[Appunti Basi di Dati]

Appunti del corso "Basi di Dati" dell'Università degli Studi di Genova

Lorenzo Vaccarecci

UNIVERSITÀ DEGLI STUDI DI GENOVA

Dipartimento di Informatica
Università degli Studi di Genova

2024

Indice

CAPITOLO 1

Modello Relazionale

1.1 Introduzione

Le interrogazioni sulle relazioni possono essere espresse in due formalismi:

- **Algebra relazionale**: le interrogazioni vengono espresse usando operatori specifici alle relazioni.
- **Calcolo relazionale**: le interrogazioni vengono espresse usando formule logiche.

1.2 Relazioni

Un dominio è un insieme (anche infinito) di valori. Indicheremo con $\mathcal{D}$ l'insieme di tutti i domini.

1.2.1 Definizione: Prodotto cartesiano

Siano $D_1, D_2, \ldots, D_k \in \mathcal{D}$ con k domini. Il prodotto cartesiano indicato con $D_1 \times D_2 \times \ldots \times D_k$, è definito come l'insieme $\{(v_1, v_2, \ldots, v_k) \,|v_1 \in D_1, \ldots, v_k \in D_k\}$.
Gli elementi appartenenti al prodotto cartesiano sono detti tuple. Il prodotto cartesiano ha **grado** k.

1.2.2 Definizione: Relazione

Una relazione di k domini è un sottoinsieme finito del prodotto cartesiano, ha **grado** k quindi ogni tupla ha k componenti. La **cardinalità** di una relazione è il numero di tuple appartenenti alla relazione. Una relazione è sempre un insieme finito. Non vi sono tuple duplicate.
La coppia (nome di attributo, dominio) è detta attributo. L'uso di attributi permette di denotare le componenti di ogni tupla per nome piuttosto che per posizione.

1.2.3 Definizione: Schema di relazione

- R un nome di relazione
- $\{A_1, A_2, \ldots, A_n\}$ un insieme di nomi di attributi
- $dom : \{A_1, A_2, \ldots, A_n\} \rightarrow \mathcal{D}$ una funzione totale che associa ad ogni nome di attributo il corrispondente dominio.

La coppia $(R(A_1, A_2, \ldots, A_n), dom)$ è uno schema di relazione. U_r denota l'insieme dei nomi di attributi di R, ovvero $U_r = \{A_1, A_2, \ldots, A_n\}$.

1.2.4 Definizione: Schema di base di dati

Siano $S_1, S_2, \ldots, S_n$ schemi di relazioni distinti, $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ è detto schema di base di dati.

1.2.5 Definizione: Tuple e relazione

Una tupla t definita su una relazione R è un insieme di funzioni totali che associano all'attributo di nome A_i un valore del dominio di tale attributo. Una relazione definita su uno schema di relazione è un insieme finito di tuple definite su tale schema. Tale relazione è anche detta istanza dello schema. $t = [A_1 : v_1, A_2 : v_2, \ldots, A_n : v_n]$ dove $v_i \in dom(A_i)$ con $i = 1, \ldots, n$. Notazione: $t[A_i]$ indica il valore dell'attributo A_i(quindi v_i) nella tupla t.

1.3 Valori nulli

Un aspetto importante nella modellazione dei dati riguarda il fatto che non sempre sono disponibili tutte le informazioni sulle entità del dominio applicativo che vengono rappresentate nella base di dati. L'approccio adottato è quello di introdurre un valore speciale, detto valore nullo, il quale denota la mancanza di un valore.
Nella trattazione assumiamo di denotare il valore nullo con il simbolo '?'. Il valore nullo è un valore accettato in tutti i domini.

1.4 Chiavi

Una chiave di una relazione è un insieme di attributi che distingue fra loro le tuple della relazione. Più formalmente:

1.4.1 Definizione: Chiave e super-chiave

Sia R uno schema di relazione. Un insieme $X \subseteq U_R$ di attributi di R è chiave di R se verifica le seguenti proprietà:

1. Univocità: nella relazione non ci possono essere due tuple distinte che abbiano lo stesso valore per tutti gli attributi della chiave X.

