SINTASSI ALGEBRA RELAZIONALE

Proiezione: $\Pi_{attr_1, attr_2, \ldots, attr_n}(\text{RELAZIONE})$ (y unaria)

→ ordine scelto e significativo.

Associazione di un nome: $r_{CM} \leftarrow \Pi_{cod, nome}(\text{STUDENTI})$ il risultato è messo nella relazione r_{CM} che può essere utilizzata da un altra interrogaz.

Selezione: - unaria; - non altera lo schema relazionale; - restituisce un SOTTOINSIEME della relazione di partenza.

$\sigma_{\text{cond. BOOLEANA}}(r)$ Es.: $\sigma_{\text{cognome = 'ESPOSITO'}}(r)$

Operatori di confronto: $=$, $<>$, $<$, $<=$, $>$, $>=$ ⇒ DOMINI DI ORDINAMENTO

- NUMERI
- STRINGHE → "aa" < "aaa" TRUE; "ZZZ" < "aaa" TRUE
- DATE
- TIME

Si ha un problema però a confrontare un valore NULL con qualcosa, infatti:

NULL < '01-01-2000' dà valore INDEFINITO, che può dare problemi con AND, OR, NOT.

Proiezione + Selezione: $\Pi_{nome, cognome}(\sigma_{CITTÀ = \text{"NAPOLI"}}(S))$

// Il passaggio da un espressione a una altra si dice Valutazione dell'espressione //

Condizioni elementari per select:

- attributo IS [NOT] NULL; Vede se un valore è NULL o meno (LAVORO IS NOT NULL)
- attributo OP. CONFRONTO valore: (NOME = "GINO") o (ETÀ < '18')
- attributo OP. CONFRONTO attributo: (CITTÀ_N = CITTÀ_M)
- Condizioni composte con operatori logici AND, OR, NOT.

Nelle Basi di dati non si può valutare un espressione booleana in un dominio a 2 valori, ma a 3. Questo perché c'è la possibilità che un valore sia UNK e che il risultato di un espressione sia indefinito.

TABELLA DI VERITÀ:

A	B	A∧B	A∨B
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F
U	T	U	T
T	U	U	T
U	U	U	U
U	F	F	U
F	U	F	U

A	NOTA
T	F
F	T
U	U

OPERAZIONI INSIEMISTICHE

CONSIDERANDO *

A	B	C
a	a	a
a	b	c
b	b	c
c	c	c

∪ ∩ \

E	F	G
a	a	a
a	b	d
b	b	c
e	e	e
c	c	e

- Una relazione è un insieme.
- ∪, ∩, \ unione, intersezione, differenza.

(∩) è definita solo se lo schema delle relazioni è compatibile, cioè:
1. [stesso numero di attributi] e [i-esimo attributo di r1 sia dello stesso dominio (tipo)
2. dell'i-esimo attributo di r2.]

* ∩ =

?	?	?
a	a	a
b	b	c
c	c	c

TABELLA CON VALORI UGUALI.

(∪) vale stessa compatibilità * ∪ =

?	?	?
a	a	a
a	b	c
b	b	c
c	c	c
a	b	d
e	e	e

TABELLA CON TUTTI I VALORI NON RIPETUTI.

(\) stesso criterio di compatibilità

* \ = | ? | ? | ? |
|---|---|---|
| a | b | c |

VALORI DI r_1 CHE NON STANNO IN r_2.

Es.: PARTITE (DATA, STADIO, SQ1, SQ2, GOL1, GOL2)
P

1) Squadre che hanno vinto al S. Paolo.

$$Vinc1 \leftarrow \Pi_{SQ1}\left(\sigma_{\substack{STADIO = "S.PAOLO" \\ AND \\ GOL1 > GOL2}}(P)\right)$$

$$Vinc2 \leftarrow \Pi_{SQ2}\left(\sigma_{\substack{STADIO = "S.PAOLO" \\ AND \\ GOL2 > GOL1}}(P)\right)$$

RISULTATO $\leftarrow$ VINC1 $\cup$ VINC2

2) Squadre che non hanno mai vinto al S. Paolo

$$\left(\Pi_{SQ1}(P) \cup \Pi_{SQ2}(P)\right) \setminus (VINC1 \cup VINC2)$$

TUTTE LE SQUADRE — TUTTE QUELLE CHE HANNO VINTO AL S. PAOLO.

