I valori predefiniti (**DEFAULT**) possono essere **generic** (scelto dal sistema), **user** (ID dell'utente che effettua l'aggiornamento), o **null**.
- Uno schema di database è un insieme di domini, tabelle, indici, asserzioni, viste e autorizzazioni.
- **CREATE SCHEMA [SchemaName] [[authorization] AuthorizedName] {DefiningElements}**.
- I **DefiningElements** possono essere definiti successivamente. Esempio: **CREATE SCHEMA azienda AUTHORIZATION antonio**.

Definizione delle Tabelle e Vincoli in SQL

- Le tabelle sono definite specificando attributi e vincoli (**CREATE TABLE**).
- **CREATE TABLE RelationName(AttributeName Domain [DefaultValue][constraints]{, AttributeName Domain [DefaultValue][constraints]} FurtherConstraints)**.
- I **constraints** includono **PRIMARY KEY** e **UNIQUE**.
- **PRIMARY KEY (AttributeName{, AttributeName })** specifica la chiave primaria composta, i cui valori non possono essere NULL.
- **UNIQUE (AttributeName {, AttributeName })** impone l'unicità degli attributi su cui è applicato, definendo una chiave candidata non primaria.

Vincoli di Integrità Inter-Relazionali in SQL

- Stabiliscono vincoli di integrità referenziale tra diverse tabelle.
- **FOREIGN KEY (AttributeName {,AttributeName}) REFERENCES TableName(AttributeName {,AttributeName})**.
- I campi della chiave esterna (FK) devono corrispondere a una chiave (preferibilmente primaria) nella tabella di riferimento.
- I valori della FK in una tabella **R** devono corrispondere a valori esistenti nella tabella **S** o essere **NULL**.
- La corrispondenza tra gli attributi avviene in base all'ordinamento dichiarato.

- In caso di violazioni, l'azione standard è il rifiuto (**no action**).
- Si possono definire diverse reazioni a violazioni dei vincoli inter-relazionali:
    - **ON DELETE**: azioni in caso di cancellazione di una tupla riferita.
    - **ON UPDATE**: azioni in caso di modifica di una chiave primaria riferita.
- Le azioni specifiche includono:
    - **cascade**: le modifiche si propagano (**DELETE** o **UPDATE** tutte le tuple corrispondenti vengono eliminate o aggiornate).
    - **set null**: il valore cancellato/modificato viene sostituito con **NULL**.
    - **set default**: il valore cancellato/modificato viene sostituito con un valore di default.
    - **no action**: la cancellazione/modifica è inibita.
- La clausola **CHECK (Condition)** viene usata per imporre condizioni arbitrarie sugli attributi di una tabella.

Modifica e Cancellazione di Schemi e Tabelle in SQL

- Le modifiche agli schemi e ai domini si effettuano con il comando **ALTER**.
- **ALTER DOMAIN DomainName <set default DefaultValue | drop default | add constraint ConstraintDefinition | drop constraint ConstraintName >**.
- Le modifiche alle tabelle si effettuano con **ALTER TABLE**.
- **ALTER TABLE TableName <alter column ColumnName <set default DefaultValue | drop default > | add constraint ConstraintDefinition | drop constraint ConstraintName | add column ColumnName | drop column ColumnName >**.
- Per le cancellazioni si usa il comando **DROP**.
- **DROP <schema|table|domain|view|assertion> ItemName [restrict|cascade]**.
- Di default, **RESTRICT** impedisce la cancellazione se ci sono oggetti dipendenti non vuoti.
- **CASCADE** esegue una cancellazione a cascata, rimuovendo tutti gli oggetti specificati e quelli ad essi collegati.

Query Semplici - Sintassi SQL (DQL)

- L'interrogazione è specificata in modo dichiarativo, definendo le caratteristiche del risultato desiderato.
- La struttura base è **SELECT AttrExpr [[as] Alias]{, AttrExpr [[as] Alias]} FROM TableName [[as] Alias]{, TableName [[as] Alias]} [WHERE condition]**.
- La clausola **SELECT** sceglie le colonne da visualizzare.
- La clausola **FROM** elenca le tabelle su cui operare.

