



Basi Dati

SQL – Lo standard dei database relazionali

a.a. 2021/2022
Prof.ssa G. Tortora

Obiettivi di Apprendimento

- Dopo aver studiato questo capitolo dovresti essere in grado di:
 - Definire un database usando il linguaggio di definizione di **Structured Query Language** (SQL).
 - Scrivere comandi SQL per inserire, cancellare ed aggiornare i dati del database.
 - Scrivere query singole e multiple usando i comandi di SQL.
 - Spiegare il concetto di vista e saperlo utilizzare.
 - Saper definire i vincoli di integrità.

Come manipolare un db relazionale

- L'algebra relazionale è il mezzo per interrogare basi di dati relazionali, ma non è sfruttabile in ambito commerciale, principalmente perché le query sono scritte **specificando l'ordine** con cui devono essere eseguite le operazioni.
- SQL è un linguaggio **dichiarativo**, basato in parte sull'algebra relazionale ed in parte sul calcolo relazionale.
 - L'utente specifica **quale risultato** deve essere raggiunto.
 - Il DBMS decide l'ordine delle operazioni.

SQL

- Il linguaggio SQL permette la **definizione**, la **manipolazione** (aggiornamento e recupero) e la **gestione** di basi di dati relazionali.
- È una delle ragioni del successo dei db relazionali in ambito commerciale:
essendo uno standard in tutti i DBMS relazionali, gli utenti sono poco propensi a migrare verso data model diversi, quali il gerarchico o il reticolare.

SQL – i vantaggi di uno standard (1)

- SQL è considerato la principale ragione di successo dei database relazionali.
- Infatti:
 - **È uno standard** – è possibile migrare da un DBMS relazionale verso un altro DBMS relazionale senza eccessivi problemi.
 - I programmi applicativi usano lo stesso insieme di istruzioni SQL per accedere a DBMS relazionali diversi, senza dover cambiare i sottolinguaggi.

SQL – i vantaggi di uno standard (2)

- *I costi di formazione del personale sono ridotti: la formazione è concentrata su di un solo linguaggio.*
- *Produttività: i tecnici, una volta imparato il linguaggio e poiché usano solo questo diventano sempre più esperti e produttivi.*
- *Portabilità delle applicazioni: le applicazioni possono essere spostate da una macchina all'altra, se entrambe usano SQL.*

SQL – i vantaggi di uno standard (3)

- *Longevità delle applicazioni*: uno standard tende a rimanere in voga per diverso tempo e non c'è necessita di riscrivere le applicazioni.
- *Ridotta dipendenza da un singolo produttore*: quando non è usato un linguaggio proprietario, è più facile usare differenti produttori di DBMS.
Di conseguenza il mercato sarà più competitivo ed i prezzi calano.
- *Comunicazione fra sistemi*: differenti DBMS e programmi applicativi possono comunicare più facilmente.

Ma ha anche degli svantaggi...

- Può in qualche modo limitare la creatività e l'innovazione.
- Può non incontrare tutte le necessità di un'industria.
- Può essere difficile da cambiare, poiché molti produttori hanno investito su di esso.
- L'utilizzo di speciali funzionalità aggiunte a SQL da qualche produttore può far perdere i vantaggi di cui alle slide precedenti.

SQL: un po' di storia

- Nel 1970 Codd propone il modello relazionale: iniziano esperimenti e ricerche per la realizzazione di linguaggi relazionali, cioè di linguaggi in grado di realizzare le caratteristiche del modello astratto.
- Il primo risultato è *SEQUEL* (*Structured English QUery Language*), definito all'IBM Research:
 - Facile da imparare e utilizzare;
 - Basato su termini inglesi che mascherano i difficili concetti dell'algebra relazionale.

SQL: un po' di storia (2)

- Una versione rivista, il **SEQUEL/2**, ridenominata **SQL (Structured Query Language)** viene definita nel 1976.
- Il primo prodotto basato su SQL viene chiamato **Oracle** (1979), lanciato dalla Relational Software, Inc.
- Nel 1981 IBM annuncia un prodotto SQL denominato **SQL/Data System**; nel 1983 viene rilasciato il DBMS relazionale **DB2** compatibile con SQL/DS.

SQL: un po' di storia (3)

- Oggi SQL è implementato da tutti i principali fornitori di DBMS, ed è il linguaggio per database più usato al mondo.
- L'ANSI e l'ISO hanno sviluppato una serie di standard per SQL:
 - ANSI SQL-86, SQL-92 (SQL2) ed SQL3.
 - Sfortunatamente ogni DBMS relazionale implementa un suo livello (o **dialetto**) di SQL, che è un'estensione o un sottoinsieme di un livello standard.

Il linguaggio SQL

- SQL fornisce *statement* per la definizione di dati, query e aggiornamenti, quindi è sia un **DDL** che un **DML**.
- Fornisce inoltre facility per definire viste e per ricavare indici.
- SQL può essere usato **interattivamente** (con maschere del DBMS) o essere **incorporato** (**embedded**) in programmi C, Cobol, Java, etc.

Definizione di dati, schemi e vincoli in SQL2

Terminologia SQL

- SQL ha una terminologia diversa da quella classica relazionale:
 - Relazione → Tabella
 - Tupla → Riga
 - Attributo → Colonna

Concetti di schema

- Il concetto di **schema** SQL è usato per raggruppare tabelle ed altri costrutti che appartengono alla stessa applicazione di database.
- Uno schema SQL è identificato da un **nome dello schema**, ed include un identificatore di autorizzazione per indicare l'utente proprietario dello schema, così come dei **descrittori** per ogni elemento dello schema.

Concetto di schema (2)

- Uno schema include:
 - Tabelle
 - Domini
 - Viste
 - Altri costrutti, quali permessi di autorizzazione, etc.
- La sintassi per creare uno schema è
CREATE SCHEMA *nome_schema*
AUTHORIZATION *nome_utente*
 - Crea uno schema chiamato *nome_schema*, il cui proprietario è l'utente con account *nome_utente*.

Il comando CREATE TABLE

- **CREATE TABLE** è usato per specificare una nuova relazione, assegnandole un **nome** ed un **insieme di attributi** e **vincoli**.
- Gli attributi sono specificati da un **nome**, un **tipo di dato** per definire il dominio dei valori, ed eventuali **vincoli**.
- In ultimo si specifica la chiave, i vincoli di integrità di entità e di integrità referenziale.