OPERAZIONI DI PRODOTTO

operazioni binarie $r_1 \times r_2$ → prodotto cartesiano

r_1

A	B
a	b
a	a

r_2

C	D	E
c	c	c
d	e	a
a	b	a

$r_1 \times r_2$

A	B	C	D	E
a	b	c	c	c
a	b	d	e	a
a	b	a	b	a
a	a	c	c	c
a	a	d	e	a
a	a	e	b	e

unione di tutti i valori in ordine

Se abbiamo r_1 con n_1 attributi
N_1 elementi

r_2 con n_2 attributi
N_2 elementi

$r_1 \times r_2 = n_1 + n_2$ attr
e
$N_1 \cdot N_2$ elem.

Per far sì che un prodotto cartesiano sia utile è necessaria la CONDIZIONE DI GIUNZIONE

$\sigma_{\text{MATR} = \text{MAT.}} (r_1 \times r_2)$

(unione tra chiave PRIMARIA e chiave esterna) ← ordina e seleziona elementi in modo corretto.

$\bowtie_c$ JOIN → CONDIZIONE DI GIUNZIONE. USO ESPLICITO

Invece di scrivere: $\sigma_{\text{MATR} = \text{MAT.}} (r_1 \times r_2) \equiv r_1 \underset{\text{MATR} = \text{MAT.}}{\bowtie} r_2$

↑ USO ESPLICITO DELLA GIUNZIONE.

GIUNZIONE NATURALE

Usa una condizione di uguaglianza implicita: gli attributi di r_1 in comune con r_2 devono avere lo stesso valore.

$r_1 \bowtie r_2$ → nel caso in cui non si verifichi la condizione implicita, esso diventa un prodotto cartesiano.

Gli attributi devono avere nome uguale e i valori al loro interno pure.

MATR	NOME	COGN
100	a	b
200	e	f

$\bowtie$

MATR	NOME	COGN
100	c	d
300	h	i

$= \varnothing$

OPERAZIONE DI RINOMINA

Serve a modificare il nome degli attributi di una relazione.

SINTASSI: $P(S)$ STUDENTE(MAT, NOME, COGNOME)

relazione originale: STUDENTE(MATRICOLA, NOME, ...

2 → $S \bowtie E$ il natural join è possibile poiché adesso MAT^{S} e MAT^{E} sono uguali.

2 → $P(S)$ MATRICOLA ← MAT più chiaro e sintetico.

in join diretto: $P(S) \bowtie E$ MATRICOLA ← MAT

ES.:

RESIDENZA(CF, DATA1, DATAF, CITTAR, VIA) → R

PERSONA(CF, NOME, COGNOME, DATAN, DATAM) → P

PARTENZA(CF, CFMADRE, CF PADRE) → PART

1) CF, nome e cognome di coloro che hanno fatto un cambio residenza, cambiando città.

$$CAMBIO \leftarrow \Pi_{CF}\left(\sigma_{CITTAR \neq CITTAR1}\left(R \bowtie_{CF = CF1} P(R)\right)\right)$$

P(R) RESIDENZA(CF1, DATA1L, DATAFL, CITTAR1, VIA1)

$$\Pi_{CF, NOME, COGNOME}(CAMBIO \bowtie P)$$

" Per poter controllare prodotti tra tabelle uguali, la seconda tabella (= alla prima) deve essere rinominata in tutti i suoi attributi "

N.B. = Il risultato di un JOIN è dato dalla corrispondenza di 1 riga di sinistra con tutte quelle di destra, e non mano che si trovano corrispondenze si continua la relazione nuova. Ciò vuol dire che: se un valore che ci serve compare più volte nelle tabelle è uguale, esso viene messo nella nuova relazione, e non può essere eliminato.