2. Nessun **sottoinsieme proprio** di X verifica la proprietà (1).

Un insieme di attributi che verifica la proprietà (1) ma non la (2) è detto super-chiave di R. Una super-chiave può essere una chiave della relazione.
In una relazione ci possono essere più insiemi di attributi che soddisfano le due proprietà. In tal caso si parla di chiavi candidate. **Una relazione ha sicuramente almeno una chiave (sia primaria che super)**. Nel caso in cui ci sono più chiavi candidate, una di queste viene scelta come chiave primaria su cui il DBMS ottimizza le operazioni.
Un criterio per scegliere la chiave primaria è quello di scegliere la chiave più piccola in termini di numero di attributi o quella più usata nelle interrogazioni. Una chiave non può contenere valori nulli.

1.4.2 Definizione: Chiave esterna

Sia R_1 ed R_2 due relazioni, sia X una chiave per R_1 e Y una chiave per R_2 tale che Y e X contengano lo stesso numero di attributi e di dominio compatibile (*es. interi e reali sono compatibili*). X è una chiave esterna di R_1 su R_2 se per ogni tupla t di R_1 esiste una tupla t' di R_2 tale che $t[X] = t'[Y]$. R_1 viene detta relazione **referente** e R_2 relazione **riferita**.
Vincolo di integrità refernziale: se una tupla t di R_1 fa riferimento ai valori della chiave di una tupla t' di R_2, allora t' deve esistere in R_2. Può essere violata da inserimenti e modifiche nella relazione referente e da cancellazioni e modifiche nella relazione riferita.
Una relazione può contenere più chiavi esterne, eventualmente anche sulla stessa relazione e possono assumere valori nulli.
Notazione:

$$R_1\left(\dots, chiave\_esterna^{R_2}, \dots\right)$$

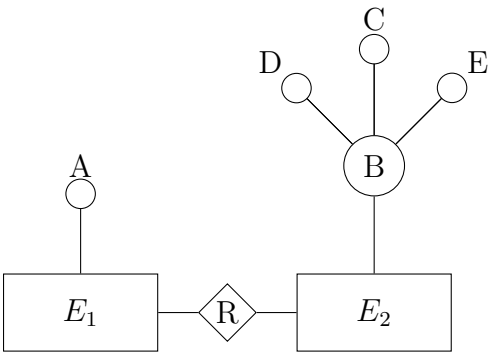
CAPITOLO 2

Modello ER

2.1 Introduzione

Il termine diagramma ER indica la rappresentazione grafica di uno schema concettuale ER. Gli elementi principali di un diagramma ER sono:

- Entità: una collezione di oggetti della realtà che vogliamo modellare che possiedono caratteristiche comuni, graficamente viene rappresentata da un rettangolo
- Relazioni (o associazione): rappresenta un legame logico tra più entità, graficamente viene rappresentata da un rombo
- Attributi: rappresenta una proprietà posseduta da un'entità o da una relazione, è mono-valore se può assumere al più un valore (*es. la matricola di uno studente*), è multi-valore altrimenti (*es. i numeri di telefono di un'azienda*). Può essere a sua volta essere formato da *sotto-attributi* rendendolo composto, un attributo composto può avere domini di diverso tipo (domini composti)



Le istanze di un'associazione sono un sottoinsieme del prodotto cartesiano delle entità coinvolte, non possono esistere duplicati. Le associazioni sono classificate in base al loro grado (il numero di entità in relazione):

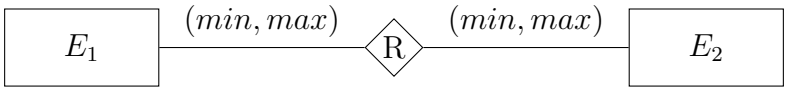
- **Unaria**: un'entità è in relazione con se stessa
- **Binaria**: due entità sono in relazione
- **n-aria**: n entità sono in relazione

Le informazioni sui domini degli attributi non sono direttamente rappresentabili in un diagramma ER, ma possono essere specificate nella documentazione a corredo dei diagrammi ER.

