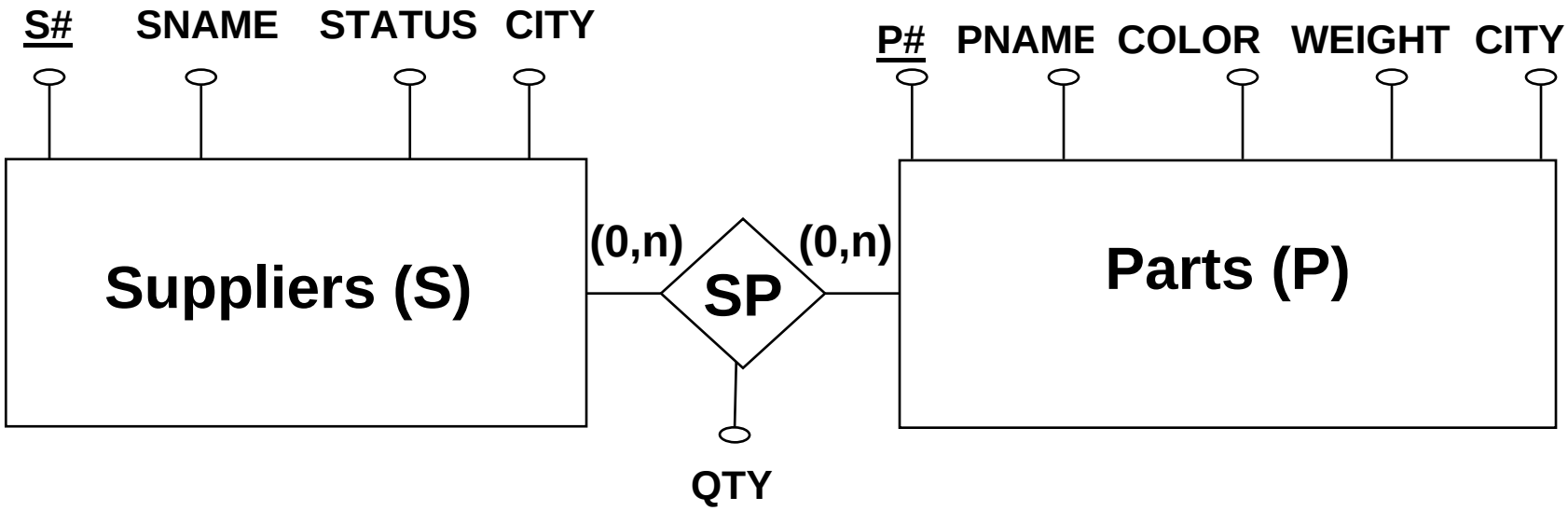
Basi di Dati Relazionali - Parte I

Basi di dati relazionali

- Introduzione
- Aspetti relativi al DBMS Oracle basato sul modello relazionale
- Il linguaggio SQL:
  - definizione dei dati
  - manipolazione dei dati

DB prodotti-fornitori



DB prodotti-fornitori

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

DB prodotti-fornitori

- **modello relazionale:**
  - **tabella prodotti**
  - **tabella fornitori**
  - **tabella forniture, che mette in relazione prodotti e fornitori che li forniscono**
- chiave primaria: **identificatore del record**
  - **prodotti: P#**
  - **fornitori: S#**
  - **forniture: (S#,P#)**

DB prodotti-fornitori

Caratteristiche delle tabelle:

- ***valori atomici*** **(non più di uno per riga/colonna)**
- ***dati espliciti*** **(no puntatori o link)**
  - **La relazione tra la riga S1 e la riga P1 è espressa dalla riga P1S1 nella tabella forniture (creata per rappresentare la relazione stessa).**
  - **I puntatori e i link non sono visibili all'utente (ma possono essere usati nel livello interno).**

SQL: Structured Query Language

- **SQL è uno standard promulgato da ANSI-ISO.**
  Ne sono state publicate varie integrazioni:
  - ***SQL-89*** **(nel 1989),**
  - ***SQL-93*** **(nel 1993, e viene anche chiamato SQL-2),**
  - ***SQL-99*** **(nel 1999, chiamato SQL-3).**
- **E' uno standard molto complesso e il linguaggio è molto sofisticato. Non tutti i prodotti sono compatibili al 100% con lo standard, per cui sono stati definiti tre livelli di compatibilità:**
  - **Entry-level (o Core SQL)**
  - **Medium**
  - **Full**

SQL

- **E' il linguaggio per definire la struttura di una base di dati relazionale e modificarne i dati.**
  Consiste di due componenti:
  - **DDL (Data Definition language): definisce la struttura della base di dati**
  - **DML (Data Manipulation Language): per manipolare e ricercare i dati**
- **Le operazioni possono essere *interattive* o c*ompilate.***
  Se compilate, un linguaggio ospite (*host)* contiene le istruzioni SQL che si distinguono dalle istruzioni del linguaggio ospite per mezzo di opportuni "artifici sintattici".

Esempi di interrogazioni

- **Interattiva (SQL-PLUS o SQL-WorkSheet):**

```
SELECT   CITY
FROM  S
      WHERE  S# = 'S4';
```

Risultato :

CITY
London

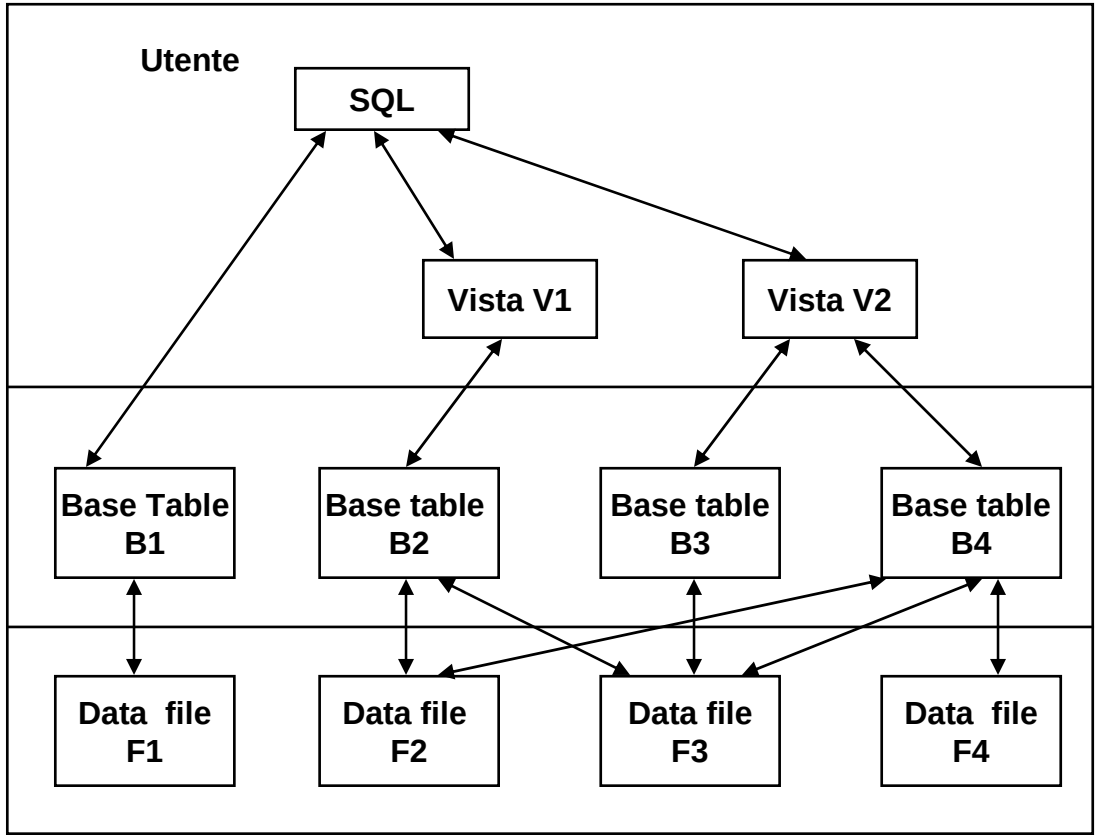
- **Embedded in C (potrebbe essere COBOL, FORTRAN, o Assembler):**

```
EXEC SQL
   SELECT CITY
   INTO  :XCIT
   FROM  S
   WHERE  S# = 'S4';
```

Risultato :

XCIT
London

Percezione del DB da parte dell'utente



Elementi costitutivi del DB

- **Utenti:**
  - **SQL**
  - **Viste delle applicazioni (VIEW): indicano tabelle non effettivamente presenti nel DB, ma derivate da una o più *base tables***
- **Tabelle (BASE TABLE): corrispondono agli elementi effettivamente esistenti nel DB.**
- **Files (livello interno). La stessa base table può essere allocata in diversi data file.**
  - **tutti i DB relazionali supportano B-trees**
  - **altri operano anche con la tecnica *hash* o con *bit-map***

Manipolazione dei dati: esempio

- ***SELECT S#***
  FROM SP
  WHERE P# = 'P2';

Risultato :

S#
S1
S2
S3
S4

- ***UPDATE S***
  SET STATUS = 2 * STATUS
  WHERE CITY = 'London';

Risultato : Stato raddoppiato per S1 e S4

Considerazioni generali

- **Il linguaggio SQL è un linguaggio a *livello di set:* operazioni e risultati coinvolgono insiemi.**
- **Il linguaggio SQL è *dichiarativo:***
  - **si pone ad un livello di astrazione superiore rispetto ai linguaggi di programmazione di terza generazione**
  - **dice *cosa fare* non *come fare***
  - **SELECT non indica le modalità di accesso al file, ma le caratteristiche dei dati da selezionare**

Tipi di dato

- **INTEGER (intero di cifre decimali: 9 al minimo, 38 al massimo)**
- **SMALLINT (intero di decimali: 4 al minimo, 38 al massimo)**
- **DECIMAL (p,q)**
- **FLOAT(b)**
- **CHAR(n)**
- **VARCHAR(n)**
- **LOGICAL**
- **BIT**
- **MONEY**
- **DATE**
- **TIME**
- **TIMESTAMPS**

Precisione

Scala (di default 0)

DECIMAL o NUMBER per numeri a virgola fissa

FLOAT per numeri a virgola mobile, con precisione binaria b (default 126 binaria, o 38 decimale).