OPERAZIONI DI PRODOTTO -CONTINUO

OUTER JOIN (GIUNZIONE ESTERNA)

"Questa serve per tenere traccia di quei elementi che col JOIN andrebbero persi"

Esistono 3 varianti: - ⟗ FULL OUTER JOIN → conserva i valori non matchati di entrambe le relazioni.
⟕ LEFT OUTER JOIN → conserva solo quelli di r_1
⟖ RIGHT OUTER JOIN → conserva solo quelli di r_2

ES.:

STUDENTI

MATR	NOME	COGNOME
100	CIRO	ESPOSITO
200	MARIO	ROSSI
300	UGO	VERDI

ESAMI

MAT	CORSO	VOTO
100	BD I	25
200	BD I	24
100	BD II	30

S ⟗ E (MATR = MAT)

MATR	NOME	COGNOME	MAT	CORSO	VOTO
100	CIRO	ESPOSITO	100	BD I	25
100	CIRO	ESPOSITO	100	BD II	30
200	MARIO	ROSSI	200	BD I	24
300	UGO	VERDI	* NULL	NULL	NULL

Con un normale JOIN l'ultima riga sarebbe andata perduta, così invece no.

"N.B.: se si recupera un valore da una relazione che non è presente nell'altra, i valori che sono nella parte che non matcha sono settati a NULL." *

OPERAZIONI DI CONTEGGIO

Per poter utilizzare operazioni di conteggio bisogna:

1) CRITERIO DI RAGGRUPPAMENTO creare partizioni
2) OP. di CONTEGGIO sulle singole partizioni.

Un criterio di raggruppamento è dato da una LISTA (POSSIBILMENTE VUOTA) di attributi prendendo come esempio un attributo (MATR) ⇒ un gruppo è dato da "tutte le righe che hanno lo stesso valore di MATR."

N.B.: Se la lista è vuota il gruppo coincide con la relazione.

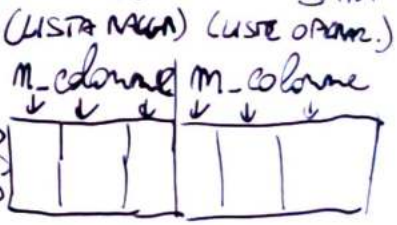
LE OPERAZIONI SONO:

COUNT (ATTRIBUTO) → conta gli elementi del GRUPPO con valore NOT NULL nell'attributo.
SUM (ATTRIBUTO) → somma i valori dell'attributo su tutti gli elementi del GRUPPO.
MIN (ATTRIBUTO), MAX (ATTRIBUTO) → Recupera valori MIN/MAX sull'attributo tra tutti gli elementi del GRUPPO.
AVG (ATTRIBUTO) → Media

SINTASSI OPERAZIONE

(LISTA RAGGRUPPAMENTO) ℰ LISTE OP. CONTEGGIO (relazione)

RISULTATO SARÀ (LISTA RAGG) (LISTE OPERAZ.)



Es.: Abbiamo: ESAMI (mat, corso, voto, CFU, data, lode)

Per ogni studente il n° esami, n° crediti per ogni anno

$$P(\text{mat}, \text{YEAR}(\text{data}))\ \mathcal{E}\ \text{COUNT}(\text{corso}), \text{SUM}(\text{CFU})\ (\text{ESAMI})$$

mat, anno, n_esami, crediti

⇓ RISULTATO

MATR	ANNO	N_ESAMI	CREDITI

ESERCIZIO ALGEBRA RELAZIONALE

STUDENTI (MATR, NOME, COGNOME, DATAN,)

ESAMI (MAT, DATA, VOTO, LODE, CORSO,)

Trovare nome, cognome e matricola degli studenti che nell'anno 2019 hanno la media più alta.

$TABM \leftarrow \rho_{matr, nome, cognome, media}\, \mathcal{E}_{AVG(VOTO)}\,(S \bowtie \sigma_{YEAR(DATA)='2019'}(E))$

$TMAX \leftarrow \rho_{VALMAX}\, \mathcal{E}_{MAX(MEDIA)}(TABM)$

RISULTATO

MATR	NOME	COGNOME	MEDIA

VALMAX

$\Pi_{nome, cognome, matricola}\,(\sigma_{MEDIA = VALMAX}\,(TABM \bowtie TMAX))$ OPPURE $\Pi_{nome, cognome, matricola}\,(TABM \bowtie_{MEDIA = VALMAX} TMAX)$

PERSONA(CF, NOME, COGNOME, DATAN, DATAM, CITTAN)

RESIDENZE (CF, VIA, CITTA, DATAI, DATAF)

GENITORI (CF, CFPADRE, CFMADRE)

Trovare le persone che hanno avuto residenza sempre nella stessa città.