2.2 Vincoli di integrità

2.2.1 Vincoli di cardinalità

I vincoli di cardinalità per associazioni stabiliscono il numero minimo e massimo di istanze di un'associazione a cui le istanze delle entità coinvolte nella associazione possono partecipare. Graficamente, vengono rappresentati come coppia di valori interi (min, max) collocata vicino alla linea che connette l'associazione con ciascuna entità che mette in relazione.



(min, max):

- $min = 0$: l'entità non è obbligata a partecipare all'associazione
- $min = 1$: l'entità è obbligata a partecipare all'associazione
- $max = 1$: l'entità può partecipare al massimo una volta all'associazione
- $max = n$: l'entità può partecipare al massimo n volte all'associazione

Se in un diagramma ER non è specificata la cardinalità di un'associazione, si assume che il vincolo sia $(0, n)$.
Date due entità E_1 e E_2 se:

- $max_{E_1} = max_{E_2} = 1$ allora l'associazione è uno a uno
- $max_{E_1} = 1, max_{E_2} = n$ allora l'associazione è uno a molti
- $max_{E_1} = max_{E_2} = n$ allora l'associazione è molti a molti

Le cardinalità possono essere specificate anche per gli attributi, permettendo di specificare se un attributo è mono o multi-valore, e se è obbligatorio o meno. Se non viene specificata la cardinalità di un attributo, si assume che sia $(1, 1)$.

2.2.2 Vincoli di identificazione

Definire un vincolo d'identificazione per un'entità significa specificare un insieme di attributi e/o entità che posseggono la proprietà di identificare univocamente le istanze dell'entità (identificatori o chaivi).

- Identificatore interno: è costituito da uno o più attributi dell'entità stessa
- Identificatore misto: è costituito da attributi dell'entità e da attributi di altre entità
- Identificatore esterno: è costituito da attributi di altre entità
- Identificatore semplice: è costituito da un solo attributo
- Identificatore composto: è costituito da più attributi

Un'entità le cui istanze vengono identificate mediante l'associazione con altre entità viene chiamata entità debole.

2.3 Gerarchie di generalizzazione

Nel modello ER è possibile organizzare le entità in gerarchia di generalizzazione definendo un insieme di entità dette figlie come specializzazione di un'altra entità detta padre rappresentante le proprietà in comune a tutte le entità figlie.

- Generalizzazione totale: ogni istanza dell'entità padre è anche un'istanza di **almeno** una delle entità figlie
- Generalizzazione parziale: se un'istanza dell'entità padre non è un'istanza di nessuna delle entità figlie
- Generalizzazione esclusiva: se un'istanza dell'entità padre è un'istanza di **al più** una delle entità figlie
- Generalizzazione condivise: se un'istanza dell'entità padre può essere un'istanza di **più** entità figlie

Le generalizzazioni possono essere di quattro tipi diversi: totali esclusive, totali condivise, parziali esclusive e parziali condivise.

CAPITOLO 3

Progettazione Logica

3.1 Introduzione

L'obiettivo principale della progettazione logica è tradurre uno schema ER in uno schema relazionale equivalente. Si articola in due fasi:

- Fase di ristrutturazione dello schema ER: questa fase prevede l'eliminazione dallo schema ER di tutti quei costrutti non direttamente rappresentabili nel modello relazionale, anche questa fase si può suddividere in più sotto-fasi:
  - analisi della ridondanza
  - partizionamento/accorpamento di entità
  - eliminazione degli attributi composti e multi-valore
  - eliminazione delle gerarchie di generalizzazione
- Fase di traduzione dello schema ER: in questa fase, lo schema ER restituito dalla fase di ristrutturazione viene tradotto in un equivalente schema relazionale. La traduzione non è sempre univoca.

3.2 Fase di ristrutturazione

3.2.1 Analisi della ridondanza

Uno schema ER presenta delle ridondanze quando un'informazione viene rappresentata sia esplicitamente nello schema sia può essere derivata da altre informazioni presenti nello schema. Nei diagrammi ER la presenza di ridondanza dovrebbe essere limitata solo a quei casi in cui sia possibile ottenere un significativo beneficio in termini di tempo di esecuzione delle interrogazioni.