- DOUBLE PRECISION per numeri a virgola mobile, con precisione binaria 126.
- REAL per numero a virgola mobile, con precisione binaria 63, o 18 decimale.

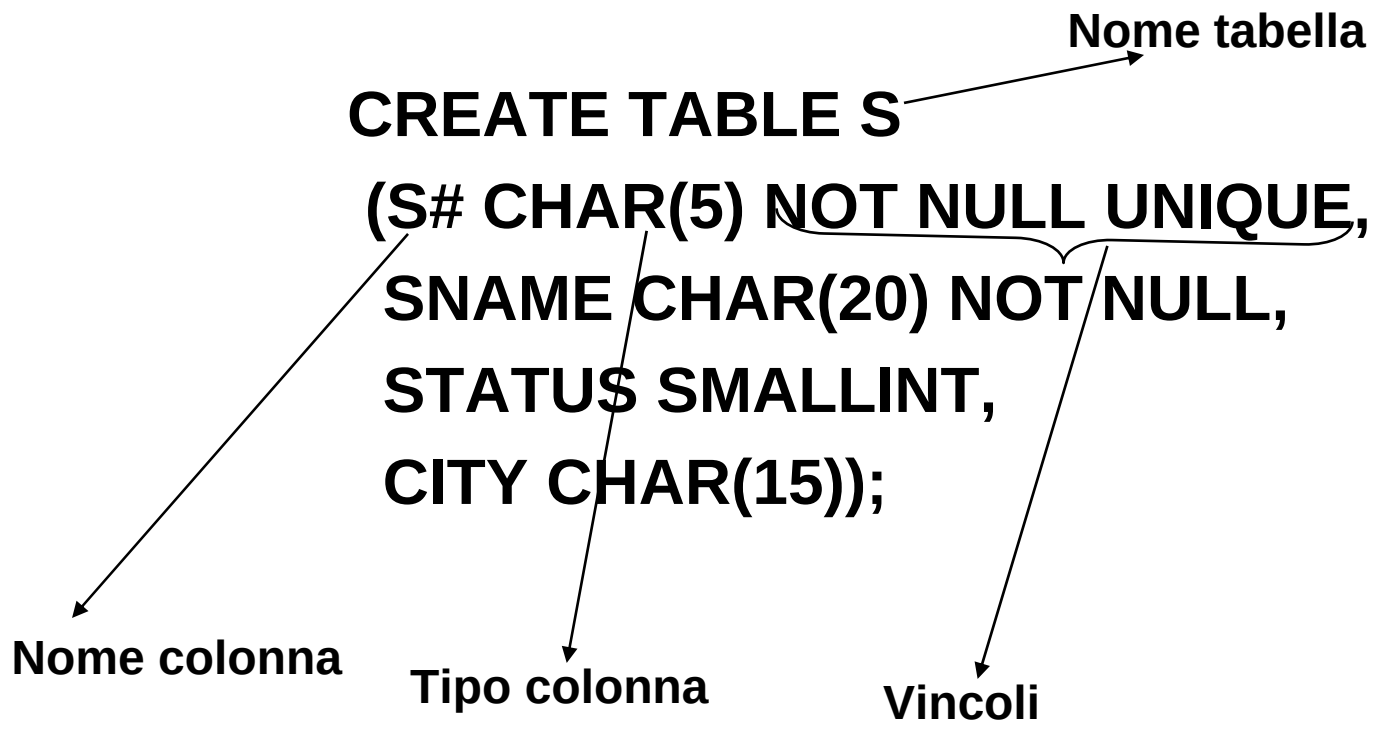
DATE e TIME usano vari formati che includono anno, mese, giorno, ora, minuti, secondi, e fuso orario

Data Definition Language

- **CREATE TABLE: creazione di una tabella**
- **CREATE INDEX: creazione di un indice**
- **ALTER TABLE: modifica della struttura di una tabella**
- **DROP TABLE: cancellazione di una tabella**
- **DROP INDEX: cancellazione di un indice**

CREATE TABLE

Esempio: **creazione della tabella fornitori.**



Il meta-linguaggio di descrizione

- **Si usa una *grammatica* per descrivere la sintassi corretta di un linguaggio (l'SQL, nel nostro caso)**
- **Una grammatica usa un *meta-linguaggio.* I simboli usati nel meta-linguaggio sono:**

  ::= indica che un termine generico a sinistra si espande in una espressione più in dettaglio, a destra

  [ ] indica che ciò che è compreso nella parentesi è opzionale

  | indica che un termine generico a sinistra si espande in due o più termini alternativi

  { } indica che ciò che è compreso nella parentesi può essere ripetuto da 0 a molte volte

CREATE TABLE

Istruzione-di-creazione-tabella ::= CREATE TABLE nome-tabella-base (definizione-colonna {,definizione-colonna} {vincoli-su-più-colonne})

definizione-colonna::= nome-colonna tipo-dato [default] {vincoli-su-colonna}

***default ::=* DEFAULT espressione**

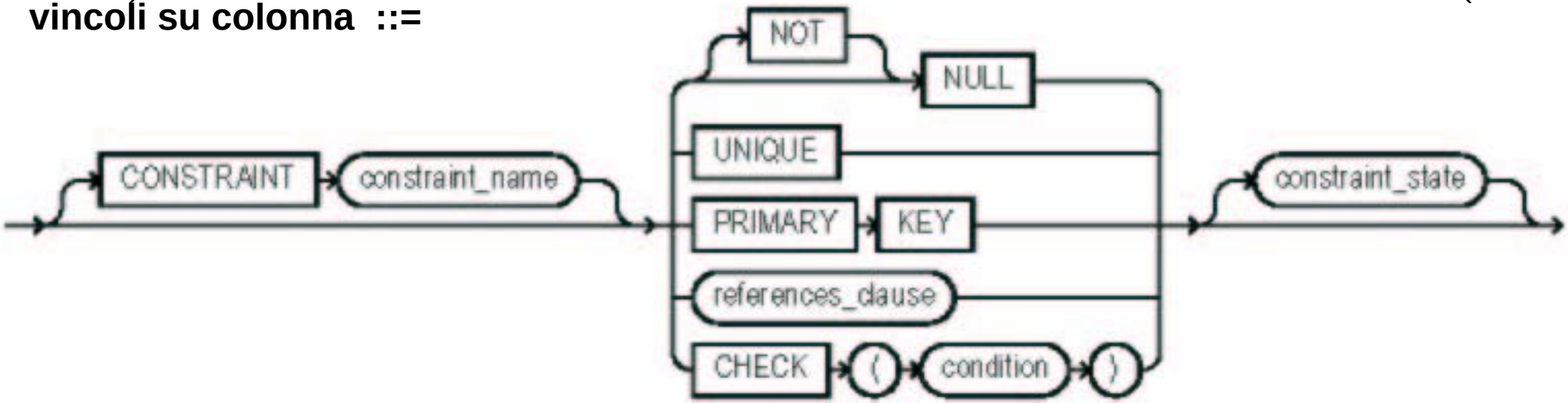
***vincoli-su-colonna* ::= [CONSTRAINT nome-vincolo] { [NOT] NULL | UNIQUE | PRIMARY KEY | clausola REFERENCES per FOREIGN KEY | altri vincoli}**

***vincoli-su-più-colonne ::=* [CONSTRAINT nome-vincolo] {PRIMARY KEY(*elenco-colonne*) | FOREIGN KEY (elenco-colonne) REFERENCES nome-tabella (elenco-colonne) | altri-vincoli}**

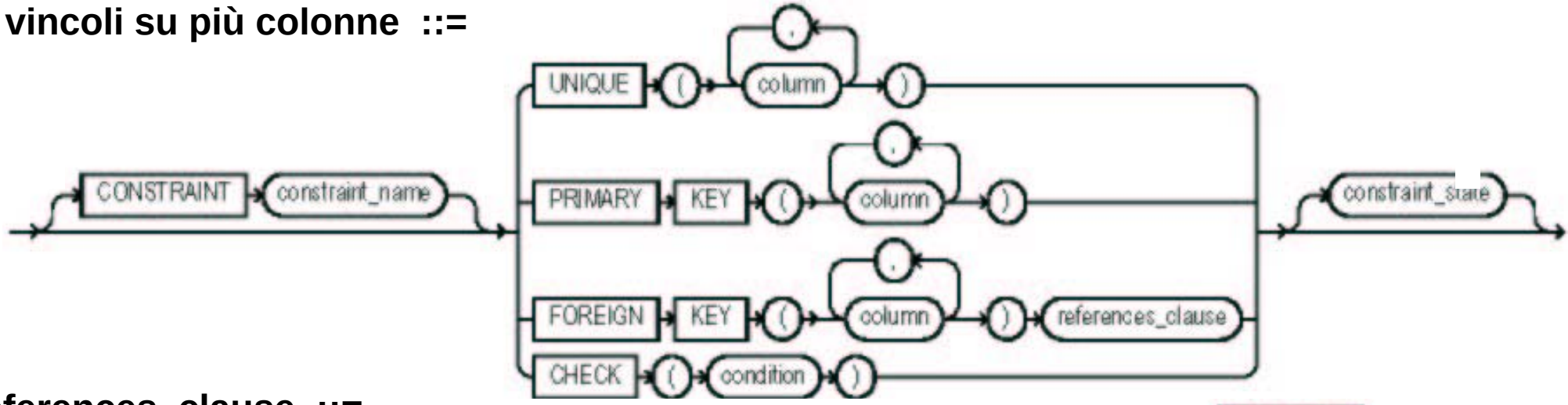
altri-vincoli* ::= *CHECK(asserzione)