- La clausola **WHERE** filtra le righe che soddisfano una condizione.
- L'uso di **AS Alias** è utile per rinominare colonne o tabelle, specialmente quando si usano espressioni o si hanno nomi di colonne identici in tabelle diverse.

Target List e Operatori Aritmetici

- Le quattro operazioni aritmetiche (+, -, *, /) si applicano a campi di tabelle, valori numerici e funzioni aggregate.
- Esempi di interrogazione includono il calcolo dello stipendio annuale (**stipendio_mensile*12**) e dello stipendio totale (**stipendio_mensile + extra_mensile**).

ALL e DISTINCT nella Target List

- **SELECT [ALL | DISTINCT] AttrExpr [[as] Alias]{, AttrExpr [[as] Alias]} FROM TableName [[as] Alias]{, TableName [[as] Alias]} [WHERE condition]**.
- **DISTINCT** elimina i duplicati nel risultato (es., **SELECT DISTINCT cognome** restituirà cognomi unici).
- **ALL** (default) include tutti i duplicati.

Clausola WHERE (Operatori a Valore Singolo e Multiplo)

- **Operatori di Confronto (=, <>, >, >=, <, <=)**:
  - **<condizione del where> ::= <espressione> OP <valore>**
- **Valori NULL**:
  - **IS [NOT] NULL** per controllare se un valore è nullo.
- **Operatori Logici (AND, OR)**:
  - **<condizione del where> ::= <espressionelogica> {<AND|OR> <espressionelogica>}**
- **LIKE**:
  - Usato per la ricerca di pattern nelle stringhe.
  - Lo _ (underscore) è un carattere jolly per un singolo carattere, **%** per zero o più caratteri.
- **Operatori a Valori Multipli**:
  - **BETWEEN <valore infer> AND <valore super>**: verifica se un valore è compreso in un intervallo.
  - **IN <valore1,valore2,..,valoreN>**: verifica se un valore è presente in una lista di valori.

Condizioni Complesse con SUBQUERY

- Le subquery (o sottointerrogazioni) possono essere utilizzate nella clausola **WHERE** per condizioni più complesse.
- **IN**: verifica se un valore è presente nell'insieme restituito da una subquery.
- Operatore di confronto: confronta un valore con il singolo valore restituito da una subquery.
- **ALL** ed **ANY**: confrontano un valore con tutti (**ALL**) o almeno uno (**ANY**) dei valori restituiti da una subquery.

Clausola FROM: Combinazione di Tabelle

- Il prodotto cartesiano unisce tutte le righe di due tabelle.
- **SELECT * FROM TAB1, TAB2**.
- Quando le tabelle in **FROM** hanno colonne con lo stesso nome, è necessario usare gli alias per specificare a quale tabella si riferisce la colonna (es., **TAB1.Col1**).

JOIN - Sintassi e Tipi

- **FROM TableName [[as] Alias] { [<tipo di join>] join TableName [[as] Alias] on <condizione di join>}**.
- **<tipo di join>** può essere **inner** (default), **right outer**, **left outer**, **full outer**.
- **Inner Join**: equivalente alla combinazione di tabelle con **FROM** e **WHERE**.
- **Outer Join**: consente di visualizzare tutte le righe di una o entrambe le tabelle, estendendo con **NULL** le righe senza corrispondenza.
    - **LEFT OUTER JOIN**: mantiene tutte le righe della tabella sinistra.
    - **RIGHT OUTER JOIN**: mantiene tutte le righe della tabella destra.
    - **FULL OUTER JOIN**: mantiene tutte le righe di entrambe le tabelle.
- **Natural Join**: un **inner join** implicito basato su attributi con lo stesso nome.