3.2.2 Partizionamento/accorpamento di entità

Lo schema ER può essere ulteriormente ristrutturato partizionando od accorpando entità ed associazioni sulla base dell'analisi del carico di lavoro. Un'entità E può essere partizionata in due entità E_1, E_2 una delle quali identificata esternamente dall'altra, collegate mediante un'associazione uno a uno. Questa operazione può essere conveniente quando alcune operazioni frequenti coinvolgono solo un sottoinsieme degli attributi di E. Viceversa, due entità E_1, E_2 collegate da un'associazione uno a uno, possono essere accorpate in un'unica entità contenente gli attributi di E_1 e E_2 nel caso in cui operazioni frequenti abbiano la necessità di accedere ad entrambi gli insiemi di attributi.

3.2.3 Eliminazione degli attributi composti e multi-valore

Il modello relazionale consente la specifica solo di attributi semplici e mono-valore. E' quindi necessario ristrutturare lo schema ER generato dalla fase di progettazione concettuale per eliminare eventuali attributi composti e multi-valore in esso presenti. L'eliminazione di un attributo composto A da un'entità E può avvenite in due modi:

- Eliminando i sotto-attributi di A e considerando l'attributo composto come un attributo semplice
- Considerando tutti i sotto-attributi di A come attributi di E

Quest'ultima soluzione richiede ovviamente una ridefinizione del dominio dell'attributo. La modellazione di attributi multi-valore mediante attributi a valore semplice richiede la definizione di una nuova entità, collegata all'entità di partenza tramite un'opportuna associazione, in cui l'attributo multi-valore è rappresentato mediante un attributo mono-valore che identifica l'entità. Il vincolo di cardinalità dell'associazione è quello dell'attributo multi-valore oggetto di ristrutturazione.

3.2.4 Eliminazione delle gerarchie di generalizzazione

Consideriamo un'entità E generalizzazione di un insieme di entità $E_1, \ldots, E_n$

- Eliminazione delle entità figlie: Le entità figlie vengono eliminate ed i loro attributi vengono inseriti nell'entità padre come attributi opzionali.
  - Generalizzazione totale: l'attributo portato al padre non può essere nullo
  - Generalizzazione parziale: un valore nullo indica un'istanza dell'entità padre che non era istanza di alcuna delle entità figlie
  - Generalizzazione condivise: l'attributo sarà multi-valore in quanto un'istanza dell'entità padre potrebbe essere istanza di più entità figlie
- Eliminazione dell'entità padre: consiste nell'eliminare l'entità padre E e nell'inserire i suoi attributi in ciascuna delle entità figlie. Tale procedimento è applicabile solo nel caso in cui la generalizzazione sia **totale**.
  - Generalizzazione esclusiva: è necessario aggiungere un vincolo per indicare che non possono esistere istanze di due entità figlie distinte aventi lo stesso valore per gli identificatori.

  Ogni associazione a cui partecipava l'entità padre l'entità padre viene inoltre sostituita con n nuove associazioni, una per ogni entità figlia.
- Sostituzione della generalizzazione con associazioni: le entità coinvolte nella generalizzazione non vengono modificate, mentre la gerarchia di generalizzazione viene sostituita da n associazioni uno a uno, ognuna delle quali lega l'entità padre con una diversa entità figlia. Le entità figlie sono identificate esternamente dall'entità padre e partecipano obbligatoriamente alle associazioni create mentre la partecipazione dell'entità padre è opzionale.
  - Generalizzazione esclusiva: un'istanza del padre non può partecipare contemporaneamente a due o più associazioni

– Generalizzazione totale: ogni istanza dell'entità padre deve partecipare obbligatoriamente ad almeno un'associazione

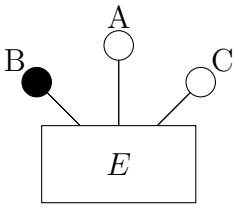
In generale, la scelta di accorpare le entità figlie nell'entità padre comporta uno spreco di memoria per la presenza dei valori nulli. La soluzione di eliminare l'entità padre consente un risparmio di memoria rispetto alla soluzione di eliminare le entità figlie in quanto evita il problema dei valori nulli.