CONTEGGIO

$TABC \leftarrow \rho_{CF, N\_CITTA}\, \mathcal{E}_{COUNT(CITTA)}\,(\Pi_{CF, CITTA}(R))$ → città diverse (elimina i duplicati)

$RIS \leftarrow \sigma_{N\_CITTA = 1}(TABC)$

NO CONTEGGIO : trovare quindi persone con + residenze in città diverse.

$NO \leftarrow \Pi_{CF}\,(\sigma_{CITTA <> CITTA1}\,(R \bowtie \rho_{(CF, VIA1, CITTA1, DATAI1, DATAF1)}(R)))$ CF uguale per il natural join

$SI \leftarrow \Pi_{CF}(P) \setminus NO$

N.B.: in NO ci sono solo CF => si proiettano solo i CF di PERSONE per poter fare la differenza.

SINTASSI PLPGSQL

PROCEDURE

```
CREATE [OR REPLACE] PROCEDURE nome_procedura ([nome_parametro IN/OUT tipo])
AS
$$
DECLARE
        Dichiarazioni
BEGIN
        Istruzioni
END;
$$
LANGUAGE plpgsql;
```

```
CALL nome_procedura(valore);
```

FUNZIONI

```
CREATE [OR REPLACE] FUNCTION nome_funcione([nome_parametro IN/OUT tipo]) RETURN tipo
AS
$$
DECLARE
        Dichiarazioni
BEGIN
        Istruzioni
        RETURN valore;
END;
$$
LANGUAGE plpgsql;
```

```
SELECT nome_funzione(valore);
```

ESEGUIRE UN COMANDO CHE RESTITUISCE UNA RIGA

```
CREATE [OR REPLACE] FUNCTION nome_funcione([nome_parametro IN/OUT tipo]) RETURN tipo
AS
$$
DECLARE
        var1 tab_name%ROWTYPE;
BEGIN
        SELECT col1 INTO var1
        FROM tab_name
        WHERE ID=valore;                    --La query deve restituire un solo valore
        RETURN var1;
END;
$$
LANGUAGE plpgsql;
```

CURSORE ESPLICITO PREDICHIARATO

```
CREATE [OR REPLACE] FUNCTION nome_funcione([nome_parametro IN/OUT tipo]) RETURN tipo
AS
$$
DECLARE
        cur CURSOR FOR SELECT * FROM tab_name WHERE condition;
        riga tab_name%ROWTYPE;
BEGIN
        OPEN cur;
        LOOP
                FETCH cur INTO riga;
                EXIT WHEN NOT FOUND;
                Istruzioni
        END LOOP;

        RETURN valore;
END;
$$
LANGUAGE plpgsql;
```

CURSORE ESPLICITO POSTDICHIARATO

```
CREATE [OR REPLACE] FUNCTION nome_funcione([nome_parametro IN/OUT tipo]) RETURN tipo
AS
$$
DECLARE
        cur REFCURSOR;
        riga tab_name%ROWTYPE;
BEGIN
        OPEN cur FOR SELECT * FROM tab_name WHERE condition;
        LOOP
                FETCH cur INTO riga;
                EXIT WHEN NOT FOUND;
                Istruzioni
        END LOOP;

        RETURN valore;
END;
$$
LANGUAGE plpgsql;
```

CURSORE IMPLICITO NEL FOR

```
CREATE [OR REPLACE] FUNCTION nome_funcione([nome_parametro IN/OUT tipo]) RETURN tipo
AS
$$
DECLARE
        riga tab_name%ROWTYPE;
BEGIN
        FOR riga IN query LOOP
                Istruzioni
        END LOOP;
        RETURN valore;
END;
$$
LANGUAGE plpgsql;
```