CREATE TABLE: *Esempio*

CREATE TABLE EMPLOYEE

```
( FNAME          VARCHAR(15)      NOT NULL ,
  MINIT          CHAR           ,
  LNAME          VARCHAR(15)      NOT NULL ,
  SSN            CHAR(9)         NOT NULL ,
  BDATE          DATE
  ADDRESS        VARCHAR(30) ,
  SEX            CHAR           ,
  SALARY         DECIMAL(10,2) ,
  SUPERSSN       CHAR(9) ,
  DNO            INT             NOT NULL ,
  PRIMARY KEY (SSN) ,
  FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN) ,
  FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER)) ;
```

CREATE TABLE DEPARTMENT

```
( DNAME          VARCHAR(15)      NOT NULL ,
  DNUMBER        INT             NOT NULL ,
  MGRSSN         CHAR(9)         NOT NULL ,
  MGRSTARTDATE   DATE ,
  PRIMARY KEY (DNUMBER) ,
  UNIQUE (DNAME) ,
  FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN)) ;
```

CREATE TABLE DEPT_LOCATIONS

```
( DNUMBER        INT             NOT NULL ,
  DLOCATION        VARCHAR(15)     NOT NULL ,
  PRIMARY KEY (DNUMBER, DLOCATION) ,
  FOREIGN KEY (DNUMBER) REFERENCES DEPARTMENT(DNUMBER)) ;
```

CREATE TABLE PROJECT

```
( PNAME          VARCHAR(15)      NOT NULL ,
  PNUMBER        INT             NOT NULL ,
  PLOCATION        VARCHAR(15) ,
  DNUM           INT             NOT NULL ,
  PRIMARY KEY (PNUMBER) ,
  UNIQUE (PNAME) ,
  FOREIGN KEY (DNUM) REFERENCES DEPARTMENT(DNUMBER)) ;
```

CREATE TABLE WORKS_ON

```
( ESSN           CHAR(9)         NOT NULL ,
  PNO            INT             NOT NULL ,
  HOURS          DECIMAL(3,1)    NOT NULL ,
  PRIMARY KEY (ESSN, PNO) ,
  FOREIGN KEY (ESSN) REFERENCES EMPLOYEE(SSN) ,
  FOREIGN KEY (PNO) REFERENCES PROJECT(PNUMBER)) ;
```

CREATE TABLE DEPENDENT

```
( ESSN           CHAR(9)         NOT NULL ,
  DEPENDENT_NAME VARCHAR(15)     NOT NULL ,
  SEX            CHAR           ,
  BDATE          DATE ,
  RELATIONSHIP    VARCHAR(8) ,
  PRIMARY KEY (ESSN, DEPENDENT_NAME) ,
  FOREIGN KEY (ESSN) REFERENCES EMPLOYEE(SSN)) ;
```

Gli statement SQL2 per definire lo schema del database "*Company*"

Tipi di dati e domini

- Tipi di dati disponibili in SQL2:
 - Numerici
 - Interi (INTEGER o INT, SMALLINT)
 - Reali (FLOAT, REAL, DOUBLE PRECISION)
 - Numeri formattati (DECIMAL(i,j), DEC(i,j), NUMERIC(i,j))
 - i, detta precisione, indica il numero di cifre decimali.
 - j, detta scala, indica il numero di cifre dopo la virgola.
 - Stringhe di caratteri
 - A lunghezza fissa (CHAR(n), CHARACTER(n))
 - A lunghezza variabile (VARCHAR(n) o CHAR VARYING(n))
 - Per default n, il numero massimo di caratteri, è 1.

Tipi di dati e domini (2)

- Stringhe di bit:
 - A lunghezza fissa (BIT(n))
 - A lunghezza variabile (BIT VARYING(n))
- DATE:
 - Ha dieci posizioni, con componenti YEAR, MONTH e DAY.
 - Formato *YYYY-MM-DD*.
- TIME:
 - Ha (almeno) otto posizioni con componenti HOUR, MINUTE e SECOND.
 - Formato *HH:MM:SS*.

Domini personalizzati

- In SQL2 è possibile sia dichiarare il tipo di dato di un attributo, sia dichiarare il dominio.
 - Ciò semplifica il cambiamento di un tipo per un dominio usato più volte nello schema.
- *Esempio:*
 - **CREATE DOMAIN SSN_TYPE AS CHAR(9);**
e poi si usa SSN_TYPE per gli attributi
 - SSN e SUPERSSN di EMPLOYEE
 - MGRSSN e ESSN di DEPARTMENT
 - ESSN di WORKS_ON

I valori null e default

- Poiché SQL consente che un attributo abbia valore **null**, se si vuole impedire ciò si usa il vincolo **NOT NULL**.
 - Tale vincolo deve sempre essere specificato per la chiave primaria (*vincolo di integrità di entità*).
- È anche possibile specificare un valore di default per un attributo, attraverso la clausola **DEFAULT** <value>, dopo la dichiarazione dell'attributo
 - Senza tale clausola il valore di default di un attributo è null.

Altri vincoli

- Dopo le specifiche degli attributi, possono essere specificati i **vincoli di tabella**, quali **chiave** ed **integrità referenziale**:
 - La clausola **PRIMARY KEY** specifica uno o più attributi che faranno da chiave primaria.
 - La clausola **UNIQUE** specifica una chiave alternativa.
 - La clausola **FOREIGN KEY** specifica l'integrità referenziale.

Altri vincoli (2)

- Il progettista dello schema può specificare l'azione da intraprendere se si viola un vincolo di integrità referenziale, attraverso la cancellazione di una tupla referenziata o attraverso la modifica di un valore di chiave referenziata.
- L'azione referenziale **triggered** può essere specificata nella clausola **FOREIGN KEY**.
 - Possibili azioni sono **SET NULL**, **CASCADE** e **SET DEFAULT**, qualificate da opzioni **ON DELETE** e **ON UPDATE**.

Altri vincoli: *Esempio*

```
CREATE TABLE EMPLOYEE
(
    ...,
    DNO          INT    NOT NULL    DEFAULT 1,
    CONSTRAINT EMPPK
        PRIMARY KEY (SSN) ,
    CONSTRAINT EMPSUPERFK
        FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN)
            ON DELETE SET NULL    ON UPDATE CASCADE ,
    CONSTRAINT EMPDEPTFK
        FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER)
            ON DELETE SET DEFAULT  ON UPDATE CASCADE );

CREATE TABLE DEPARTMENT
(
    ...,
    MGRSSN CHAR(9) NOT NULL DEFAULT '888665555' ,
    ...,
    CONSTRAINT DEPTPK
        PRIMARY KEY (DNUMBER) ,
    CONSTRAINT DEPTSK
        UNIQUE (DNAME),
    CONSTRAINT DEPTMGRFK
        FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN)
            ON DELETE SET DEFAULT  ON UPDATE CASCADE );

CREATE TABLE DEPT_LOCATIONS
(
    ...,
    PRIMARY KEY (DNUMBER, DLOCATION),
    FOREIGN KEY (DNUMBER) REFERENCES DEPARTMENT(DNUMBER)
        ON DELETE CASCADE  ON UPDATE CASCADE );
```

Specifica di valori di default e azioni referenziali triggered.

Altri vincoli: *Esempio*

- Nell'esempio, per la chiave esterna SUPERSSN di EMPLOYEE ci sono i vincoli:
 - **SET NULL ON DELETE**
 - Se la tupla dell'impiegato che supervisiona viene cancellata, il valore di SUPERSSN è posto a null in tutte le tuple impiegato che lo referenziano.
 - **CASCADE ON UPDATE**
 - Se il valore SSN di un impiegato che supervisiona è aggiornato, il nuovo valore è riportato in SUPERSSN di tutte le tuple impiegato che referenziano il valore aggiornato.
- Ai vincoli può essere dato un nome (per poterli riutilizzare), usando la keyword **CONSTRAINT**.

Relazioni base e virtuali

- Le relazioni create con **CREATE TABLE** sono dette **tabelle base** o **relazioni base** in SQL, e significa che sono create e memorizzate come file dal DBMS.
- Le relazioni base sono distinte dalle **relazioni virtuali**, create mediante **CREATE VIEW**, cui può o meno corrispondere un file fisico.
- In SQL gli attributi sono considerati ordinati nella sequenza in cui sono stati specificati.
Le righe non sono considerate ordinate.