3.3 Fase di traduzione

La fase di traduzione può essere suddivisa nelle seguenti sotto-fasi:

- Traduzione delle entità
- Traduzione delle associazioni
- Traduzione dei vincoli di integrità
- Ottimizzazioni finali

3.3.1 Traduzione delle entità

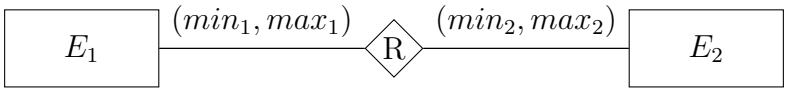


→ E(B,A,C);

Nel caso di chiavi esterne si scrivono in fondo (di solito) e in corsivo indicando anche la tabella di provenienza per una maggiore comprensione della traduzione: E(K,...,FK^{E_i}).
I criteri per la scelta della chiave primaria si possono sintetizzare come segue:

- Non devono contenere valori nulli
- Devono essere composti da pochi attributi
- Sono gli attributi più utilizzati nelle interrogazioni per accedere alle entità

3.3.2 Traduzione delle associazioni



Consideriamo che E_1 ha gli attributi A,B e E_2 ha gli attributi C,D. La chiave di E_1 è K e la chiave di E_2 è L.
Casi:

- Associazione uno a uno: si "elimina" una delle due entità e si accorpa con l'altra: $E_1(\underline{K, L}, A, B, C, D)$
- Associazione uno a molti: l'entità "da sola" riceve la chiave dell'altra entità come chiave esterna: $E_1(\underline{K}, A, B, L), E_2(\underline{L}, C, D, K^{E_1})$
- Associazione molti a molti: si crea una nuova relazione con le chiavi delle entità coinvole e gli eventuali attributi dell'associazione: $R(\underline{K, L})$

CAPITOLO 4

Linguaggio SQL

4.1 Tipi

- `INTEGER`: interi a 32 bit
- `SMALLINT`: interi a 16 bit
- `BIGINT`: interi a 64 bit
- `NUMERIC`: numeri decimali, usa due parametri: la precisione (il numero totale di cifre) e la scala (il numero di cifre dopo la virgola) → `NUMERIC(precisione,scala)` i valori di default sono 1 (precisione) e 0 (scala)
- `DECIMAL`: come `NUMERIC` ma le cifre possono essere $\geq$ p
- `REAL`: numeri in virgola mobile a precisione singola
- `DOUBLE PRECISION`: numeri in virgola mobile a precisione doppia
- `FLOAT`: numeri in virgola mobile a precisione personalizzata → `FLOAT(p)` con $1 \leq p \leq n$ dove n dipende dall'implementazione
- `CHARACTER`: stringhe di lunghezza fissa → `CHARACTER(n)`, di default è 1
- `CHARACTER VARYING`: stringhe di lunghezza variabile → `CHARACTER VARYING(n)` dove n è la lunghezza massima
- `DATE`: data in formato `YYYY-MM-DD`
- `TIME`: tempo in formato `HH:MM:SS`, può essere specificata la precisione dei secondi → `TIME(p)` con $0 \leq p \leq 6$
- `TIMESTAMP`: concatenazione di `DATE` e `TIME`
- `INTERVAL`: durata temporale in riferimento ad uno o più dei qualificatori tra `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND` → `INTERVAL 'valore' qualificatore`
- `BOOLEAN`: valore booleano `TRUE`,`FALSE` o `UNKNOWN`
- `BLOB`: dati binari di lunghezza variabile
- `CLOB`: dati di testo di lunghezza variabile

4.2 Creazione di tabelle

```
1        CREATE TABLE nome_tabella(
2            nome_colonna tipo_colonna,
3            ...
4        );
5
```