***Nota:* asserzione è un *predicato-booleano* e verrà spiegato in dettaglio nel seguito**

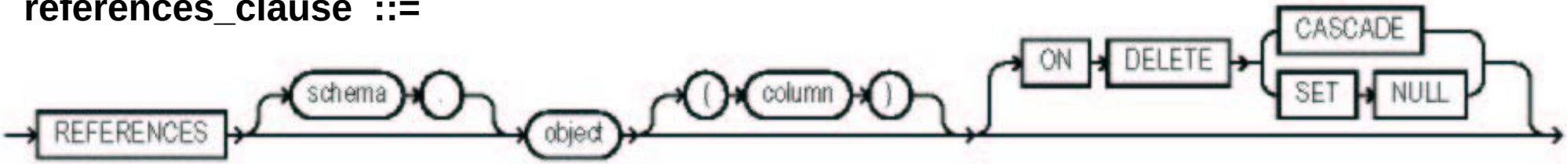
vincoli su colonna ::=



vincoli su più colonne ::=



references_clause ::=



Definizione del DB parti-fornitori

```
CREATE TABLE S
      ( S#       CHAR(5) PRIMARY KEY,
        SNAME    CHAR(20)  NOT NULL,
                 STATUS    SMALLINT,
                 CITY   CHAR(15) NOT NULL );
CREATE TABLE P
        (        P#     CHAR(6) PRIMARY KEY,
                 PNAME  CHAR(20) NOT NULL,
                 COLOR  CHAR(6),
                 WEIGHT SMALLINT,
                 CITY   CHAR(15) NOT NULL );
CREATE TABLE SP
        (        S#     CHAR(5)      NOT NULL,
                 P#     CHAR(6)      NOT NULL,
                 QTY    INTEGER      NOT NULL CHECK(QTY>=0)
                 PRIMARY KEY (S#,P#),
                 FOREIGN KEY (S#) REFERENCES S (S#),
                 FOREIGN KEY (P#) REFERENCES P (P#));
```

ALTER TABLE

Istruz-alterazione-definizione-tabella::=
***ALTER TABLE base-table-name {*ADD *definizione-colonna* | DROP nome-colonna};**

definizione-colonna::= nome-colonna tipo-dato [default] {vincoli-su-colonna}

- ***Esempio:*** **aggiungere la colonna DISCOUNT alla tabella fornitori.**

ALTER TABLE S
ADD DISCOUNT SMALLINT;

- ***Esempio:*** **togliere la colonna STATUS alla tabella fornitori.**

ALTER TABLE S
DROP STATUS;

DROP TABLE

DROP TABLE base-table-name;

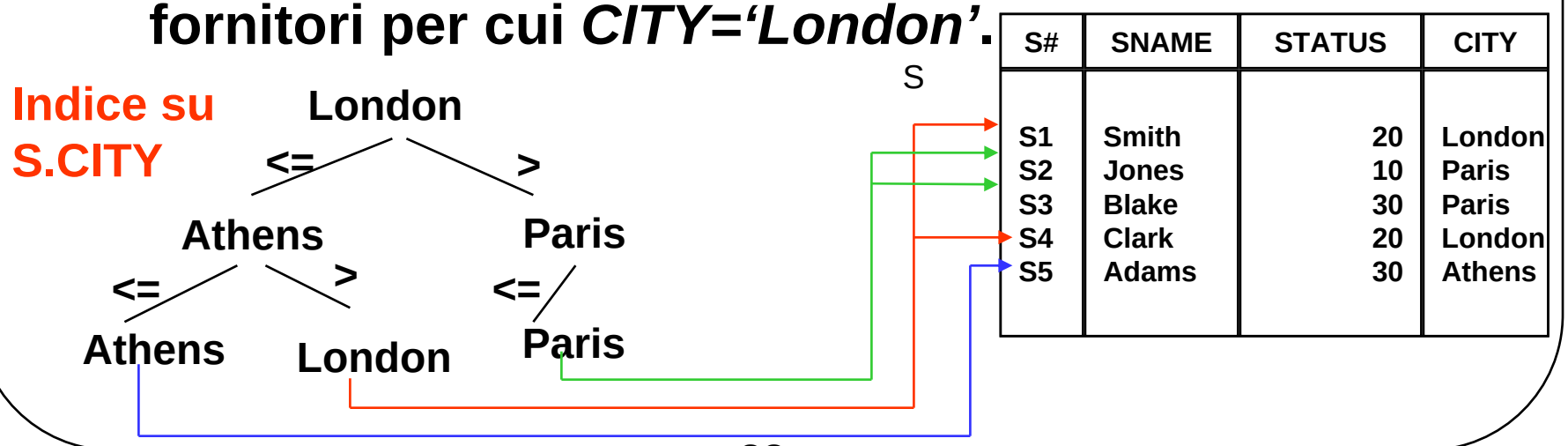
- **sono eliminati anche tutti gli indici e le viste definite su *base-table-name.***

Esempio: **cancellare la tabella fornitori.**

DROP TABLE S;

INDICI

- **Un indice è una struttura dati definita a livello logico-fisico.**
- **Solitamente si tratta di un albero di ricerca bilanciato, detto B-tree.**
- **Permette un accesso ad una tabella facilitato quando si ricercano i dati che soddisfano una certa proprietà. Ad esempio: si ricercano i fornitori per cui *CITY='London'*.**



S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

INDICI

Creazione di un indice:

CREATE [UNIQUE] INDEX *nome-indice*

ON *nome-tabella-base (nome-col {,nome-col});*

- **l'opzione UNIQUE specifica che solo un record nel file può assumere un dato valore**

Esempi:

CREATE UNIQUE INDEX XS ON S (S#);

CREATE UNIQUE INDEX XP ON P (P#);

CREATE UNIQUE INDEX XSP ON SP (S#,P#);

La violazione dell'unicità impedisce l'operazione.

INDICI (cont.)

Esempio:

CREATE INDEX XSC ON S (CITY);

UNIQUE non è specificato, poichè più record possono fare riferimento alla stessa città.

Cancellazione di un indice:

DROP INDEX index-name;

Manipolazione dei dati

Esempio: **Ricavare i codici e lo stato dei fornitori nella città di Parigi.**

```
SELECT S#, STATUS
FROM S
WHERE CITY='Paris';
```

Risultato :

S#	STATUS
S2	10
S3	30

Il risultato è una tabella.

E' equivalente ad una operazione di Selezione e Proiezione in algebra:

$$\pi_{S\#,STATUS}\,\sigma_{CITY='Paris'}\,S$$

Manipolazione dei dati

Formato generale dell'interrogazione:

SELECT [DISTINCT] col-espr-agg {,col-espr-agg}
FROM tabella {,tabella}
[WHERE predicato-booleano-su-riga]
[GROUP BY col {,col}
[HAVING predicato-booleano-su-gruppo]]
[ORDER BY col [tipo ordine] {,col [tipo ordine]}]

col-espr-agg ::= colonna | espressione | funzione-aggregata

tipo ordine ::= ASC | DESC

Nota: le funzioni aggregate le vedremo in seguito

SELECT

Esempio

SELECT P#
FROM SP;

Risultato :

P#
P1
P2
P3
P4
P5
P6
P1
P2
P2
P2
P4
P5

I duplicati non vengono eliminati.

SELECT con DISTINCT

Eliminazione dei duplicati:

SELECT DISTINCT P#

FROM SP;

Risultato :

P#
P1
P2
P3
P4
P5
P6

SELECT con espressioni

Ricerca con espressioni:

SELECT P#, 'Weight in grams=', WEIGHT * 454
FROM P;

Risultato :

P#		
P1	Weight in grams =	5448
P2	Weight in grams =	7718
P3	Weight in grams =	7718
P4	Weight in grams =	6356
P5	Weight in grams =	5448
P6	Weight in grams =	8626

SELECT

Estrazione di tutte le informazioni:

```
SELECT *
FROM S;
```

Identificatori completamente specificati:

```
SELECT S.S#, S.NAME, S.STATUS, S.CITY
FROM S;
```

SELECT (cont.)

Ricerca qualificata:

Individuare i numeri dei fornitori di Parigi con stato > 20

```
SELECT S#
FROM S
WHERE CITY = 'Paris'  AND STATUS > 20 ;
```

Risultato :

S#
S3

Ricerca con Ordinamento

Ricavare i codici e lo stato dei fornitori di Parigi in ordine decrescente rispetto allo stato:

SELECT S#, STATUS FROM S

WHERE CITY='Paris'

ORDER BY STATUS DESC;

Risultato :

S#	STATUS
S3	30
S2	10

nome-colonna [ordine] {,nome-colonna [ordine]}

ASC
crescente

DESC
decrescente

Ricerca con Ordinamento (cont.)

La colonna dell'ordinamento può essere identificata dal numero d'ordine dell'argomento.

```
SELECT P.P#, 'Weight in grams=‘ , P.WEIGHT * 454
FROM P
ORDER BY 3,P#;
```

Risultato :

P#		
P1	Weight in grams =	5448
P5	Weight in grams =	5448
P4	Weight in grams =	6356
P2	Weight in grams =	7718
P3	Weight in grams =	7718
P6	Weight in grams =	8626

JOIN

Interrogazione che ricerca un insieme di dati in due o più tabelle.

Equijoin semplice:

Ricavare tutte le combinazioni di informazioni riguardanti fornitori e parti in modo che entrambi appartengano alla stessa città.

```
SELECT S.*,P.*
FROM S,P
WHERE S.CITY=P.CITY;
```

Nomi delle colonne completamente espressi per evitare ambiguità.

JOIN

Esecuzione del prodotto cartesiano delle tabelle elencate: questo dà origine ad una tabella risultante composta da tutte le possibili righe r dove r è la concatenazione di una riga della prima tabella, una della seconda ... una della ennesima tabella; da tale tabella vengono eliminate tutte le righe che non soddisfano la condizione espressa in WHERE.