Il comando DROP SCHEMA

- Se uno schema non è più necessario, si usa il comando **DROP SCHEMA**, con due possibili opzioni (*drop behaviour*): **CASCADE** e **RESTRICT**.
- *Esempi:*
 - **DROP SCHEMA COMPANY CASCADE;**
 - Lo schema del db COMPANY viene rimosso, con tutte le tabelle, domini ed altri elementi.
 - **DROP SCHEMA COMPANY RESTRICT;**
 - Lo schema è eliminato solo se non contiene elementi.

Il comando DROP TABLE

- Permette di eliminare una tabella:
 - **DROP TABLE DEPENDENT CASCADE;**
 - Se non si vuole tenere più traccia delle persone a carico nel db COMPANY.
 - **DROP TABLE DEPENDENT RESTRICT;**
 - La tabella è eliminata solo se non è referenziata in alcun vincolo o vista.
 - Con l'opzione CASCADE sarebbero automaticamente eliminati insieme alla tabella stessa.

Il comando **ALTER TABLE**

- La definizione di una tabella base può essere cambiata usando il comando **ALTER TABLE**.
- Possibili azioni di modifica di una tabella sono:
 - Aggiunta o rimozione di attributi;
 - Cambio di definizione di una colonna;
 - Aggiunta o rimozione di un vincolo.

ALTER TABLE: *Esempio*

- Vogliamo aggiungere il lavoro svolto da un impiegato nella tabella Employee:
 - **ALTER TABLE COMPANY.EMPLOYEE ADD JOB VARCHAR(12);**
- Il valore di JOB o si specifica di **default** o sarà **null**.
- *Con la ALTER TABLE non è permessa la clausola NOT NULL.*

ALTER TABLE: *Esempio* (2)

- Vogliamo eliminare una colonna: occorre scegliere l'opzione CASCADE o RESTRICT:
 - Con CASCADE tutti i vincoli e le viste che referenziano la colonna sono eliminati automaticamente dallo schema.
 - Con RESTRICT il comando ha successo solo se nessun vincolo o vista referencia la colonna.
- ***Esempio:*** rimuovere ADDRESS dalla tabella EMPLOYEE:
 - **ALTER TABLE COMPANY.EMPLOYEE DROP ADDRESS CASCADE;**

ALTER TABLE: *Esempio* (3)

- Modifica di una colonna eliminando una clausola di default o definendone una nuova.
- ***Esempio:***
 - **ALTER TABLE COMPANY.DEPARTMENT ALTER MGRSSN DROP DEFAULT;**
 - **ALTER TABLE COMPANY.DEPARTMENT ALTER MGRSSN SET DEFAULT “333444555”;**
- Cambio di vincoli:
 - È possibile eliminare un vincolo solo se ha un nome:
 - **ALTER TABLE COMPANY.EMPLOYEE DROP CONSTRAINT EMPSUPERFK CASCADE;**

Esempio 1

- Definire un dominio che permetta di rappresentare stringhe di lunghezza massima pari a 256 caratteri, su cui non sono ammessi valori nulli e con valore di default “*sconosciuto*”.

Soluzione 1

- CREATE DOMAIN String AS CHARACTER VARYING (256) DEFAULT 'sconosciuto' NOT NULL;

Esempio 2

- Dare le definizioni SQL delle tre entità:

FONDISTA(Nome, Nazione, Età*)

GAREGGIA(NomeFondista↑, NomeGara ↑, Piazzamento)

GARA(Nome, Luogo, Nazione, Lunghezza*)

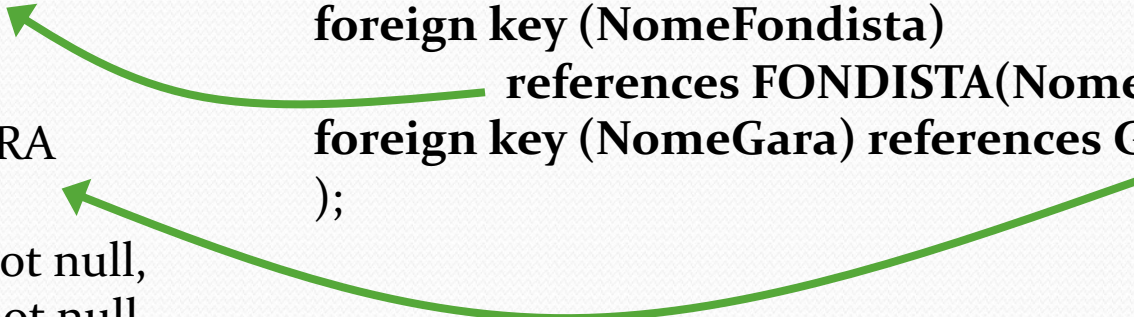
rappresentando in particolare i vincoli di foreign key della tabella GAREGGIA.

Soluzione 2

```
CREATE TABLE FONDISTA  
(  
  Nome varchar(20) not null,  
  Nazione varchar(30) not null,  
  Età smallint,  
  primary key (Nome)  
);
```

```
CREATE TABLE GARA  
(  
  Nome varchar(20) not null,  
  Luogo varchar(20) not null,  
  Nazione varchar(30) not null,  
  Lunghezza integer,  
  primary key (Nome)  
);
```

```
CREATE TABLE GAREGGIA  
(  
  NomeFondista varchar(20) not null,  
  NomeGara varchar(20) not null,  
  Piazzamento smallint,  
  primary key (NomeFondista, NomeGara),  
  foreign key (NomeFondista)  
    references FONDISTA(Nome)  
  foreign key (NomeGara) references GARA(Nome)  
);
```



Esempio 3

- Dare le definizioni SQL delle entità:

AUTORE (Nome, Cognome, DataNascita*, Nazionalità*)

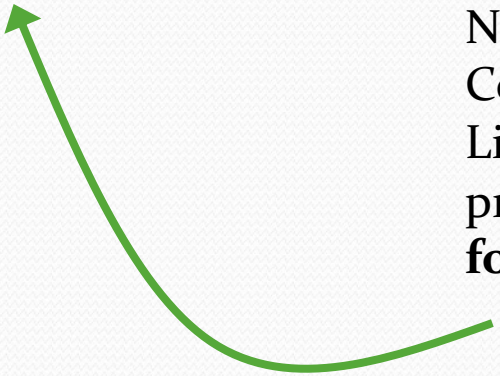
LIBRO (TitoloLibro, NomeAutore↑, CognomeAutore↑, Lingua*)

- Per il vincolo foreign key specificare una politica di cascade sulla cancellazione e di set null sulle modifiche.