4.2.1 Obbligatorietà di colonne

Per la specifica dell'obbligatorietà di una colonna si utilizza la clausola `NOT NULL`:

```
CREATE TABLE nome_tabella(
    nome_colonna tipo_colonna NOT NULL,
    ...
);

```

4.2.2 Chiavi

Per la specifica di una chiave primaria si utilizza la clausola `PRIMARY KEY`:

```
CREATE TABLE nome_tabella(
    nome_colonna tipo_colonna PRIMARY KEY,
    ...
);

```

Per le chiavi alternative si usa `UNIQUE`. Se si hanno chiavi composte da più attributi:

```
CREATE TABLE nome_tabella(
    nome_colonna1 tipo_colonna,
    nome_colonna2 tipo_colonna,
    ...
    PRIMARY KEY(nome_colonna1,nome_colonna2)
);

```

(stessa cosa per `UNIQUE`).
Per la specifica di una chiave esterna si utilizza la clausola `FOREIGN KEY`:

```
CREATE TABLE nome_tabella(
    nome_colonna tipo_colonna,
    ...
    FOREIGN KEY(nome_colonna)
        REFERENCES nome_tabella_esterna(nome_colonna_esterna)
);

```

La chiave esterna può avere delle "azioni" da eseguire in caso di cancellazione (`ON DELETE`):

- `NO ACTION`: la cancellazione di una tupla non è permessa se esistono riferimenti ad essa
- `CASCADE`: la cancellazione di una tupla comporta la cancellazione delle tuple che fanno riferimento ad essa
- `SET NULL`: la cancellazione di una tupla comporta l'impostazione a `NULL` del valore della chiave esterna delle tuple che fanno riferimento ad essa
- `SET DEFAULT`: la cancellazione di una tupla comporta l'impostazione al valore di default del valore della chiave esterna delle tuple che fanno riferimento ad essa

In caso di modifica (`ON UPDATE`) le "azioni" funzionano in modo simile.

4.3 Cancellazione di tabelle

```
DROP TABLE nome_tabella [CASCADE | RESTRICT];
```

L'opzione `CASCADE` permette di eliminare anche le tabelle che fanno riferimento alla tabella da eliminare, mentre `RESTRICT` impedisce la cancellazione se esistono tabelle che fanno riferimento alla tabella da eliminare.

4.4 Modifica di tabelle

4.4.1 Aggiunta di colonne

```
ALTER TABLE nome_tabella
    ADD nome_colonna tipo_colonna;
```

4.4.2 Modifica di colonne

```
ALTER TABLE nome_tabella
    ALTER COLUMN nome_colonna SET DATA TYPE tipo_colonna;
```

4.4.3 Cancellazione di colonne

```
ALTER TABLE nome_tabella
    DROP COLUMN nome_colonna [CASCADE | RESTRICT];
```

4.5 Interrogazioni

4.5.1 Selezione

```
SELECT [DISTINCT] lista_attributi
FROM nome_tabella
WHERE condizione;
```

La clausola `DISTINCT` permette di eliminare i duplicati.
Nella selezione ci si può riferire alle colonne mettendo prima il nome della tabella a cui appartengono seguito da un punto:

```
SELECT nome_tabella.nome_colonna
FROM nome_tabella;
```

Questo può aiutare in caso di colonne con lo stesso nome in tabelle diverse o semplicemente per una maggiore chiarezza.
Se ci si vuole riferire a tutte le colonne di una (o più) tabelle si può usare il simbolo `*`:

```
SELECT *
FROM nome_tabella;
```

Nel `WHERE` si possono usare i connettivi logici `AND`, `OR` e `NOT`.

4.5.2 Operatori di confronto

Condizioni su intervalli di valori

L'operatore `BETWEEN` permette di ritrovare le tuple che contengono valori di una colonna in un intervallo specificato:

```
        SELECT *
        FROM nome_tabella
        WHERE nome_colonna BETWEEN valore1 AND valore2;

```

`BETWEEN` è l'abbreviazione di:

```
        SELECT *
        FROM nome_tabella
        WHERE nome_colonna >= valore1 AND nome_colonna <= valore2;

```

Può essere usato con il `NOT`.