JOIN: Esempio

S#	SNAME	STATUS	S.CITY	P#	PNAME	COLOR	WEIGHT	P.CITY
S1	Smith	20	London	P1	Nut	Red	12	London
S1	Smith	20	London	P2	Bolt	Green	17	Paris
S1	Smith	20	London	P3	Screw	Blue	17	Rome
S1	Smith	20	London	P4	Screw	Red	14	London
S1	Smith	20	London	P5	Cam	Blue	12	Paris
S1	Smith	20	London	P6	Cog	Red	19	London
S2	Jones	10	Paris	P1	Nut	Red	12	London
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
S5	Adams	30	Athens	P6	Cog	Red	19	London

JOIN: Risultato

S#	SNAME	STATUS	S.CITY	P#	PNAME	COLOR	WEIGHT	P.CITY
S1	Smith	20	London	P1	Nut	Red	12	London
S1	Smith	20	London	P4	Screw	Red	14	London
S1	Smith	20	London	P6	Cog	Red	19	London
S2	Jones	10	Paris	P2	Bolt	Green	17	Paris
S2	Jones	10	Paris	P5	Cam	Blue	12	Paris
S3	Blake	30	Paris	P2	Bolt	Green	17	Paris
S3	Blake	30	Paris	P5	Cam	Blue	12	Paris
S4	Clark	20	London	P1	Nut	Red	12	London
S4	Clark	20	London	P4	Screw	Red	14	London
S4	Clark	20	London	P6	Cog	Red	19	London

JOIN: Osservazioni

Il costrutto sintattico indica le due tabelle da collegare e la condizione esprime la connessione tra di esse (città uguali)

S#	SNAME	STATUS	CITY
S1	Smith	20	London

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London

S#	SNAME	STATUS	CITY	P#	PNAME	COLOR	WEIGHT	P.CITY
S1	Smith	20	London	P1	Nut	Red	12	London

INNER JOIN

- ***SELECT S.*,P.****
  FROM S,P
  WHERE S.CITY=P.CITY;

Un altro modo alternativo di esprimere sintatticamente il join sopra menzionato è:

- ***SELECT S.*,P.****
  FROM S [INNER] JOIN P ON S.CITY=P.CITY;

OUTER JOIN

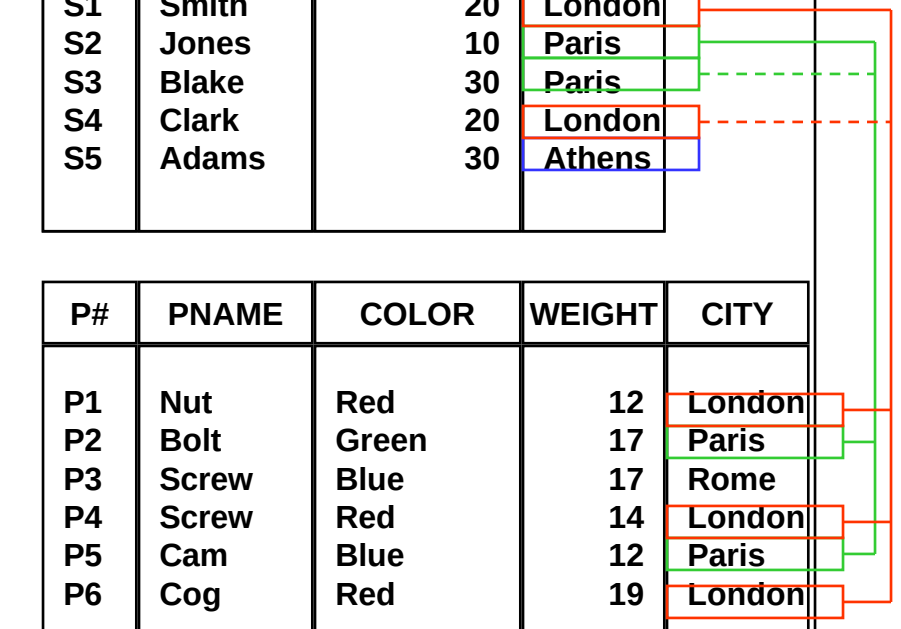
- ***SELECT S.*,P.****
  FROM S LEFT [OUTER] JOIN P ON S.CITY=P.CITY;
- **Mostra tutte le righe di S eventualmente completate con le righe di P in cui la città è uguale.**
- **Quelle righe di S per cui non vi è una riga di P con città uguale, sono completate da una riga di P con colonne aventi valore NULL.**
- **Simmetricamente con RIGHT JOIN.**
- **FULL JOIN completa con valori NULL le righe di entrambe le tabelle dove non ci sia una corrispondenza nelle righe delle rispettive tabelle.**

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London



LEFT OUTER JOIN: Risultato

S#	SNAME	STATUS	S.CITY	P#	PNAME	COLOR	WEIGHT	P.CITY
S1	Smith	20	London	P1	Nut	Red	12	London
S1	Smith	20	London	P4	Screw	Red	14	London
S1	Smith	20	London	P6	Cog	Red	19	London
S2	Jones	10	Paris	P2	Bolt	Green	17	Paris
S2	Jones	10	Paris	P5	Cam	Blue	12	Paris
S3	Blake	30	Paris	P2	Bolt	Green	17	Paris
S3	Blake	30	Paris	P5	Cam	Blue	12	Paris
S4	Clark	20	London	P1	Nut	Red	12	London
S4	Clark	20	London	P4	Screw	Red	14	London
S4	Clark	20	London	P6	Cog	Red	19	London
S5	Adams	30	Athens	NULL	NULL	NULL	NULL	NULL

Natural JOIN

Natural join: provoca l'eliminazione di una delle due colonne 'CITY'.

Equivale a:

```
SELECT S#, SNAME, STATUS, S.CITY, P#,
            PNAME, COLOR, WEIGHT
FROM S,P
WHERE S.CITY=P.CITY;
```

Theta JOIN

Ricava tutte le combinazioni di fornitori e prodotti tali che la città del fornitore segue la città del prodotto in ordine alfabetico:

```
SELECT S.*,P.*
FROM S,P
WHERE S.CITY>P.CITY;
```

Oppure

```
SELECT S.*,P.*
FROM S JOIN P
ON S.CITY>P.CITY;
```

Theta JOIN: esempio

```
SELECT S.*,P.*
FROM S,P
WHERE S.CITY>P.CITY;
```

Risultato :

S#	SNAME	STATUS	S.CITY	P#	PNAME	COLOR	WEIGHT	P.CITY
S2	Jones	10	Paris	P1	Nut	Red	12	London
S2	Jones	10	Paris	P4	Screw	Red	14	London
S2	Jones	10	Paris	P6	Cog	Red	19	London
S3	Blake	30	Paris	P1	Nut	Red	12	London
S3	Blake	30	Paris	P4	Screw	Red	14	London
S3	Blake	30	Paris	P6	Cog	Red	19	London

JOIN con predicati vari

Ricavare tutte le combinazioni di fornitori e parti in cui i fornitori e le parti relative sono della stessa città, tralasciando i fornitori con stato=20

```
SELECT S.*,P.*
FROM S,P
WHERE S.CITY = P.CITY  AND  S.STATUS <> 20;
```

JOIN: esempio

S#	SNAME	STATUS	S.CITY	P#	PNAME	COLOR	WEIGHT	P.CITY
S2	Jones	10	Paris	P2	Bolt	Green	17	Paris
S2	Jones	10	Paris	P5	Cam	Blue	12	Paris
S3	Blake	30	Paris	P2	Bolt	Green	17	Paris
S3	Blake	30	Paris	P5	Cam	Blue	12	Paris

JOIN di 3 tabelle

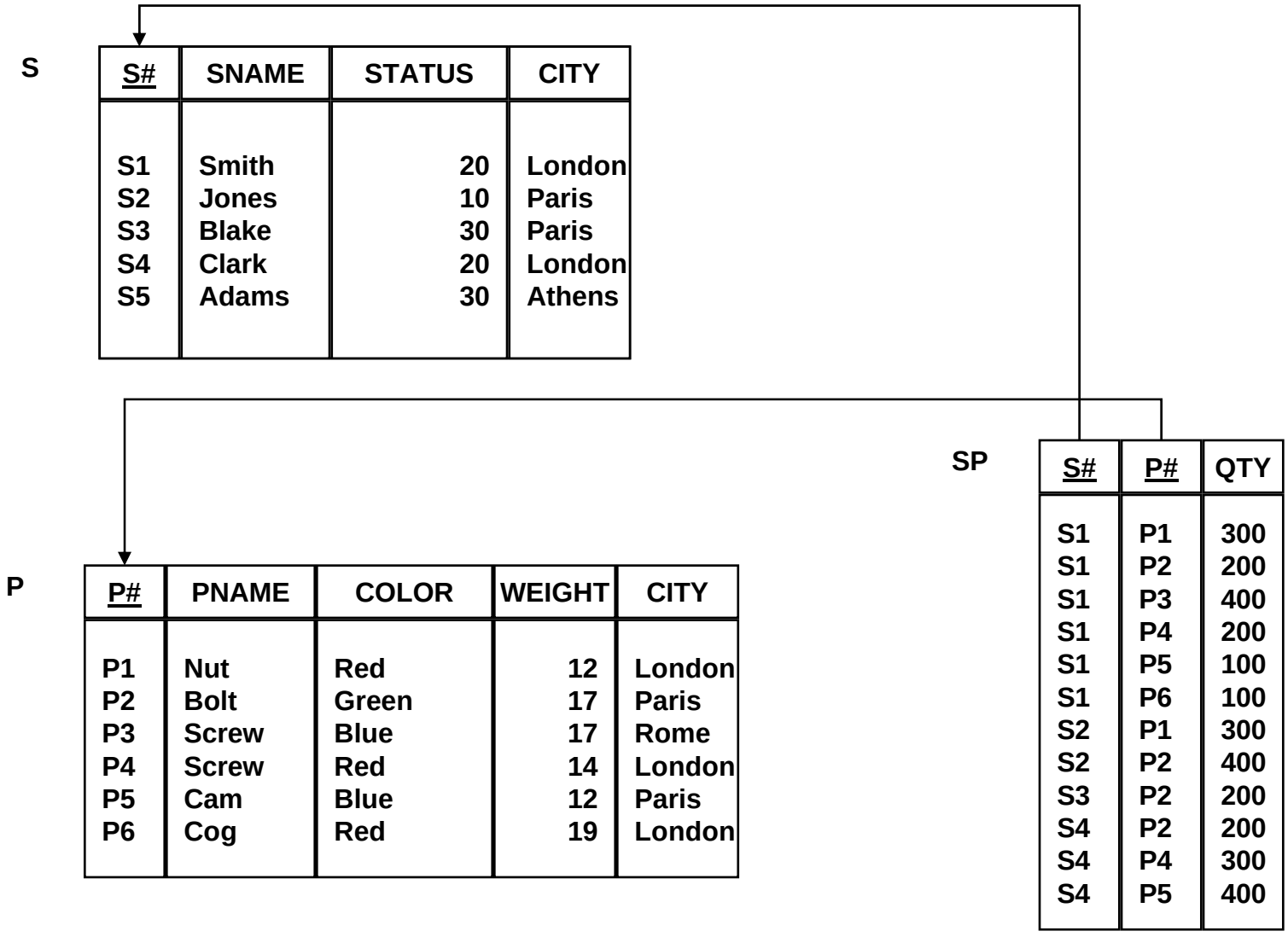
Ricavare tutte le coppie di nomi di città tali che un fornitore della prima città fornisca una parte immagazzinata nella seconda:

```
SELECT DISTINCT S.CITY,P.CITY
FROM S,SP,P
WHERE S.S#=SP.S#
AND SP.P#=P.P#;
```

Si noti l'uso di DISTINCT.