Soluzione 3

```
CREATE TABLE AUTORE  
(  
  Nome varchar(20) not null,  
  Cognome varchar(20) not null,  
  DataNascita date,  
  Nazionalità varchar(30),  
  primary key(Nome, Cognome)  
);
```

```
CREATE TABLE LIBRO  
(  
  TitoloLibro varchar(30) not null,  
  NomeAutore varchar(20) ,  
  CognomeAutore varchar(20),  
  Lingua varchar(20),  
  primary key (TitoloLibro),  
  foreign key (NomeAutore, CognomeAutore)  
  references AUTORE(Nome, Cognome)  
  on delete cascade  
  on update set NULL  
);
```



Esempio 4

- Dato lo schema dell'esercizio precedente, spiegare cosa può capitare con l'esecuzione dei seguenti comandi di aggiornamento:
 1. `DELETE FROM AUTORE WHERE Cognome = 'Rossi' ;`
 2. `UPDATE LIBRO SET NomeAutore= 'Umberto' WHERE CognomeAutore = 'Eco';`
 3. `INSERT INTO AUTORE(Nome,Cognome) VALUES('Ugo', 'Bianchi');`
 4. `UPDATE AUTORE SET Nome = 'Italo' WHERE Cognome = 'Calvino';`

Soluzione 4

1. Il comando cancella dalla tabella AUTORE tutte le tuple con Cognome = 'Rossi'. A causa della politica cascade anche le tuple di LIBRO con CognomeAutore = 'Rossi' verranno eliminate.
2. Il comando modifica la tabella LIBRO per tutti i cognomi di autori 'Eco' viene settato il nome.
3. Il comando aggiunge una nuova tupla alla tabella AUTORE. *Non ha alcun effetto sulla tabella LIBRO.*
4. Le tuple di AUTORE con Cognome = 'Calvino' vengono aggiornate a Nome = 'Italo'. A causa della politica set null gli attributi NomeAutore e CognomeAutore delle tuple di LIBRO con CognomeAutore = 'Calvino' vengono posti a NULL.

Esempio 5

- Date le definizioni:

```
CREATE DOMAIN Dominio AS INTEGER DEFAULT 10  
CREATE TABLE Tabella (Attributo Dominio DEFAULT 5);
```

- indicare cosa avviene in seguito ai comandi:
 1. ALTER TABLE Tabella ALTER COLUMN Attributo
DROP DEFAULT;
 2. ALTER DOMAIN Dominio DROP DEFAULT;
 3. DROP DOMAIN Dominio;

Soluzione 5

1. Il comando cancella il valore di default di Attributo. Il valore di default dopo il comando sarà quello impostato in Dominio, ossia 10.
2. Il comando cancella il valore di default di Dominio. Dopo l'operazione il valore di default sarà NULL.
3. Il comando cancella l'intero dominio Domain. In Tabella il dominio di Attributo diventerà INTEGER.

Esercizio

- Dare le definizioni SQL delle relazioni

Rappresentanti(CodRapp, Cognome, Nome, Via, Città, Provincia, TotProvvigioni, PercentProvv)

Clienti (CodCliente, Cognome, Nome, Via, Città, Provincia, Saldo, Fido, CodRapp ↑)

Ordini (NroOrdine, Data, CodCliente ↑)

DettOrdini (NroOrdine↑, NroArt↑, Qta, Prezzo)

Articoli(NroArt, Descrizione, Giacenza, Categoria, PrezzoUnitario)

dimensionando opportunamente gli attributi ed indicando i vincoli sia intrarelazionali che interrelazionali

Query di base in SQL



SQL e gli insiemi

- Occorre fare un'importante distinzione tra SQL ed il modello relazionale formale:
 - SQL consente di avere **più tuple identiche** in tutti gli attributi, quindi in generale una tabella SQL **non è un insieme** di tuple.
È invece un **multiset** (o bag) di tuple.
- Alcune relazioni possono essere vincolate ad essere insiemi, usando il vincolo di chiave oppure l'opzione **DISTINCT** con lo statement SELECT...

SQL, operazioni sui dati

- interrogazione:
 - SELECT
- modifica:
 - INSERT, DELETE, UPDATE

Il comando SELECT

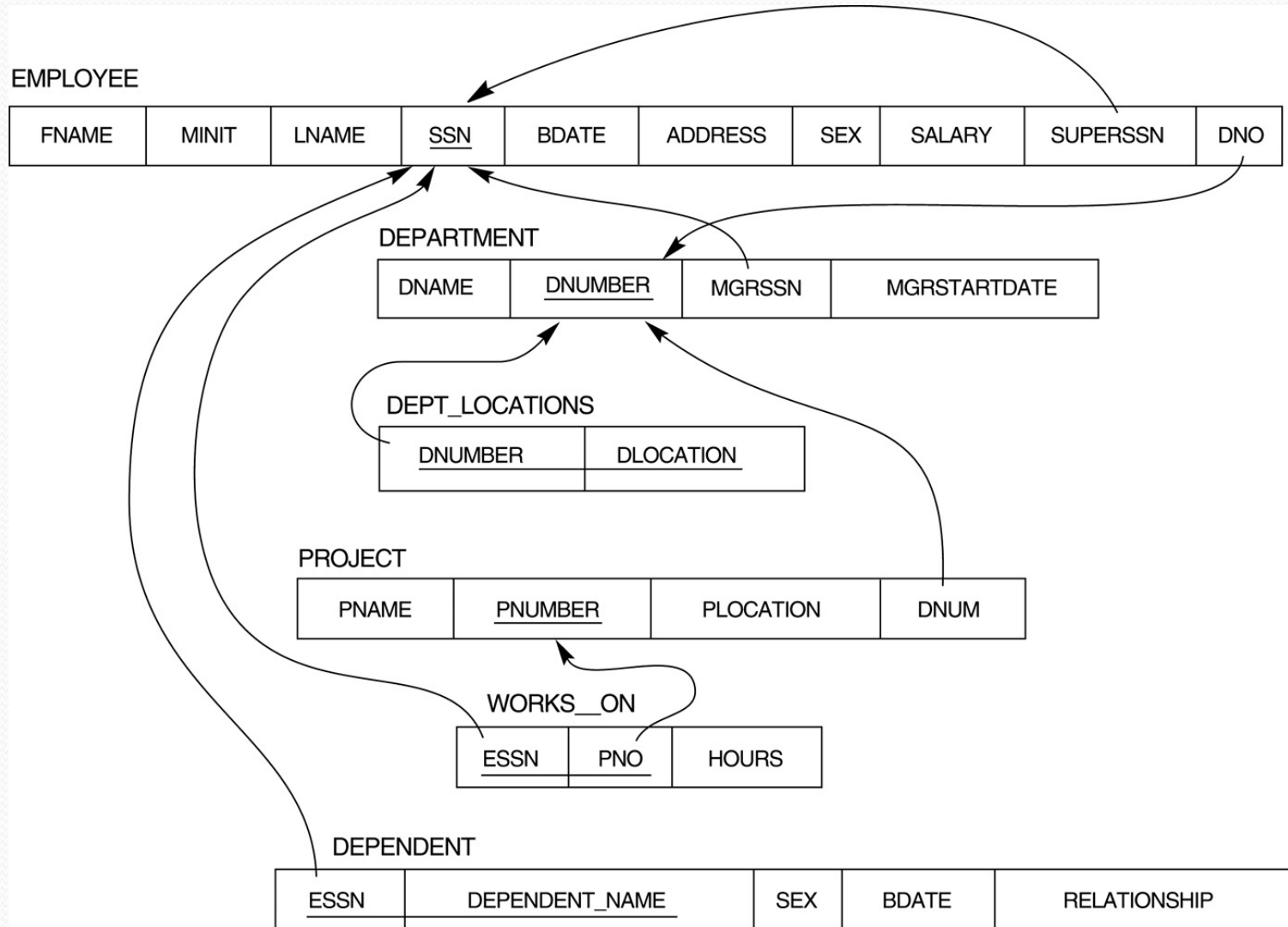
- Il comando **SELECT** è l'istruzione di base per recuperare informazioni da un database.
- Il **SELECT** dell'SQL non ha relazioni con l'operatore di select dell'algebra relazionale.
- La forma di base, detta mapping o blocco di **SELECT FROM WHERE** è formata da tre clausole:

```
SELECT <lista_attributi>  
FROM   <lista_tabelle>  
WHERE <condizione>
```


Il comando SELECT (2)

- *<lista_attributi>* è una lista di nomi di attributi i cui valori devono essere recuperati dalla query.
- *<lista_tabelle>* è una lista di nomi di relazioni richiesti per elaborare la query.
- *<condizione>* è un'espressione booleana di ricerca che identifica la tupla da ritrovare.