Ricerca di valori in un insieme

L'operatore `IN` permette di determinare le tuple che contengono uno tra i valori di un insieme spiecificato:

```
        SELECT *
        FROM nome_tabella
        WHERE nome_colonna IN (valore1,valore2,...);

```

E' l'abbreviazione di:

```
        SELECT *
        FROM nome_tabella
        WHERE nome_colonna = valore1 OR nome_colonna = valore2 OR ...;

```

Può essere usato con il `NOT`.

Condizioni di confronto per stringhe di caratteri

L'operatore `LIKE` permette di eseguire alcune sepmlici operazioni di *pattern matching* su colonne di tipo stringa:

- `%`: rappresenta una sequenza di zero o più caratteri
- `_`: rappresenta un singolo carattere

```
        SELECT *
        FROM nome_tabella
        WHERE nome_colonna LIKE '__d%';  -- trova la tupla che inizia
    con due caratteri qualsiasi, seguita da 'd' e poi da zero o piu'
    caratteri

```

4.5.3 Espressioni e funzioni aritmetiche

- `+`: somma
- `-`: sottrazione
- `*`: moltiplicazione
- `/`: divisione
- `ABS(`n`)`: valore assoluto di n
- `MOD(`n`,`m`)`: resto della divisione di n per m

4.5.4 Espressioni e funzioni per stringhe

- `||`: concatenazione
- `LENGTH(`s`)`: lunghezza della stringa s
- `LOWER(`s`)`: trasforma la stringa s in minuscolo
- `UPPER(`s`)`: trasforma la stringa s in maiuscolo
- `SUBSTR(`s`,`m`,`n`)`: sottostringa di s a partire dalla posizione m di lunghezza n, se n è omesso la sottostringa è fino alla fine di s
- `TRIM` $s1$ `FROM` $s2$: rimuove gli spazi bianchi da $s2$, se $s1$ è specificato rimuove i caratteri di $s1$ da $s2$

4.5.5 Espressioni e funzioni per date e tempi

- `CURRENT_DATE`: data corrente
- `CURRENT_TIME`: tempo corrente
- `CURRENT_TIMESTAMP`: data e tempo correnti
- `EXTRACT(`*campo* `FROM` *data*`)`: estrae il campo specificato da una data o un tempo (*es.* `EXTRACT(YEAR FROM CURRENT_DATE)` restituisce l'anno corrente)

4.5.6 Ordinamento dei risultati di una query

```
SELECT *
FROM nome_tabella
ORDER BY nome_colonna [ASC | DESC];

```

Ordina i risultati in base ai valori della colonna specificata in ordine crescente (`ASC`) o decrescente (`DESC`).

4.5.7 Operazione di join

- `CROSS JOIN`: prodotto cartesiano tra due tabelle

```
SELECT *
FROM tabella1
CROSS JOIN tabella2;

```

- `JOIN ON`: prodotto cartesiano tra due tabelle con una condizione

```
SELECT *
FROM tabella1
JOIN tabella2
ON tabella1.colonna = tabella2.colonna;
```

- `JOIN USING`: prodotto cartesiano tra due tabelle con una condizione su una o piu' colonne comuni

```
SELECT *
FROM tabella1
JOIN tabella2
USING (colonna_comune1, ...);
```

- `NATURAL JOIN`: prodotto cartesiano tra due tabelle con una condizione su tutte le colonne con lo stesso nome

```
SELECT *
FROM tabella1
NATURAL JOIN tabella2;
```

Outer join

Con i `JOIN` non si ha traccia delle tuple di una tabella che corrispondono a nessuna tupla dell'altra tabella. Per ovviare a questo problema si usano le `OUTER JOIN` che aggiunge al risultato anche le tuple che non hanno corrispondenza completandole con valori nulli. Il `JOIN` originario è detto `INNER JOIN`.