Risultato :

S.CITY	P.CITY
London	London
London	Paris
London	Rome
Paris	London
Paris	Paris



S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

JOIN: campi specifici

Individuare tutti i codici dei fornitori combinati in modo tale che la parte ed il fornitore appartengano alla stessa città.

```
SELECT S.S#,P.P#
FROM S,P
WHERE S.CITY=P.CITY;
```

Risultato :

S#	P#
S1	P1
S1	P4
S1	P6
S2	P2
S2	P5
S3	P2
S3	P5
S4	P1
S4	P4
S4	P6

JOIN di una tabella con se stessa

Individuare tutte le coppie di fornitori tali che i due fornitori appartengano alla stessa città.

```
SELECT FIRST.S#,SECOND.S#
FROM S FIRST,S SECOND
WHERE FIRST.CITY=SECOND.CITY AND
FIRST.S#<SECOND.S#;
```

Risultato :

FIRST.S#	SECOND.S#
S1	S4
S2	S3

Questa operazione si basa sul join della tabella S con se stessa (fatta sulle città uguali). La tabella S compare due volte in FROM.

Funzioni aggregate

- **COUNT(att): conta gli elementi della colonna *att***
- **SUM(att): somma dei valori nella colonna *att***
- **AVG(att): media dei valori della colonna *att***
- **MAX(att): massimo valore nella colonna *att***
- **MIN(att): minimo valore nella colonna *att***

SUM e AVG agiscono su valori numerici.

- **L'argomento della funzione può essere preceduto da DISTINCT.**
- **COUNT(*) conta le righe di una tabella.**
- **COUNT(att) conta i valori *non nulli* in *att***

Nota: non si può annidare due funzioni aggregate: ad esempio SUM(COUNT(*))

Funzioni aggregate

Esempi:

- **conteggio delle tuple in S:**

  ```
  SELECT COUNT(*) FROM S;
  ```

- **uso di DISTINCT:**

  ```
  SELECT COUNT(DISTINCT S#)
  FROM SP;
  ```

- **numero di forniture del prodotto P2:**

  ```
  SELECT COUNT(*) FROM SP
  WHERE P#='P2';
  ```

Funzioni aggregate

– **Ricavare la quantità totale di prodotti P2 forniti.**

SELECT SUM(QTY) FROM SP

WHERE P#='P2'

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Risultato :

1000

Funzioni aggregate

– **Uso di GROUP BY: calcolare la quantità totale fornita per ogni prodotto.**

SELECT P#,SUM(QTY)

FROM SP

GROUP BY P#;

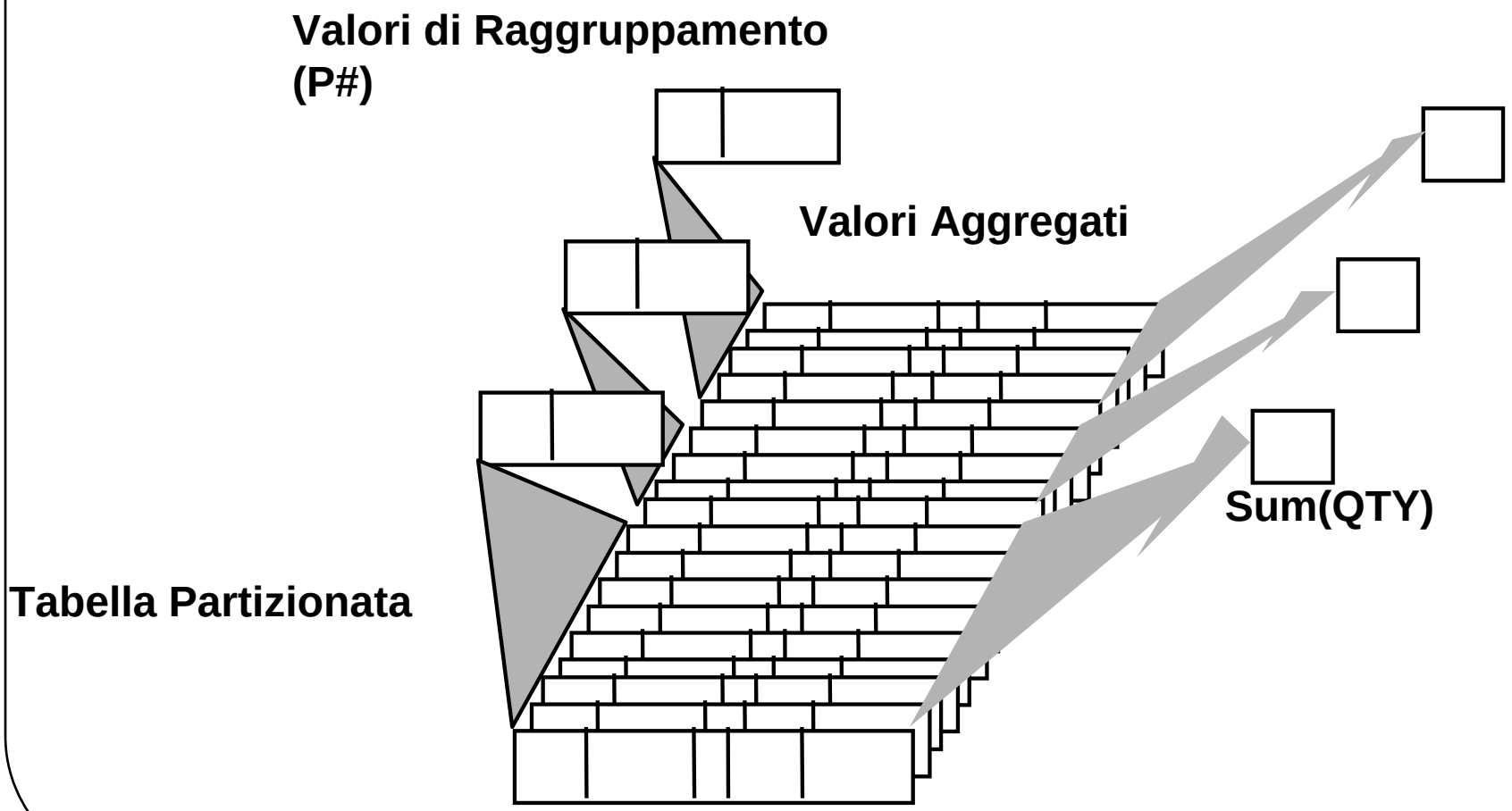
SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Risultato :

P#	....
P1	600
P2	1000
P3	400
P4	500
P5	500
P6	100

GROUP BY


Valori di Raggruppamento
(P#)
Valori Aggregati
Sum(QTY)
Tabella Partizionata

HAVING

Ricavare i codici delle parti fornite da più di un fornitore.

```
SELECT P# FROM SP
GROUP BY P#
HAVING COUNT(*)>1;
```

Risultato :

P#
P1
P2
P4
P5

Having è per Group by ciò che Where è per le righe della tabella

Osservazione

- **SELECT P#**
  FROM SP
  GROUP BY P#

È equivalente a:

- **SELECT DISTINCT P#**
  FROM SP

- **Il DBMS internamente esegue una operazione di ordinamento delle righe per il valore di P# ed elimina i duplicati di P#. (Oppure se esiste un indice su P# potrebbe scandire solo l'indice)**

Ricerca testuale

Si utilizza il costrutto *LIKE:*

column-name LIKE char-string-const

- **il carattere _ sta per qualsiasi carattere**
- **% sta per qualsiasi sequenza di caratteri anche nulla**

Esempio: **ricavare le informazioni sui prodotti i cui nomi iniziano con la lettera C:**

```
SELECT P.* FROM P
WHERE P.PNAME LIKE 'C%';
```

P#	PNAME	COLOR	WEIGHT	CITY
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Ricerca testuale

Esempi:

- **ADDRESS contiene la stringa 'Berkeley'**

  ADDRESS LIKE '%Berkeley%'

- **S# è esattamente di 3 caratteri con il primo uguale a 'S'**

  S# LIKE 'S_ _'

- **PNAME è più lungo o uguale a 4 caratteri con il quart'ultimo uguale a ‘c'**

  PNAME LIKE '%c_ _ _’

- **CITY non contiene una 'E'**

  CITY NOT LIKE '%E%'

Ricerca testuale (cont.)

Esempi:

Supponiamo di voler trovare nell'indirizzo la stringa '%Berkeley' in cui il carattere speciale '%' deve essere considerato come carattere normale.

***Introduciamo un carattere di* escape per far sì che il carattere speciale sia valutato come normale.**

- **ADDRESS contiene la sottostringa '%Berkeley'**

  ADDRESS LIKE '%@%Berkeley%' ESCAPE '@'

La seconda occorrenza del carattere '%' è considerata come tale.

Se volessimo considerare il carattere di escape '@' come tale basterebbe anteporvi un altro '@'. Ad esempio:

- **ADDRESS contiene la sottostringa '@Berkeley'**

  ADDRESS LIKE '%@@Berkeley%' ESCAPE '@':

Valori NULL

Esempio: **ricavare i codici dei fornitori con stato maggiore di 25 (si supponga che S5 abbia stato pari a NULL).**

```
SELECT S#
FROM S
WHERE STATUS>25;
```

Risultato :

S#
S3

S5 non viene restituito perché la condizione STATUS>25 dà risultato UNKNOWN (STATUS ha valore NULL).
Si noti che SELECT restituisce solo le tuple con condizione soddisfatta (TRUE).

Logica a 3 valori

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

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

NOT	*T*	*F*	*U*
	F	T	U

Valori NULL

Esempio: **ricavare i codici dei fornitori con stato maggiore di 25 e nome Adams (e si supponga che il fornitore di nome Adams abbia stato pari a NULL).**

Risultato : empty set

```
SELECT S#
FROM S
WHERE STATUS>25 AND SNAME='Adams';
```

S5 non viene restituito anche se la condizione su SNAME è vera. Infatti STATUS>25 dà risultato UNKNOWN (STATUS ha valore NULL) e di conseguenza AND dà risultato UNKNOWN.