Mapping dello schema *Company*



Il comando SELECT: *Esempio*

- Trovare la data di nascita e l'indirizzo dell'impiegato di nome *John B. Smith*:

```
SELECT  BDATE, ADDRESS  
FROM    EMPLOYEE  
WHERE   FNAME='JOHN' AND MINIT='B' AND  
        LNAME='SMITH';
```

- BDATE e ADDRESS sono detti anche *Attributi di proiezione*.
- Equivalente nell'algebra relazionale:
$$\pi_{\langle \text{BDATE}, \text{ADDRESS} \rangle} (\sigma_{\text{FNAME}='JOHN' \text{ AND } \text{MINIT}='B' \text{ AND } \text{LNAME}='SMITH'} (\text{EMPLOYEE}))$$

Il comando SELECT: *Esempio (2)*

- Trovare cognome, nome e indirizzo di tutti gli impiegati del dipartimento *'Research'*:

```
SELECT  FNAME, LNAME, ADDRESS
FROM    EMPLOYEE, DEPARTMENT
WHERE   DNAME='Research' AND DNUMBER=DNO;
```

- È simile alla sequenza SELECT-PROJECT-JOIN dell'algebra relazionale, ed è perciò detta query *select-project-join*.
- Equivalente nell'algebra relazionale:

$$\pi_{\langle \text{FNAME}, \text{LNAME}, \text{ADDRESS} \rangle} (\sigma_{\text{DNAME}='Research'} (\text{EMPLOYEE} \bowtie \text{DEPARTMENT}))$$

DNO=DNUMBER

Maternità

Madre	Figlio
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

Paternità

Padre	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

Persone

Nome	Età	Reddito
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	30
Luigi	50	40
Franco	60	20
Olga	30	41
Sergio	85	35
Luisa	75	87

Selezione e proiezione

- Nome e reddito delle persone con gli anni minore o uguale a trenta:

$$\pi_{\langle \text{nome}, \text{reddito} \rangle}(\sigma_{\text{eta} \leq 30}(\text{Persone}))$$

```
SELECT nome, reddito  
FROM persone  
WHERE eta <= 30
```

- Abbreviazione di:

```
SELECT p.nome AS nome, p.reddito AS reddito  
FROM persone p  
WHERE p.eta <= 30
```


Persone

Nome	Reddito
Andrea	21
Aldo	15
Filippo	30

Selezione, proiezione e join

- Istruzioni SELECT con una sola relazione nella clausola FROM permettono di realizzare:
 - selezioni, proiezioni, ridenominazioni.
- Con più relazioni nella FROM si realizzano join (e prodotti cartesiani).

SQL e algebra relazionale

- $R_1(A_1, A_2) \bowtie R_2(A_3, A_4)$

```
SELECT R1.A1, R2.A4  
FROM   R1, R2  
WHERE  R1.A2 = R2.A3
```

- proiezione (**SELECT**)
- prodotto cartesiano (**FROM**)
- selezione (**WHERE**)

SQL e algebra relazionale (2)

- Consideriamo gli schemi $R_1(A_1, A_2)$ $R_2(A_3, A_4)$:

```
SELECT R1.A1, R2.A4  
FROM   R1, R2  
WHERE  R1.A2 = R2.A3
```

$$\pi_{\langle A_1, A_4 \rangle} (\sigma_{A_2=A_3} (R_1 \times R_2))$$

Proiezione senza selezione

- Nome e reddito di tutte le persone :

$\pi_{\langle \text{nome}, \text{reddito} \rangle}(\text{Persone})$

SELECT nome, reddito
FROM persone

- Abbreviazione di:

SELECT p.nome AS nome, p.reddito AS reddito
FROM persone p

Abbreviazioni

- Dato uno schema $R(A,B)$; tutti gli attributi di R :

$$\pi_{A, B}(R)$$

SELECT *
FROM R

- equivale (intuitivamente) a:

SELECT X.A AS A, X.B AS B
FROM R X
WHERE TRUE

Esempio

- I padri di persone che guadagnano più di 22:

$\pi_{\text{Padre}} (\text{Paternita} \bowtie_{\text{Figlio=Nome}} \sigma_{\text{Reddito} > 22} (\text{Persone}))$

```
SELECT  DISTINCT Padre
FROM    Persone, Paternita
WHERE   Figlio = Nome AND Reddito > 22
```

Padre

Luigi

Luigi

Padre

Luigi

Il comando SELECT: *Esempio*

- Per ogni progetto localizzato a '*Stafford*', listare il n° di progetto, il n° di dipartimento di controllo, ed il cognome, l'indirizzo e la data di nascita del manager del dipartimento:

```
SELECT  PNUMBER, DNUM, LNAME, ADDRESS, BDATE
FROM    PROJECT, DEPARTMENT, EMPLOYEE
WHERE    DNUM=DNUMBER AND MGRSSN=SSN AND
           PLOCATION='Stafford'
```


Nomi di attributi e Renaming

- In SQL lo stesso nome può essere usato per più attributi solo se questi appartengono a relazioni diverse.
- Se una query coinvolge tali relazioni, occorre **qualificare** il nome dell'attributo con il nome della relazione per evitare ambiguità.
- ***Esempio:*** Employee.SSN

Nomi di attributi e Renaming (2)

- Si può avere ambiguità anche nel caso di query che riferiscono due volte alla stessa relazione:
 - *Esempio:* Per ogni impiegato, trovare il suo nome il suo cognome e quello del suo diretto superiore:

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME  
FROM EMPLOYEE AS E, EMPLOYEE AS S  
WHERE E.SUPERSSN=S.SSN;
```
 - Abbiamo dichiarato nomi di relazione alternativi **E** ed **S**, detti **alias**, per la relazione EMPLOYEE .

Nomi di attributi e Renaming (3)

- È anche possibile rinominare gli attributi della relazione nella query, dando loro degli alias scrivendo:

EMPLOYEE **AS** E(FN, MI, LN, SSN, BD, ADDR, SEX, SAL, SSSN, DNO)

nella clausola **FROM**.

- *Quella che abbiamo visto è un esempio di query ricorsiva.*

Manca del WHERE

- Omettere la clausola WHERE equivale a **WHERE TRUE**, cioè tutte le tuple della relazione specificata nella clausola FROM fanno parte del risultato.
- Se più di una relazione è specificata nella clausola **FROM**, allora il risultato sarà il *prodotto cartesiano* delle relazioni.

Manca del WHERE: Esempi

- **Esempio:** Selezionare tutti i SSN:

```
SELECT SSN  
FROM EMPLOYEE;
```

- **Esempio:** Selezionare tutte le combinazioni Employee.SSN e Department.Dname:

```
SELECT SSN, DNAME  
FROM EMPLOYEE, DEPARTMENT;
```

Il carattere jolly “*”

- Per recuperare tutti gli attributi delle tuple selezionate, si usa il carattere jolly *:
 - **Esempio:** Trovare tutti i valori degli attributi degli impiegati che lavorano per il dipartimento n° 5:
- **Esempio:** Trovare tutti gli attributi di Employee e gli attributi di Department per cui lavora ogni impiegato del dipartimento *‘Research’*:

```
SELECT *  
FROM EMPLOYEE  
WHERE DNO=5;
```

```
SELECT *  
FROM EMPLOYEE, DEPARTMENT  
WHERE DNAME='Research' AND DNO=DNUMBER;
```


Duplicazioni di tuple in SQL

- SQL non tratta relazioni come insiemi: *tuple duplicate possono apparire più di una volta.*
- SQL non elimina le duplicazioni per le seguenti ragioni:
 - è un'operazione costosa (l'implementazione richiederebbe l'ordinamento e poi l'eliminazione);
 - l'utente può essere interessato alle duplicazioni;
 - con funzioni di aggregazione siamo interessati a non eliminarle.

La clausola DISTINCT

- Se le duplicazioni non sono volute, lo si specifica con la clausola **DISTINCT**:
- *Esempi:*
 - Trovare i salari di tutti gli impiegati:
**SELECT SALARY
FROM EMPLOYEE;**
 - Trovare i salari distinti degli impiegati:
**SELECT DISTINCT SALARY
FROM EMPLOYEE;**

Distinct, *attenzione*

- **cognome e filiale** di tutti gli impiegati

Cognome	Filiale
Neri	Napoli
Neri	Milano
Rossi	Roma

π <Cognome, Filiale> (Impiegati)

SELECT

Cognome, Filiale

FROM Impiegati

Cognome	Filiale
Neri	Napoli
Neri	Milano
Rossi	Roma
Rossi	Roma

SELECT DISTINCT

Cognome, Filiale

FROM Impiegati

Cognome	Filiale
Neri	Napoli
Neri	Milano
Rossi	Roma

Operazioni insiemistiche

- SQL incorpora le seguenti operazioni insiemistiche (*è richiesta la union-compatibilità*):
 - UNION
 - EXCEPT
 - INTERSECT } SQL 2
- **EXCEPT** restituisce tutti i valori distinti della query a sinistra dell'operando non presenti nella query a destra.
- Usando tali operazioni le tuple duplicate sono eliminate (a meno che non venga richiesto il contrario con la clausola **ALL**).

Operazioni insiemistiche - *Esempio*

- Fare una lista dei numeri di progetti per i progetti che coinvolgono un impiegato il cui cognome è 'Smith' come lavoratore **oppure** come manager del dipartimento che controlla il progetto:

```
(SELECT PNUMBER
  FROM   PROJECT, WORKS_ON, EMPLOYEE
 WHERE  PNUMBER=PNO AND ESSN=SSN AND
        LNAME='Smith');
```

UNION

```
(SELECT PNUMBER
  FROM   PROJECT, DEPARTMENT, EMPLOYEE
 WHERE  DNUM=DNUMBER AND MGRSSN=SSN AND
        LNAME='Smith')
```


Confronto tra sottostringhe

- Per il confronto tra stringhe si usa l'operatore **LIKE**
- *Caratteri jolly:*
 - '%' rimpiazza qualsiasi numero di caratteri;
 - '_' rimpiazza un singolo carattere;
- **Esempio:** Trovare tutti gli impiegati il cui indirizzo è a *'Houston, Texas'*:

```
SELECT FNAME, LNAME  
FROM EMPLOYEE  
WHERE ADDRESS LIKE '%HOUSTON, TEXAS%';
```

“LIKE” - *Esempio*

- Le persone che hanno un nome che inizia per 'A' e ha una 'd' come terza lettera:

```
SELECT *  
FROM persone  
WHERE nome LIKE 'A_d%';
```


Confronto tra sottostringhe - *Esempi*

- Trovare tutti gli impiegati nati negli anni '50. Il formato di data è *YYYY-MM-DD*:

```
SELECT FNAME, LNAME  
FROM EMPLOYEE  
WHERE BDATE LIKE '__ 5__ __ __ __ __ __';
```

- Mostrare i salari risultanti se a tutti gli impiegati che lavorano sul progetto '*Product X*' viene concesso un aumento del 10%:

```
SELECT FNAME, LNAME, 1.1*SALARY  
FROM EMPLOYEE, WORKS_ON, PROJECT  
WHERE ESSN=SSN AND PNO=PNUMBER AND  
PNAME='Product X';
```

Gestione dei valori nulli

Impiegati

Matricola	Cognome	Filiale	Età
5998	Neri	Milano	45
9553	Bruni	Milano	NULL

- Gli impiegati la cui età è NULL o potrebbe essere maggiore di 40.

σ $\text{Età} > 40 \text{ OR Et\`a IS NULL}$ (Impiegati)

Gestione dei valori nulli (2)

- Gli impiegati la cui età è NULL o potrebbe essere maggiore di 40.

$\sigma_{\text{Età} > 40 \text{ OR Et\`a IS NULL}}$ (Impiegati)

SELECT *

FROM Impiegati

WHERE eta > 40 **OR** eta **IS NULL**

Query più complesse in SQL

Query annidate e confronto di insiemi

- La query usata per mostrare la **UNION** può essere così riformulata:

```
SELECT DISTINCT PNUMBER  
FROM PROJECT  
WHERE PNUMBER IN  
      (SELECT PNUMBER  
       FROM PROJECT, DEPARTMENT, EMPLOYEE  
       WHERE DNUM=DNUMBER AND MGRSSN=SSN  
         AND LNAME= 'Smith')
```

OR

```
PNUMBER IN  
      (SELECT PNO  
       FROM WORKS_ON, EMPLOYEE  
       WHERE ESSN=SSN AND LNAME='Smith');
```

Queries annidate e confronto di insiemi (2)

- La prima query seleziona il numero dei progetti che hanno uno *'Smith'* come manager, mentre la seconda seleziona il numero di progetto dei progetti che hanno uno *'Smith'* come lavoratore.
- Nella query esterna selezioniamo una tupla PROJECT se il valore PNUMBER compare nel risultato di una delle query annidate.

L'operatore IN

- L'operatore **IN** confronta un valore con un insieme di tuple union-compatibili.
- L'operatore **IN** permette di specificare valori multipli nella clausola **WHERE**.

- ***Esempio:***

```
SELECT DISTINCT ESSN
FROM   WORKS_ON
WHERE  (PNO, HOURS) IN
        (SELECT PNO, HOURS
         FROM WORKS_ON
         WHERE ESSN='123456789');
```

Altri operatori

- **ANY** (o **SOME**)
 - confronta un singolo valore (attributo) v con un multiset V , restituendo TRUE se v è uguale a qualche valore in V .
 - **ANY** e **SOME** sono equivalenti. Possono essere combinati con $\{>, \geq, <, \leq, <>\}$.
 - $=$ **ANY** è equivalente ad usare l'operatore **IN**.
- Anche **ALL** può essere combinato con questi operatori.
 - $v > \text{ALL } V$ è TRUE se il valore v è maggiore di tutti i valori in V .

Operatore *ALL* - *Esempio*

- Trovare tutti i nomi degli impiegati il cui salario è maggiore del salario di tutti gli impiegati del dipartimento 5:

```
SELECT LNAME, ENAME  
FROM   EMPLOYEE  
WHERE  SALARY > ALL (SELECT SALARY  
                     FROM   EMPLOYEE  
                     WHERE DNO=5);
```

Operatore ANY - *Esempio*