- `FULL`: tutte le tuple di entrambe le tabelle che non hanno corrispondenza vengono completate ed inserite nel risultato
- `LEFT`: solo le tuple della tabella a sinistra che non hanno corrispondenza vengono completate ed inserite nel risultato
- `RIGHT`: solo le tuple della tabella a destra che non hanno corrispondenza vengono completate ed inserite nel risultato

4.5.8 Funzioni di gruppo

- `MAX`: massimo valore di un insieme di valori
- `MIN`: minimo valore di un insieme di valori
- `SUM`: somma di un insieme di valori
- `AVG`: media di un insieme di valori
- `COUNT`: cardinalità di un insieme

Esistono anche le funzioni `STDEV` e `VAR` per calcolare la deviazione standard e la varianza. La funzione `COUNT` può avere tre tipi di argomenti:

- un nome di una colonna: conta il numero di valori non nulli nella colonna specificata
- un nome di colonna preceduto dal qualificatore `DISTINCT`: conta il numero di valori distinti non nulli nella colonna specificata
- `*`: conta il numero di tuple nel risultato

4.5.9 Raggruppamento

```
SELECT colonna1
FROM tabella
GROUP BY colonna1;
```

Nel `GROUP BY` si possono usare solo le colonne che compaiono nella clausola `SELECT`. Raggruppa le tuple in base ai valori della colonna specificata.
E' possibile specificare condizioni di ricerca su gruppi di tuple utilizzando la clausola `HAVING`:

```
SELECT colonna1
FROM tabella
GROUP BY colonna1
HAVING condizione;
```

4.5.10 Sotto-interrogazioni

```
SELECT *
FROM tabella
WHERE colonna IN (SELECT colonna FROM tabella2);
```

Per le sotto-interrogazioni si possono usare i quantificatori `ALL` e `ANY`:

- `ANY`: restituisce vero quando la valutazione dell'operatore di confronto è vera per almeno una delle tuple restituite dalla sotto-interrogazione, restituisce falso altrimenti o se la sotto-interrogazione non restituisce tuple
- `ALL`: restituisce vero quando la valutazione dell'operatore di confronto è vera per tutte le tuple restituite dalla sotto-interrogazione o se la sotto-interrogazione non restituisce tuple, restituisce falso altrimenti

`IN` è equivalente a `= ANY`.
`NOT IN` è equivalente a `<> ALL`.

4.5.11 Sotto-interrogazioni correlate

Una sotto-interrogazione è detta correlata se il risultato della sotto-interrogazione dipende dal valore di una colonna della query esterna. Per fare riferimento alle colonne delle tuple della query esterna si usano degli alias:

```
SELECT *
FROM tabella T
WHERE colonna IN (SELECT colonna
                  FROM tabella2
                  WHERE tabella2.colonna = T.colonna);
```

oppure

```
SELECT *
FROM tabella AS T
WHERE colonna IN (SELECT colonna
```

```
                        FROM tabella2
                        WHERE tabella2.colonna = T.colonna);

```

Operatori

Le sotto-interrogazioni correlate sono spesso usate in combinazione con gli operatori `EXISTS` e `NOT EXISTS`:

- `EXISTS`: restituisce vero se la sotto-interrogazione restituisce almeno una tupla, falso altrimenti

```
            SELECT *
            FROM tabella
            WHERE EXISTS (SELECT *
                          FROM tabella2
                          WHERE tabella2.colonna = tabella.colonna)
;

```

- `NOT EXISTS`: restituisce vero se la sotto-interrogazione non restituisce alcuna tupla, falso altrimenti

```
            SELECT *
            FROM tabella
            WHERE NOT EXISTS (SELECT *
                              FROM tabella2
                              WHERE tabella2.colonna = tabella.
colonna);

```

4.5.12 Operatori insiemistici

- `UNION`: restituisce l'unionde di due insiemi di tuple, elimina i duplicati
- `INTERSECT`: restituisce l'intersezione di due insiemi di tuple, elimina i duplicati
- `EXCEPT (o MINUS)`: restituisce la differenza tra due insiemi di tuple, elimina i duplicati

4.5.13 Inserimento

```
        INSERT INTO nome_tabella
        VALUES (valore1,valore2,...);

```