Query con Raggruppamento (GROUP BY, HAVING, ORDER BY)

- **SELECT AttrExpr [[as] Alias]{, AttrExpr [[as] Alias]} FROM TableName [[as] Alias]{, TableName [[as] Alias]} [WHERE condition] [GROUP BY Attr {, Attr}] [HAVING condition] [ORDER BY AttrExpr [asc|desc] {, AttrExpr [asc|desc] }]**.
- Ordine di esecuzione:
    1. Le righe vengono selezionate in base alla clausola **WHERE**.
    2. Le righe selezionate vengono raggruppate in base alla clausola **GROUP BY**.

3. Per ciascun gruppo, vengono calcolati i risultati delle funzioni aggregate.
4. I gruppi vengono filtrati in base alla clausola **HAVING**.
5. I gruppi vengono ordinati sulla base della clausola **ORDER BY**.

Funzioni Aggregate

- **count(<* | [all|distinct] lista di attributi >)**: conta il numero di righe o valori.
    - **count(*)** conta tutte le righe.
- **avg**, **sum**, **max**, **min** (per media, totale, massimo, minimo dei valori).
    - **sum** e **avg** si applicano solo a valori numerici.
    - **max** e **min** anche a stringhe e date.
- **DISTINCT** nelle funzioni aggregate esclude i duplicati; **ALL** (default) li include.
- I valori **NULL** vengono sempre scartati nel calcolo delle funzioni aggregate (eccetto **count(*)**).
- Le funzioni aggregate non possono essere concatenate direttamente (es., **sum(avg(...))**).
- È possibile utilizzare più funzioni aggregate nella **target list** e nella clausola **ORDER BY**.

Operatori Insiemistici SQL

- **UNION**, **EXCEPT** (o **MINUS**), **INTERSECT**.
- **SELECTsql UNION | EXCEPT | INTERSECT [ALL] SELECTsql**.
- **UNION**: restituisce i valori di due insiemi in un unico insieme.
- **EXCEPT**: restituisce solo i valori del primo insieme non contenuti nel secondo.
- **INTERSECT**: restituisce solo i valori contenuti sia nel primo che nel secondo insieme.
- Di default, gli operatori insiemistici eliminano i duplicati; **ALL** li conserva.
- I nomi dei campi delle due **SELECT** non devono coincidere ma devono essere dello stesso numero e tipo.
- L'uso delle parentesi è importante per definire l'ordine di esecuzione delle subquery con operatori insiemistici.

Predicato EXISTS

- **<condizione del where> ::= [NOT] EXISTS <subquery a valori multipli>**.
- La subquery dopo **EXISTS** restituisce l'intera tabella dalla clausola **FROM** e il predicato si basa sulle righe.
- Se la subquery restituisce almeno una riga, il predicato è **TRUE**.

- Se la subquery è vuota, il predicato è **FALSE**.

Operatori Equivalenti

- SQL offre diverse sintassi per risolvere le stesse interrogazioni.
- Alcune equivalenze comuni:
    - **IN** è equivalente a **=ANY**.
    - **NOT IN** è equivalente a **<>ALL**.
    - **BETWEEN val_inf AND val_sup** è equivalente a **>=val_inf AND <=val_sup**.
    - **EXCEPT** è equivalente a **NOT IN**.
    - **EXISTS** (per le righe) è equivalente a **IN** (per le colonne/valori).

INSERT, DELETE, UPDATE (DML)

- **INSERT**: inserisce nuove tuple in una tabella.
    - **INSERT INTO TableName [ListaAttributi] VALUES (Lista di valori) | SELECTsql**.
    - È possibile specificare una lista di valori o utilizzare il risultato di una **SELECT** per inserire più righe.
- **DELETE**: cancella tuple da una tabella.
    - **DELETE FROM TableName [WHERE condition]**.
    - Senza la clausola **WHERE**, svuota completamente la tabella.
    - Con **WHERE**, cancella solo le tuple che soddisfano la condizione.
- **UPDATE**: modifica i valori degli attributi nelle tuple esistenti.
    - **UPDATE TableName SET attributo = < espressione | selectSQL | null | default > { , attributo = < espressione | selectSQL | null | default >} [WHERE condition]**.
    - Permette di modificare la chiave primaria o altri attributi, anche basandosi sul risultato di un'espressione o di una subquery a valore singolo.