CURSORE DINAMICO

```
CREATE [OR REPLACE] FUNCTION nome_funcione([nome_tabella IN pg_stat_all_tables.relname])
RETURN tipo
AS
$$
DECLARE
        cur REFCURSOR;
        comando VARCHAR(100);
        riga nome_tabella%ROWTYPE;
BEGIN
        comando := 'SELECT * FROM ' || nome_tabella || ' WHERE condition;";[1]
        OPEN cur FOR EXECUTE comando;
        LOOP
                FETCH cur INTO riga;
                EXIT WHEN NOT FOUND;
                Istruzioni
        END LOOP;

        RETURN valore;
END;
$$
LANGUAGE plpgsql;
```

TRIGGER

```
CREATE [OR REPLACE] TRIGGER nome_trigger
BEFORE/AFTER/INSTEAD OF INSERT/UPDATE/DELETE ON nome_tabella
FOR EARCH ROW
WHEN condizione                 --Non sono possibili query      --Si ha accesso alle variabili NEW/OLD
BEGIN[2]
        Istruzioni      --Si ha accesso alle variabili NEW/OLD
END;
```

[1] In alternativa comando:= format("*SELECT * FROM %I WHERE condition;", nometabella);*

[2] In alternativa *EXECUTE FUNCTION nome_funzione(parametri);*

SINTASSI SQL

ASSERTION

```
CREATE ASSERTION nomeasserzione
CHECK NOT EXISTS ( SELECT...
                   FROM...
                   WHERE... )
```

CREAZIONE TABELLE

```
CREATE TABLE nome_tabella (
  nome_attributo tipo (dim.) NOT NULL / PRIMARY KEY, UNIQUE
  CONSTRAINT nomevincolo CHECK (attributo <> 10)
  CONSTRAINT nomevincolo FOREIGN KEY (nome_attributo) REFERENCES (nometabella.nome attributo)
);
```

CONSTRAINT DOPO CREAZIONE TABELLE

```
ALTER TABLE nometabella
  ADD CONSTRAINT nome_vincolo CHECK / PRIMARY KEY / FOREIGN KEY (nome_attributo).
```

INSERT / UPDATE / DELETE

```
INSERT INTO nome_tabella (attributo1, attributo2, ...)
VALUES (val1, val2, ...);
```

```
UPDATE nome_tabella SET nome_attributo = valore
WHERE condizione.
```

```
DELETE FROM nometabella
WHERE condizione;
```

SELECT

```
SELECT CF, SUM(VOTO)
FROM tab1 AS T1 INNER JOIN tab2 AS T2 ON T1.attr = T2.attr
WHERE condizione
GROUP BY CF
HAVING SUM(VOTO) > 25
ORDER BY nomeattributo DESC;
```

N.B.:
HAVING si usa per le aggregazioni, poiché a fare (SUM(VOTO > 25)) questo non si potrà fare nella WHERE, ma solo nell' HAVING.

DESC = decrescente
ASC/nulla = crescente.

SINTASSI VARIE SQL

SUBSTR (STRINGA, POSI, LUNGHEZZA)

INSTR (STRINGA, 'CARATTERE DA TROVARE', [POSI, RICORRENZA])

TRIGGER

CREATE TRIGGER nometrigger
{AFTER/BEFORE} {ISTEAD OF} → SU VISTE
{INSERT / DELETE / UPDATE [OF colonna]}
ON tabella
FOR EACH ROW
[WHEN (CONDIZIONE)]
BEGIN
  DECLARE
  ...
  BEGIN
  :
  END
END

nella procedura di un trigger è possibile utilizzare NEW e OLD che si riferiscono ai dati inseriti o cancellati.

SEQUENCE

CREATE SEQUENCE <NOME>
START WITH <numero partenza>
INCREMENT BY <numero_incremento>
MIN VALUE <valore_minimo>
MAX VALUE <valore_massimo> → al raggiungimento del massimo, inizia daccapo.