Valori NULL

Il confronto di valori con NULL non dà mai risultati corretti per qualunque operatore di confronto.

- **STATUS <=25 unkwnown**
- **STATUS = 25 unkwnown**
- **STATUS > 25 unkwnown**
- **STATUS <> 25 unkwnown**
- **STATUS = NULL errato**
- **STATUS <> NULL errato**

Valori NULL

Predicato di confronto per il valore NULL:

nome-colonna IS [NOT] NULL

Esempio: **trovare il codice di tutti i fornitori con stato pari a NULL.**

```
SELECT S# FROM S
WHERE STATUS IS NULL;
```

Interrogazione con sottointerrogazione

Una sottointerrogazione è una interrogazione annidata in un'altra interrogazione.

Ne esistono di due tipi:

- ***Stratificate***
  - **permette di valutare completamente (e una volta per tutte) l'interrogazione annidata. Poi, sulla base del suo risultato, viene valutata l'interrogazione più esterna**
- ***Incrociate***
  - **l'interrogazione annidata fa riferimento a una delle relazioni appartenenti all'interrogazione più esterna. Quindi, l'interrogazione annidata deve essere valutata appositamente per ciascuna riga ``candidata'' alla selezione nell'interrogazione più esterna.**

Interrogazione con sottointerrogazione: uso di IN

Una sottointerrogazione annidata in un'altra interrogazione può essere introdotta dal predicato IN.

Esempio: **trovare i nomi dei fornitori che forniscono il prodotto P2.**

```
SELECT SNAME FROM S
WHERE S# IN
    (SELECT S# FROM SP
     WHERE P#='P2');
```

Risultato :

SNAME
Smith
Jones
Blake
Clark

Nota: è una interrogazione stratificata

Interrogazione con sottointerrogazione: uso di IN

- La precedente interrogazione è equivalente al seguente join:

  SELECT S.SNAME FROM S,SP

  WHERE S.S#=SP.S# AND SP.P#='P2':

- Per la base di dati dell'esempio, si può anche scrivere l'interrogazione utilizzando un set predefinito:

  SELECT SNAME FROM S

  WHERE S# IN ('S1','S2','S3','S4');

Interrogazioni annidate

Trovare i nomi dei fornitori che forniscono almeno un prodotto rosso.

```
SELECT SNAME FROM S
WHERE S# IN
    (SELECT S# FROM SP
    WHERE P# IN
        (SELECT P# FROM P
        WHERE COLOR='Red'));
```

SNAME
Smith
Jones
Clark

Nota: è ancora una interrogazione stratificata

Interrogazioni annidate

Trovare il codice dei fornitori che operano nella stessa città di 'S1'.

```
SELECT S# FROM S
WHERE CITY =
        (SELECT CITY FROM S
        WHERE S#='S1');
```

S#
S1
S4

Se è noto a priori che il valore restituito è *unico,* è possibile usare = o > ... al posto di IN.

Altrimenti, in alcuni sistemi (tra cui Oracle) è possibile usare il qualificatore ANY in combinazione con gli operatori di confronto, come = o > …, con l'ovvio significato che la tupla del ciclo SELECT esterno sia = o > di *uno qualunque* dei risultati restituiti dalla SELECT interna.

Interrogazioni annidate

Ricavare i codici dei fornitori il cui stato è minore del valore massimo attualmente presente nella tabella.

```
SELECT S# FROM S
WHERE STATUS <
      (SELECT MAX(STATUS) FROM S);
```

S#
S1
S2
S4

Nota: è ancora una interrogazione stratificata

Interrogazioni annidate: op ANY

Ricavare i codici dei fornitori il cui stato è minore del valore massimo attualmente presente nella tabella.

```
SELECT S# FROM S
WHERE STATUS < ANY
      (SELECT STATUS FROM S);
```

S#
S1
S2
S4

Solo i fornitori (S3 e S5) con stato massimo (30) non vengono selezionati perchè il loro stato *non* è minore dello stato di alcuno

Nota: è ancora una interrogazione stratificata

Interrogazione con EXISTS

Ricavare i nomi dei fornitori che forniscono il prodotto 'P2'.

```
SELECT SNAME FROM S
WHERE EXISTS
      (SELECT * FROM SP
      WHERE SP.S# = S.S#
      AND SP.P# = 'P2');
```

L'espressione EXISTS (SELECT * FROM ...) è vera se e solo se il risultato della SELECT è diverso dall'insieme vuoto.

Interrogazione con EXISTS

```
SELECT SNAME FROM S
  WHERE EXISTS
        (SELECT * FROM SP
        WHERE S# = S.S#
        AND P# = 'P2');
```

E' un'interrogazione incrociata:

Il riferimento a S.S# nell'interrogazione annidata è alla relazione S, candidata alla selezione nell'interrogazione esterna.

Interrogazione con EXISTS

Equivalente a: Selezionare i nomi dei fornitori per i quali esiste una fornitura, qualunque essa sia, relativa al prodotto indicato.

Per valutare il funzionamento:

Considerare ogni valore di SNAME a turno e valutare se il test successivo è vero o falso.

Es. SNAME='Smith',(S#=S1): il set di record di SP, caratterizzati da S#=S1 e P#=P2 è vuoto?, Se no, ciò implica l'esistenza di un record con S#=S1 e P#=P2, pertanto Smith è uno dei valori da restituire. **Parafrasando:** ***selezionare i nomi dei fornitori tali che esista una fornitura relativa al prodotto P2.***

Interrogazione con EXISTS

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Interrogazione con EXISTS

La precedente interrogazione:

```
SELECT SNAME FROM S
WHERE EXISTS
        (SELECT * FROM SP
        WHERE S# = S.S#
        AND P# = 'P2');
```

è equivalente alle seguenti:

```
SELECT SNAME FROM S
 WHERE  S# IN
        (SELECT S# FROM SP
        WHERE SP.P# = 'P2');
```

```
SELECT SNAME
FROM S, SP
WHERE  S.S# = SP.S# AND
        SP.P# = 'P2';
```

NOT EXISTS

Ricavare i nomi dei fornitori che *non* forniscono il prodotto 'P2'.

```
SELECT SNAME FROM S
WHERE NOT EXISTS
      (SELECT * FROM SP
      WHERE S# = S.S#
      AND P# = 'P2');
```

Risultato :

SNAME
Adams

NOT IN

La precedente interrogazione è equivalente alla seguente:

```
SELECT SNAME
FROM S
WHERE S# NOT IN
        (SELECT S# FROM SP
        WHERE P# = 'P2');
```

op ALL

- In alcuni sistemi (tra cui Oracle) è disponibile il qualificatore ALL in combinazione con gli operatori di confronto (=, >, >=, <, <=, <>)
- Confronta il valore dell'attributo del ciclo esterno con tutti i valori restituiti dal ciclo interno

Esercizio

Trovare i codici dei prodotti che hanno un peso maggiore a tutti gli altri prodotti (ossia che hanno il peso massimo).

```
SELECT P1.P#
FROM P P1
WHERE  P1.WEIGHT >ALL  (SELECT P2.WEIGHT
                        FROM P P2 WHERE P2.P#<>P1.P#)
```

Nota: è una interrogazione incrociata

Non si può fare:

```
SELECT P.P#
FROM P
WHERE  P.WEIGHT =MAX(P.WEIGHT)
```

si confronterebbe in una stessa SELECT il valore di dettaglio di una riga con un valore aggregato sulla stessa tavola

op ALL e NOT EXISTS

- **attr op ALL (*SELECT attr' FROM …WHERE …*)**
- **E' equivalente a:**
- **NOT EXISTS (*SELECT attr' FROM .. WHERE ..* AND attr NOT op attr')**

Esercizio

Trovare i codici dei prodotti che hanno un peso maggiore a tutti i prodotti.

```
SELECT P1.P#
FROM P P1
WHERE  NOT EXISTS (SELECT P2.WEIGHT
                   FROM P P2
                   WHERE P2.P#<>P1.P# AND
                         P1.WEIGHT<=P2.WEIGHT)
```

Nota: è una interrogazione incrociata

NOT EXISTS

Trovare i nomi dei fornitori che forniscono tutti i prodotti (elencati in P):

```
SELECT SNAME FROM S
WHERE NOT EXISTS
        (SELECT * FROM P
         WHERE NOT EXISTS
              (SELECT * FROM SP
               WHERE SP.S# = S.S#
               AND SP.P# = P.P#));
```

Nota: è una interrogazione incrociata

NOT EXISTS

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

NOT EXISTS

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

NOT EXISTS

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

NOT EXISTS

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

NOT EXISTS

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

NOT EXISTS

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

NOT EXISTS

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

NOT EXISTS

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

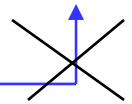
SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

NOT EXISTS

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens



P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

NOT EXISTS

Trovare i codici dei fornitori che forniscono almeno tutti i prodotti forniti da S2.

La ricerca è *concettualmente* frazionabile in passi successivi:

1. **Trovare tutti i codici dei prodotti forniti dal fornitore S2.**

   SELECT P# FROM SP

   WHERE S# = 'S2';

2. **Con CREATE TABLE e INSERT si possono salvare tali dati in una tabella TEMP.**

2. **Successivamente si cercano tutti i fornitori che forniscono tali prodotti.**

NOT EXISTS

```
SELECT S# FROM S
WHERE NOT EXISTS
        (SELECT * FROM TEMP
        WHERE NOT EXISTS
                (SELECT * FROM SP
                WHERE S.S# = SP.S#
                AND SP.P# = TEMP.P#));
```

NOT EXISTS: Soluzione finale

```
SELECT S# FROM S
WHERE NOT EXISTS
        (SELECT * FROM SP SPY
        WHERE  S#='S2' AND NOT EXISTS
                (SELECT * FROM SP SPZ
                WHERE SPZ.S# = S.S#
                AND SPZ.P# = SPY.P# ));
```

Nota:
È sempre bene ottenere una unica query sulla tabella originaria e *non* materializzare la query intermedia in una tabella ex-novo (come temp) perchè su tale tabella l'ottimizzatore non ha indici che possano aiutare!

UNION, INTERSECT e EXCEPT

Sono gli operatori insiemistici su due relazioni A e B (restituite da due SELECT).