```
SELECT LNAME, ENAME  
FROM EMPLOYEE  
WHERE SALARY > ANY (SELECT SALARY  
                        FROM EMPLOYEE);
```

- Trovare tutti i nomi degli impiegati tranne quelli il con il salario minimo.

Casi di ambiguità

- Ambiguità nei nomi di attributi:
 - Si ha, se esistono attributi con lo stesso nome, uno in una relazione nella clausola FROM della query esterna e l'altro in una relazione della clausola FROM della query interna.
- **Regola:** un riferimento a un attributo non qualificato riferisce alla relazione dichiarata nella query annidata più interna.

Casi di ambiguità - *Esempio*

- Trovare il nome di ogni impiegato che ha una persona a carico con lo stesso nome e lo stesso sesso dell'impiegato:

```
SELECT E.ENAME, E.LNAME
FROM EMPLOYEE AS E
WHERE E.SSN IN ( SELECT ESSN
                  FROM DEPENDENT
                  WHERE ESSN=E.SSN
                  AND DEPENDENT_NAME = E.FNAME
                  AND SEX=E.SEX);
```

È necessario qualificarlo altrimenti farebbe riferimento alla relazione DEPENDENT.

Casi di ambiguità (2)

- In generale, una query scritta con blocchi annidati **SELECT...FROM...WHERE** e con operatori di confronto = o **IN** può essere sempre espressa come un singolo blocco.
- La precedente query può anche essere scritta come:

```
SELECT E.FNAME, E.LNAME  
FROM EMPLOYEE AS E, DEPENDENT AS D  
WHERE E.SSN=D.SSN AND E.SEX=D.SEX AND  
E.FNAME=D.DEPENDENT_NAME
```

L'operatore CONTAINS

- L'implementazione originale SQL su **systemR** prevedeva un operatore **CONTAINS** per confrontare due insiemi.
- È stato poi eliminato per motivi di efficienza.

L'operatore CONTAINS - *Esempio*

- Ritrovare il nome e cognome di ciascun impiegato che lavora su tutti i progetti controllati dal dipartimento 5:

```
SELECT FNAME, LNAME
FROM EMPLOYEE AS E
WHERE (( SELECT PNO
          FROM WORKS_ON
          WHERE E.SSN=ESSN)
CONTAINS
( SELECT PNUMBER
  FROM PROJECT
  WHERE DNUM=5));
```

EXIST e NOT EXIST

- **EXISTS e NOT EXISTS:**
 - Per verificare se il risultato di una query annidata correlata è vuota.
- ***Esempio:*** Ritrovare il nome degli impiegati che non hanno persone a carico:

```
SELECT FNAME, LNAME  
FROM EMPLOYEE  
WHERE NOT EXISTS ( SELECT *  
                    FROM DEPENDENT  
                    WHERE SSN=ESSN);
```


Insiemi espliciti

- Trovare il SSN di tutti gli impiegati che lavorano sui progetti 1, 2 o 3:

```
SELECT DISTINCT ESSN  
FROM WORKS_ON  
WHERE PNO IN (1, 2, 3);
```

- Si può anche testare se un valore è **NULL**:
 - $=$ e \neq sono scritti come *'IS'* e *'IS NOT'* per confronti con *NULL*.
 - **Esempio:** Trovare il nome di tutti gli impiegati che non hanno supervisori:

```
SELECT FNAME, LNAME  
FROM EMPLOYEE  
WHERE SUPERSSN IS NULL;
```

La keyword AS

- È possibile rinominare qualsiasi attributo che compare in una query con la keyword **AS**:

```
SELECT E.LNAME AS EMPLOYEE_NAME,  
       S.LNAME AS SUPERVISOR_NAME  
FROM EMPLOYEE AS E, EMPLOYEE AS S  
WHERE E.SUPERSSN=S.SSN;
```


Tabelle Joined

- Il concetto di tabella joined compare con SQL2 per specificare un'operazione di **JOIN** nella clausola **FROM**.
- Possono essere usati i seguenti tipi di join:
 - **NATURAL JOIN**
 - **INNER JOIN**
 - **LEFT OUTER JOIN / RIGHT OUTER JOIN**
 - **FULL OUTER JOIN**

Tabelle Joined - *Esempio*

- Trovare il nome e l'indirizzo di ogni impiegato che lavora per il Dipartimento *'Research'*:

```
SELECT FNAME, LNAME, ADDRESS  
FROM (EMPLOYEE JOIN DEPARTMENT ON  
      DNO=DNUMBER)  
WHERE DNAME='Research';
```


Aggregazione e raggruppamento

- Le funzioni di aggregazione e di raggruppamento sono diffusissime nella gestione di basi di dati. SQL incorpora le seguenti funzioni:
 - **COUNT**: conteggio tuple.
 - **COUNT(DISTINCT ...)**: conteggio di tuple distinte.
 - **SUM**: somma dei valori di un attributo in una tabella.
 - **MAX**: valore massimo tra gli attributi di una tabella.
 - **MIN**: valore minimo tra gli attributi di una tabella.
 - **AVG**: valore medio tra gli attributi di una tabella.
 - **STD**: deviazione standard tra gli attributi di una tabella.

Aggregazione e raggruppamento

- **Esempio:** Trovare la somma dei salari di tutti gli impiegati, il massimo, il minimo e la media dei salari:

```
SELECT SUM(SALARY), MAX(SALARY),  
        MIN(SALARY), AVG(SALARY)  
FROM EMPLOYEE;
```


Count - *Esempi*

- Conta il numero di impiegati:

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

- Restituisce il numero di tuple nel risultato della query (*):

```
SELECT COUNT(*) AS Conteggio  
FROM EMPLOYEE, DEPARTMENT  
WHERE DNO=DNUMBER AND DNAME='Research';
```

Count - *Esempi* (2)

- Conta il numero di valori di stipendi distinti:
SELECT COUNT (DISTINCT SALARY)
FROM EMPLOYEE;
- Elencare il nome ed il cognome degli impiegati che hanno due o più persone a carico:

```
SELECT LNAME, FNAME  
FROM EMPLOYEE  
WHERE ( SELECT COUNT(*)  
        FROM DEPENDENT  
        WHERE SSN=ESSN)>=2;
```


Ordinamento di tuple

- Per ordinare le tuple nel risultato della query si usa la clausola **ORDER BY**.
- *Esempio:* Ritrovare una lista di impiegati e dei progetti su cui lavorano, ordinati per dipartimento, e nell'ambito di ciascun dipartimento, alfabeticamente per cognome e nome:

```
SELECT DNAME, FNAME, LNAME, PNAME  
FROM DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT  
WHERE DNUMBER=DNO AND SSN=ESSN AND  
      PNO=PNUMBER  
ORDER BY DNAME, LNAME, FNAME;
```

Ordinamento di tuple (2)

- L'ordine di default è crescente:
 - **ASC** per crescente.
 - **DESC** decrescente.
- ***Esempio:*** per avere un ordine decrescente di dipartimento e crescente per nome e cognome:

...