VIEW

CREATE VIEW <nome view> AS
  (SELECT -
   FROM -
   WHERE -)

Per le view si utilizzano i
TRIGGER ISTEAD OF INSERT/DELETE/UPDATE ...
(come AFTER
o
BEGIN)

SINTASSI SQL

PROCEDURE

STORED PROCEDURE (procedure memorizzate)

```
CREATE PROCEDURE nome_procedura (eventuali_parametri TIPO, altri tipi) AS
DECLARE
  ⋮
BEGIN
  ⋮
END
```

FUNCTION

```
CREATE FUNCTION nome_funzione (var TIPO) RETURNS tipo_ritorno
DECLARE ──→ { VARIABILI → dopo es. 1
              COSTANTI
              CURSORI → dopo es. 2
              NOMI ECCEZIONI
BEGIN
  ⋮
EXCEPTION
  ⋮
END
```

SELECT INTO

```
SELECT exp1, ..., expn
    INTO var1, .., varn
FROM tabella
WHERE condizione.
```

Questa soluzione vuole che il risultato della SELECT restituisca AL PIÙ UNA RIGA, quindi massimo 1 RIGA.

CURSORE

DECLARE nome_cursore [SCROLL] [NO SCROLL] CURSOR FOR SELECT ... FROM ... WHERE ... [FOR <READ ONLY | UPDATE [OF ATTRIBUTO...]>]

(default) NO SCROLL legge in modo sequenziale i valori della SELECT.

SCROLL ordine di scansione alterabile.

DECLARE ← associa un cursore alla tabella risultato della SELECT.

OPEN (nome_cursore) ← esegue la SELECT mettendo il risultato nel buffer.

CLOSE (nome_cursore) ← disalloca il buffer.

FETCH [POSIZIONE FROM] (nome_cursore) INTO var1, ..., varn ← trasferisce i dati della RIGA CORRENTE nelle variabili.

Il buffer gestito ha la posizione fuori dal buffer, così che al primo FETCH di default va al prossimo, cioè la prima riga del buffer.

[POSIZIONE] ha diversi valori disponibili: NEXT, che è automatico, di fatti non è necessario scrivere FETCH NEXT, ma è possibile scrivere solo FETCH nome_cursore, per andare avanti.

SCROLL
- PRIOR, va al precedente del CORRENTE;
- FIRST, va al primo del BUFFER;
- LAST, va all'ultimo del BUFFER;
- ABSOLUTE (espressione INT), va al posto indicato dall'intero;
- RELATIVE (espressione INT), va al CORRENTE + INT indicato dall'intero.

SINTASSI CURSOR NO SCROLL per FETCH:

// FETCH (NEXT implicito) <nome_cursore> INTO <lista variabili> //

// V.B.: un cursore può essere aperto e chiuso più volte. Il senso è quello che, essendo solitamente funzioni parametriche, cambiando valore del parametro, si può avere un risultato diverso dall'interrogazione. //

Esempio FUNCTION

Si vuole nome e cognome di uno studente, dato in input una matricola

1) VARIABILI

```
CREATE FUNCTION nome_cogn ( matr STUDENTI.MATRICOLA%TYPE)
                RETURNS VARCHAR(40)

info VARCHAR(40)
BEGIN
SELECT S.NOME || ' - ' || S.COGNOME INTO info
FROM STUDENTI AS S
WHERE S.MATRICOLA = MATR;

RETURN info;
END
```

→ formattazione output.

2) CURSORI:

Dato una matricola, restituire l'insieme di esami sostenuti da tale studente, ovviamente!

```
CREATE FUNCTION esami (matr STUDENTI.MATRICOLA%TYPE)
                RETURNS VARCHAR(1000)
risultato VARCHAR(1000) := ' ';
CURSOR traeEsami AS
    (SELECT E.CORSO || ';' || TOchar(E.VOTO) AS esame*
     FROM ESAMI AS E
     WHERE E.MATRICOLA = MATR);
ex traeEsame%ROWTYPE;

BEGIN
OPEN (traeEsami);
WHILE traeEsami%FOUND
    LOOP
    FETCH traeEsami INTO ex;
    risultato := risultato || ex.esame* || ';';
    END LOOP
CLOSE (traeEsami);
RETURN risultato;
END
```

DEFINIZIONE CURSORI

(trasforma un INT in carattere da poter concatenare.)

Lancio l'interrogazione

finché c'è qualcosa

ex è un RECORD, da cui possiamo estrarre la colonna esame*, del tipo del cursore.

Var <nome_cursore>%ROWTYPE

Molto utile per conservare in variabili il tipo di valore che ne esce dall'interrogazione.

Serve questo tipo di variabile per fare un FETCH INTO!!