Per poterli usare occorre che le due relazioni siano definite sullo stesso insieme di attributi.

A UNION B è l'insieme delle tuple x tali che x appartiene ad A o appartiene a B o appartiene ad entrambi. Le repliche vengono eliminate.

A INTERSECT B è l'insieme delle tuple x tali che x appartiene ad A e appartiene anche a B.

A EXCEPT B è l'insieme delle tuple di A che non sono presenti (anche) in B.

Nota: In Oracle, EXCEPT si chiama MINUS.

UNION

Ricavare i codici dei prodotti che o pesano più di 16 o sono forniti dal fornitore S2, o entrambe le cose:

```
SELECT P# FROM P
WHERE WEIGHT > 16
UNION
SELECT P# FROM SP
WHERE S# = 'S2';
```

Esempio alternativo (ma peggiore!)

Ricavare i codici dei prodotti che o pesano più di 16 o sono forniti dal fornitore S2, o entrambe le cose:

Con una join invece che con una UNION:

```
SELECT P# FROM P, SP
WHERE   P.P#=SP.P#   AND   (SP.S#   =   'S2'
                OR P.WEIGHT > 16);
```

È peggiore perchè fa un prodotto cartesiano tra S e SP

Esercizio

- **Trovare i codici dei fornitori che hanno fornito almeno un prodotto di colore rosso o verde**

1. SELECT DISTINCT SP.S#
   FROM SP, P
   WHERE SP.P#=P.P# AND (P.color='Red' OR P.color='Green')
2. SELECT SP.S#
   FROM SP, P
   WHERE SP.P#=P.P# AND P.color='Red'
   UNION
   SELECT SP.S#
   FROM SP, P
   WHERE SP.P#=P.P# AND P.color='Green'

Esercizio

- **Trovare i codici dei fornitori che hanno fornito almeno un prodotto di colore rosso e uno verde**

1. SELECT DISTINCT SP.S#
   FROM SP SP1, P P1, SP SP2, P P2
   WHERE SP1.P#=P1.P# AND P1.color='Red' AND
   SP2.P#=P2.P# AND P2.color='Green' AND SP1.S#=SP2.S#

2. SELECT DISTINCT S# FROM
   ((SELECT SP.S#
   FROM SP, P
   WHERE SP.P#=P.P# AND P.color='Red')
   INTERSECT
   (SELECT SP.S#
   FROM SP, P
   WHERE SP.P#=P.P# AND P.color='Green'))

Esercizio

- **Trovare i fornitori che <u>non</u> hanno fornito un prodotto di colore rosso**

1. SELECT S#
   FROM S
   WHERE S# NOT IN (SELECT SP.S#
   FROM SP, P
   SP.P#=P.P# AND P.color='Red')

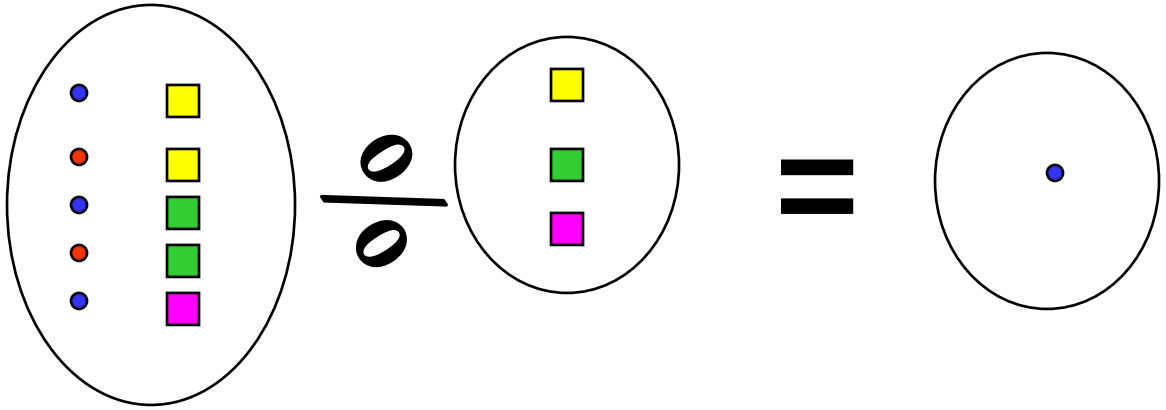
2. SELECT S#
   FROM S
   EXCEPT
   SELECT SP.S#
   FROM SP, P
   WHERE SP.P#=P.P# AND P.color='Red'

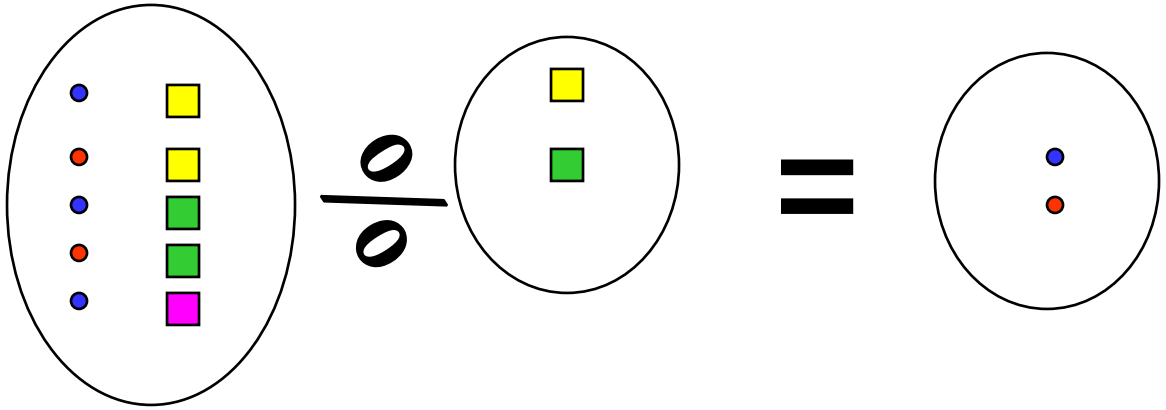
Quantificatore universale con uso della differenza e NOT EXISTS in SQL

- **Esempio:** Trovare i codici dei fornitori che forniscono (almeno) tutti i prodotti forniti da S2.

```
SELECT S#
FROM S
WHERE NOT EXISTS
        ((SELECT SP1.P#
        FROM SP SP1 WHERE SP1.S#='S2')
        EXCEPT
        (SELECT SP2.P#
        FROM SP SP2 WHERE SP2.S#=S.S#))
```

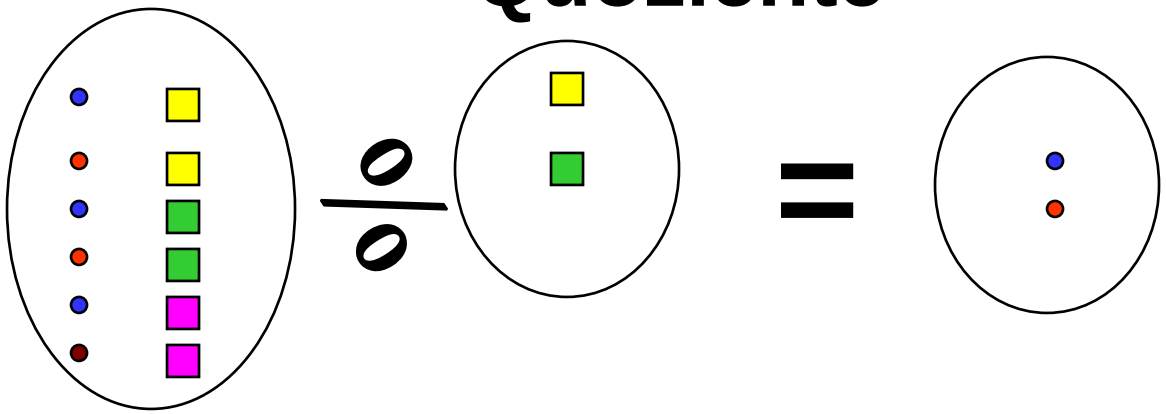
Quoziente





dividendo divisore quoziente

Quoziente



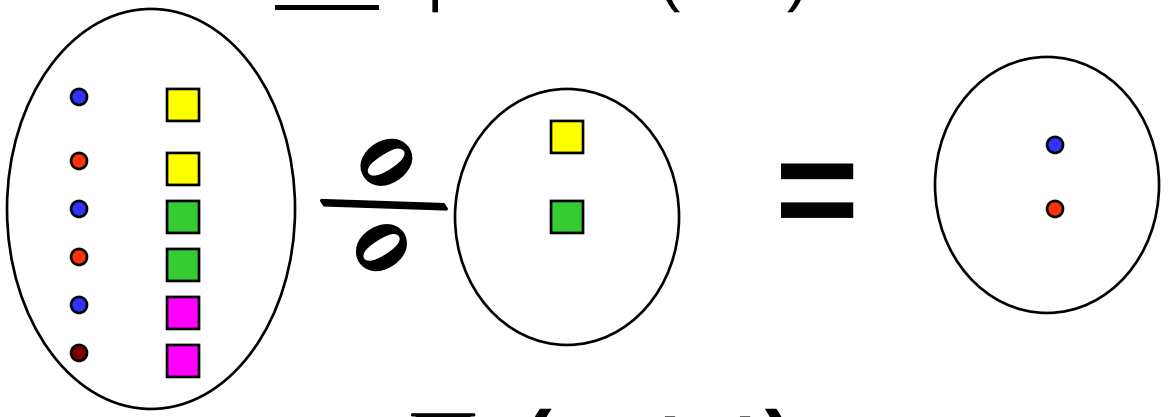
dividendo **divisore** **quoziente**

Il quoziente è costituito da tutti quegli elementi tali che:

1. **appartengono al dividendo**
2. **nel dividendo sono associati a <u>tutti</u> gli elementi del divisore**

Metodo 1. Quantificazione universale con uso del quoziente

- **Esempio:** Trovare i codici dei fornitori (S') che forniscono <u>tutti</u> i prodotti (■ ■) forniti da S2 (●).