```
ORDER BY DNAME DESC, LNAME ASC, FNAME ASC;
```


Group by

- Raggruppiamo le tuple che hanno lo stesso valore per alcuni attributi.
- ***Esempio:***
SELECT DNO, COUNT(*), AVG(SALARY)
FROM EMPLOYEE
GROUP BY DNO;
- Le tuple sono divise in gruppi, ogni gruppo ha lo stesso valore per DNO.
- Le funzioni COUNT e AVG sono applicate ad ogni gruppo di queste tuple.

Group by (2)

- Risultato:

DNO	COUNT(*)	AVG(SALARY)
1	4	23000
4	3	25000
3	4	22000

Group by (3)

- **Esempio:** Per ogni progetto, visualizzare il numero del progetto, il nome del progetto ed il numero di impiegati che lavorano su quel progetto:

```
SELECT pnumber, pname, COUNT(*)  
FROM project, works_on  
WHERE pnumber = pno  
GROUP BY pnumber;
```

Group by (4)

- **Esempio:** Per ogni progetto visualizzare il numero del progetto, il nome del progetto ed il numero di impiegati del dipartimento n.5 che lavorano su quel progetto:

```
SELECT pnumber, pname, COUNT(*)  
FROM project, works_on, employee  
WHERE pnumber = pno AND ssn = essn AND dno = 5  
GROUP BY pnumber, pname;
```


Group by (5) – uso di HAVING

- **Esempio:** Per ogni progetto su cui lavorano più di due impiegati, visualizzare il numero del progetto, il nome del progetto ed il numero di impiegati che lavorano su quel progetto:

```
SELECT pnumber, pname, COUNT(*)  
FROM project, works_on  
WHERE pnumber = pno  
GROUP BY pnumber, pname  
HAVING COUNT(*) > 2;
```

Group by (6) – uso di HAVING (2)

- **Esempio:** Determinare, per ogni dipartimento che ha più di 6 impiegati, il numero totale degli impiegati il cui stipendio è maggiore di \$40.000:

```
SELECT dname, COUNT(*)  
FROM department, employee  
WHERE dnumber = dno AND salary > 40000  
GROUP BY dname  
HAVING COUNT(*) > 6;
```


Riepilogo delle interrogazioni

SELECT *<elenco attributi e funzioni>*

FROM *<elenco delle tabelle>*

[WHERE *<condizioni>* **]**

[GROUP BY *<attributo o attributi di raggruppamento>* **]**

[HAVING *<condizione di raggruppamento>* **]**

[ORDER BY *<elenco attributi>* **]**

Aggiornamenti in SQL

Aggiornamenti in SQL

- In SQL sono previsti tre comandi per modificare il database:
 - INSERT
 - DELETE
 - UPDATE

Il comando Insert

- Il comando **INSERT INTO** inserisce nuove righe in una relazione.

- Sintassi:

```
INSERT INTO Target [(FieldName,...)]  
VALUES (Value1,...);
```

oppure

```
INSERT INTO Target [(FieldName,...)]  
SELECT FieldName,...  
FROM TableExpression;
```


Il comando Insert - *Esempio*

- Aggiungere una nuova tupla alla relazione *'Employee'*:

```
INSERT INTO EMPLOYEE  
VALUES ('Richard', 'K', 'Marini', '654765876',  
'30-DEC-52', '98 Oak Forest, Katy, TX', 'M',  
37000, '987654321', 4);
```

Il comando Insert (2)

- È possibile non assegnare valori a tutti gli attributi.
- In tal caso, questi avranno il valore di **default** o **NULL**.
- *Esempio:*
 - **INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)**
VALUES ('Richard', 'Marini', '654765876');

Il comando Insert - *Esempio*

- Creare una tabella temporanea che ha nome, numero di impiegati e salari totali per ciascun dipartimento:

```
CREATE TABLE DEPTS_INFO ( DEPT_NAME VARCHAR(15),  
                           NO_OF_EMPS INTEGER,  
                           TOTAL_SAL INTEGER);  
INSERT INTO DEPTS_INFO (DEPT_NAME, NO_OF_EMPS,  
                        TOTAL_SAL)  
SELECT DNAME, COUNT(*), SUM(SALARY)  
FROM DEPARTMENT, EMPLOYEE  
WHERE DNUMBER=DNO  
GROUP BY DNAME;
```

- *Eventuali aggiornamenti successivi non influenzano la tabella originale. Per aggiornarle, è invece necessario definire una **view**.*

Uso di *auto_increment*

- L' attributo **AUTO_INCREMENT** può essere usato per generare un identificatore unico per le nuove righe:

```
CREATE TABLE animals (  
  id INT NOT NULL AUTO_INCREMENT,  
  name CHAR(30) NOT NULL,  
  PRIMARY KEY (id)  
) AUTO_INCREMENT=5;
```

```
INSERT INTO animals (name) VALUES  
( 'dog'), ('cat'), ('penguin'), ('wolf'), ('whale'), ('ostrich');
```

```
SELECT * FROM animals;
```

id	name
5	dog
6	cat
7	penguin
8	wolf
9	whale
10	ostrick

Il comando DELETE

- Il comando **DELETE** rimuove una o più tuple da una relazione.

- Sintassi:

```
DELETE  
FROM TableName  
WHERE Criteria;
```

Il comando DELETE - *Esempi*

```
DELETE FROM EMPLOYEE  
WHERE LNAME='Brown';
```

```
DELETE FROM EMPLOYEE  
WHERE DNO IN (SELECT DNUMBER  
               FROM DEPARTMENT  
               WHERE DNAME='Research');
```


Il comando UPDATE

- Il comando **UPDATE** permette di modificare valori in una relazione:

```
UPDATE PROJECT  
SET PLOCATION='Bellaire', DNUM=5  
WHERE PNUMBER=10;
```

```
UPDATE EMPLOYEE  
SET SALARY=SALARY * 1.1  
WHERE DNO IN (SELECT DNUMBER  
               FROM DEPARTMENT  
               WHERE DNAME='Research');
```

Viste in SQL

- Le viste sono tabelle *'virtuali'* derivate da tabelle esistenti nel db.
- Possono essere definite per nascondere dei dati da alcune tabelle (*es.* per questioni di privacy), per combinare più tabelle, per creare report, etc.
- Sintassi:

```
CREATE VIEW ViewName  
AS SelectStatement;
```


Viste in SQL - *Esempio*

```
CREATE VIEW WORKS_ON1  
AS      SELECT FNAME, LNAME, PNAME, HOURS  
          FROM EMPLOYEE, PROJECT, WORKS_ON  
          WHERE SSN=ESSN AND PNO=PNUMBER;
```

WORKS_ON1

FNAME	LNAME	PNAME	HOURS
-------	-------	-------	-------

Viste in SQL - *Esempio* (2)

```
CREATE VIEW DEPTS_INFO (DEPT_NAME,  
                        NO_OF_EMPS, TOTAL_SAL)  
SELECT DNAME, COUNT(*), SUM(SALARY)  
FROM DEPARTMENT, EMPLOYEE  
WHERE DNUMBER=DNO  
GROUP BY DNAME;
```

DEPT_INFO

DEPT_NAME	NO_OF_EMPS	TOTAL_SAL
-----------	------------	-----------

Indici in SQL

- Per creare indici si usa il comando **CREATE INDEX**
- Sintassi:
 - **CREATE INDEX** *IndexName* **ON** *TableName*(*AttributeName*);
- Il DBMS crea dei path di accesso predefiniti per accedere più velocemente agli attributi indicizzati.

Indici in SQL - *Esempi*

```
CREATE INDEX LNAME_INDEX  
ON EMPLOYEE(LNAME);
```

```
CREATE INDEX NAME_INDEX  
ON EMPLOYEE(LNAME ASC, FNAME DESC, MINIT);
```

- Per specificare un vincolo di chiave su un attributo:

```
CREATE UNIQUE INDEX SSN_INDEX  
ON EMPLOYEE(SSN)
```