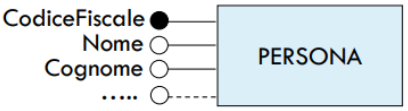
BASI DI DATI

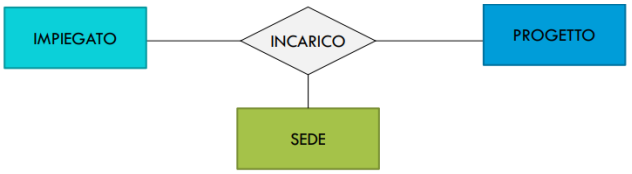
Modello E/R

- Entità da rappresentare con rettangoli e attributo chiave da riempire il pallino
- Usare SOLO il singolare



- Associazioni si scrivono come un rombo e possono essere:
  - uno a uno
  - uno a molti
  - molti a molti
- Usare sostantivo e NON verbi tipo: Residenza e non Risiede

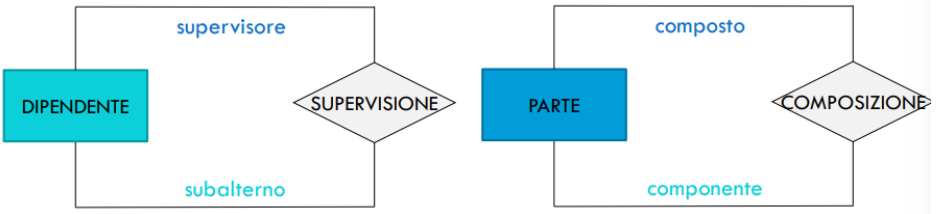
- Grado di un'associazione è il numero di entità coinvolte in un'associazione



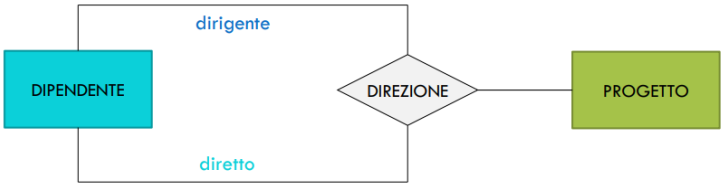
grado 3

- E' possibile avere più associazioni tra le stesse entità
- Un'associazione può essere ad anello:
  -

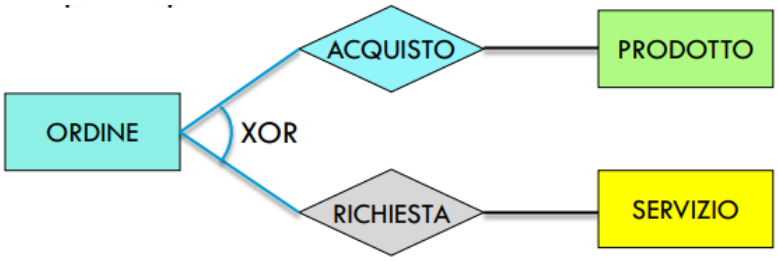
  - Simmetrica
  - Riflessiva
  - Transitiva
    - Nelle associazioni ad anello non simmetriche bisogna specificare il ruolo di ogni ramo



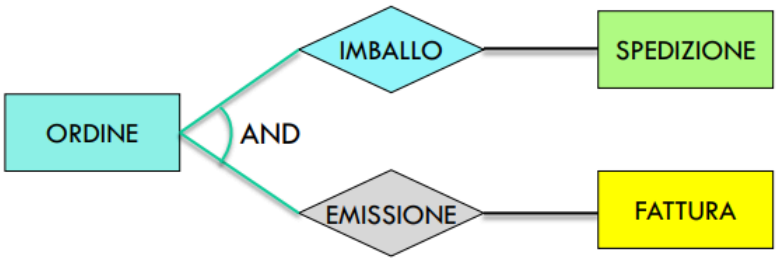
- È possibile avere anelli anche in relazioni n-arie generiche (n > 2):



- Associazioni possono essere anche XOR/AND

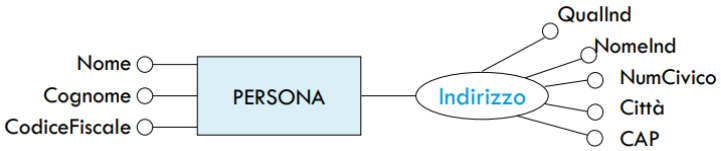


Un ordine riguarda o l'acquisto di un prodotto o la richiesta di un servizio, ma non entrambe le cose.

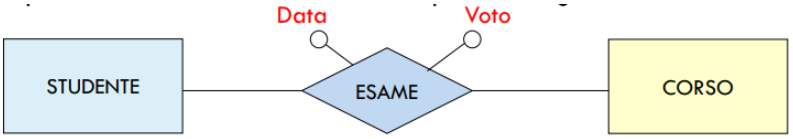


Ogni volta che si emette fattura a fronte di un ordine deve esserci anche il confezionamento dell'imballo per la spedizione di quanto ordinato.