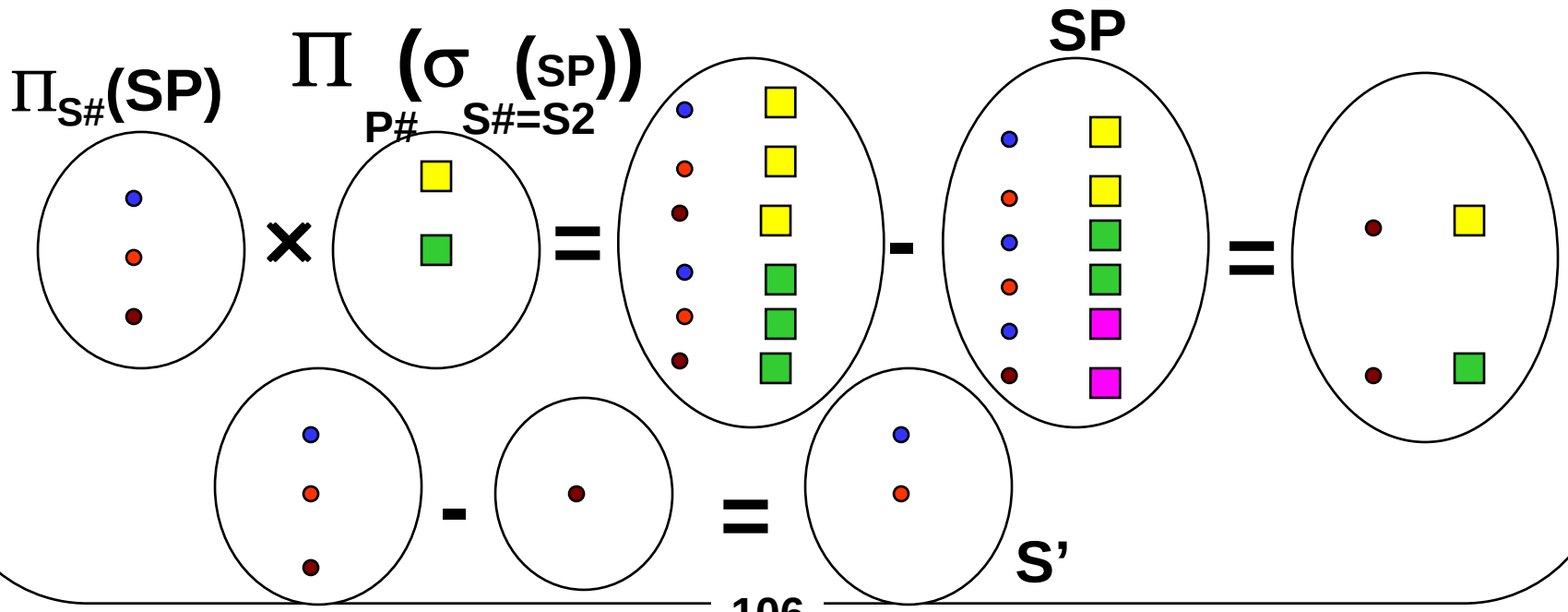
SP $\Pi_{P\#}(\sigma_{S\#=S2}(SP))$ **S'**

1. **S'=** $\Pi_{S\#}(SP / \Pi_{P\#}(\sigma_{S\#=S2} SP))$

Metodo 2: Quantificazione universale con uso della differenza in algebra relazionale

- **Esempio:** Trovare i codici dei fornitori (S') che forniscono <u>tutti</u> i prodotti forniti da S2 (•).

2. $S' = (\Pi_{S\#} SP) - \Pi_{S\#}((\Pi_{S\#}(SP) \times \Pi_{P\#}(\sigma_{S\#=S2} SP)) - \Pi_{S\#,P\#}(SP))$



Metodo 3: Quantificazione universale in logica

- **Esempio:** Trovare i codici dei fornitori che forniscono tutti i prodotti forniti da S2.
- **$(\exists SX)\ SP(SX,PX) \wedge$**
  $\neg(\exists PY)[SP(SY,PY) \wedge SY='S2' \wedge$
  $\neg(\exists PZ,SZ)(SP(SZ,PZ) \wedge PZ=PY \wedge SZ=SX)]$
- **SELECT S# FROM SP SPX**
  WHERE NOT EXISTS (
  SELECT P# FROM SP SPY
  WHERE SPY.S#='S2' AND NOT EXISTS (
  SELECT P# FROM SPZ
  WHERE SPZ.S#=SPX.S# AND SPZ.P#=SPY.P#))

Operatore di aggiornamento

Istr-aggiornamento ::= UPDATE nome-tabella
SET colonna = espressione
{,colonna=espressione}
[WHERE predicato]

Tutti i record della tabella *nome-tabella* che soddisfano il predicato vengono modificati in base all'assegnazione *colonna=espressione* nell'opzione *SET*.

Aggiornamento di un solo record

```
UPDATE P
    SET COLOR = 'Yellow' ,
    WEIGHT=WEIGHT+12,
    CITY = NULL,
    WHERE P# = 'P1';
```

L'aggiornamento viene effettuato per il record corrispondente al codice P1.

Aggiornamento multiplo

Aggiornare lo stato al doppio del valore per tutti i fornitori di Londra.

```
UPDATE S
    SET STATUS = 2 * STATUS
    WHERE CITY = 'London'
```

L'aggiornamento avviene per tutti i record che soddisfano la condizione specificata.

Aggiornamento con sottointerrogazione

Aggiornare a 0 la quantità fornita per tutti i fornitori di Londra.

```
UPDATE SP
    SET QTY = 0
    WHERE 'London'= (SELECT CITY FROM S
                     WHERE S.S# = SP.S#);
```

Aggiornamento di più tabelle

Modificare il codice del fornitore S2 a S9.

```
UPDATE S
    SET S# = 'S9'
        WHERE S# = 'S2';

UPDATE SP
    SET S# = 'S9'
        WHERE S# = 'S2';
```

Aggiornamento di più tabelle

Una istruzione UPDATE può aggiornare una sola tabella. Si manifesta un problema di integrità dopo aver modificato la tabella dei fornitori. Per mantenere la coerenza è necessario completare l'aggiornamento di entrambe le tabelle.

Operatore di cancellazione

Istr-canc-in-tab ::= DELETE FROM nome-tabella
[WHERE predicato];

Cancella dalla tabella *nome-tabella* tutti i record che soddisfano il predicato.

Esempio: **cancellare tutte le forniture.**

DELETE FROM SP;

Svuota la tabella SP.

Cancellazione di record

Cancellare il record corrispondente al fornitore con codice S1.

```
DELETE FROM S
WHERE S# = 'S1';
```

Attenzione: **se SP contiene dei riferimenti a S1 il database perde la propria integrità.**

Cancellare tutti i fornitori di Madrid.

```
DELETE FROM S
    WHERE CITY = 'Madrid';
```

Cancella un insieme di fornitori!

Cancellazione con sottointerrogazione

Cancellare le forniture dei fornitori di Londra.

```
DELETE FROM SP
    WHERE 'London'= (SELECT CITY FROM S
                     WHERE S.S# = SP.S#);
```

Operatore di inserimento

Due possibilità:

Istr-inserimento-in-tab ::=
INSERT INTO nome-tabella
[(colonna {,colonna})]
VALUES (costante {, costante})
| *INSERT INTO nome-tabella*
[(colonna {,colonna})]
SELECT ... FROM ... WHERE ... ;

Inserimento di un solo record

INSERT INTO P (P#,CITY,WEIGHT)

VALUES ('P7','Athens',24);

Viene creato un nuovo record per il prodotto P7, NAME e COLOR sono inizializzati a NULL (Attenzione alla CREATE TABLE).

Inserimento di un solo record

Inserire il prodotto P8 (name: Sprocket, colour: Pink, Weight: 12, city: Nice).

INSERT INTO P

VALUES ('P8','Sprocket','Pink',12,'Nice');

Omettere la lista dei campi equivale a specificare tutti i campi secondo l'ordine di creazione delle colonne nella tabella.

Inserimento di un solo record

Inserire una nuova fornitura con fornitore S20, parte P20 e quantità 1000.

```
INSERT INTO SP (S#, P#, QTY)
    VALUES ('S20','P20',1000);
```

Attenzione: **in questo caso è necessario che P20 e S20 esistano in S e P (problema di integrità referenziale).**

Inserimento di più record

Per ogni prodotto trovare il codice e la corrispondente quantità totale di forniture, salvando poi il risultato nel database.

```
CREATE TABLE TEMP
    (P# CHAR(6),
    TOTQTY INTEGER);
INSERT INTO TEMP (P#, TOTQTY)
    (SELECT P#, SUM(QTY) FROM SP
    GROUP BY P#);
```

Inserimento di più record

- SELECT restituisce dati che vengono immediatamente inseriti nella tabella TEMP.
- La tabella TEMP resta a disposizione per successive elaborazioni.
- Alla fine la tabella TEMP può essere eliminata:

  DROP TABLE TEMP;

Interrogazione complessa

Per tutti i prodotti rossi o blu, tali che la quantità totale fornita sia superiore a 350 (escludendo dal totale tutte le forniture per cui la quantità è minore di 200), ricavare il codice dei prodotti, il loro peso in grammi, il colore e la quantità massima fornita, ordinando il risultato in ordine di valori crescenti della quantità massima e in ordine decrescente per codice prodotto.

Interrogazione complessa

```
SELECT P.P#, 'Weight =', P.WEIGHT*454,
            P.COLOR,'Max qty =' ,MAX(SP.QTY)
FROM P,SP
WHERE P.P#=SP.P#
AND (P.COLOR='Red' OR P.COLOR='Blue')
AND SP.QTY>=200
GROUP BY P.P#,P.WEIGHT,P.COLOR
HAVING SUM(QTY) > 350
ORDER BY 6, P.P# DESC;
```

Interrogazione complessa

a) FROM: dà origine al prodotto cartesiano delle due tabelle P e SP

b) WHERE: il risultato di (a) viene ridotto eliminando le righe che non soddisfano i predicati

c) GROUP BY: il risultato di (b) è raggruppato secondo i valori dei campi indicati: P.P#, P.WEIGHT, P.COLOR

d) HAVING: i gruppi che non soddisfano la condizione SUM(QTY) sono eliminati dal risultato di (c)

Interrogazione complessa

e) **SELECT: ogni gruppo in (d) genera una singola riga; viene estratto da ogni gruppo il codice della parte, il peso, il colore e la massima quantità**

f) **ORDER BY: il risultato del quinto passo viene ordinato secondo la specifica**