- Attributi possono essere composti

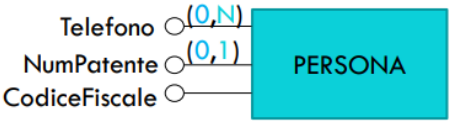


- Attributi possono essere assegnati alle associazioni. Se attributi non appartengono a entità gli assegno alle associazioni..



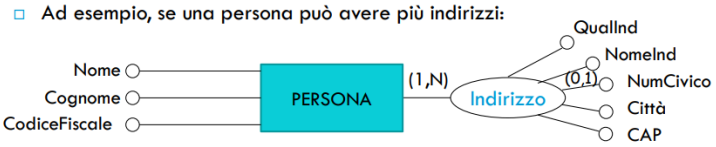
- Si può specificare la cardinalità degli attributi di un'entità

In assenza d'indicazione il valore di default è (1,1).

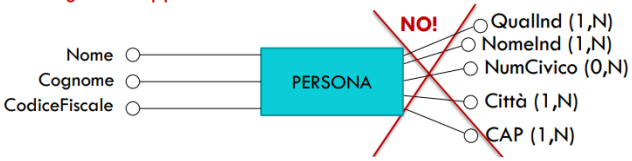


- Attributo può essere:
  - Opzionale: se cardinalità minima è 0 (NumPatente)
  - Monovalore: se cardinalità massima è 1 (CodFiscale)
  - Multivalore: se cardinalità massima è N (Telefono)

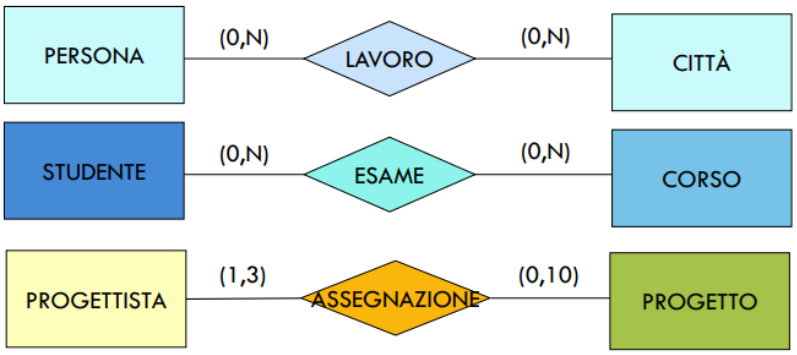
- Ad esempio, se una persona può avere più indirizzi:



- La seguente rappresentazione non sarebbe infatti corretta!

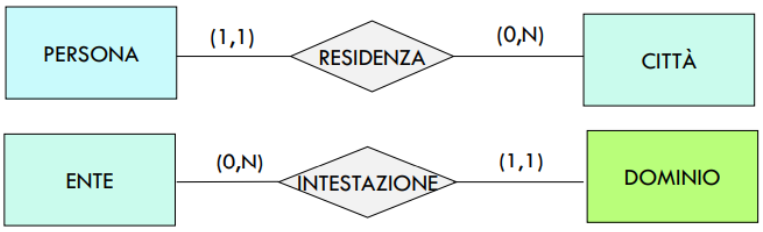


- Esempio di associazione molti a molti, perchè cardinalità massima è N



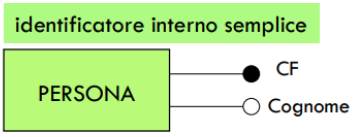
- Esempio di associazione uno a uno, perché cardinalità massima è 1

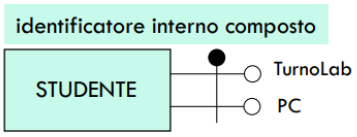
- Esempio di associazione uno a molti, perchè cardinalità massima da un lato è 1 e dall'altro è N

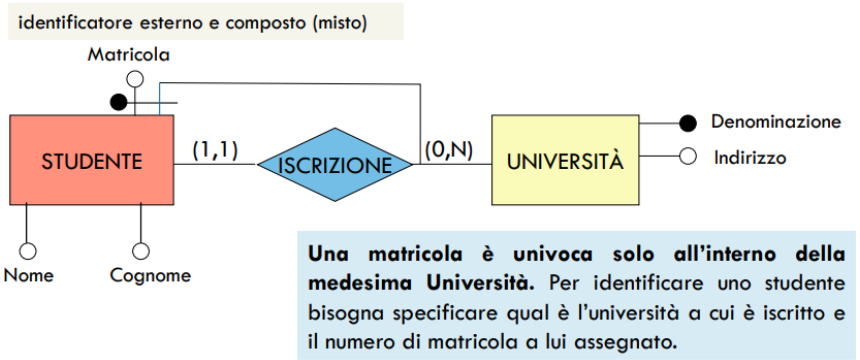


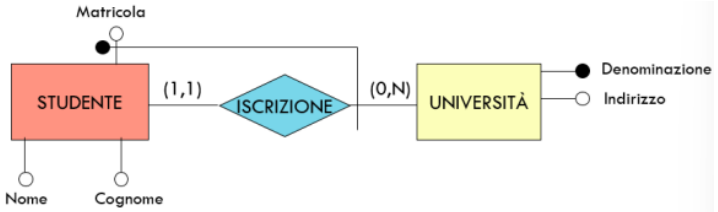
Si legge "Una persona risiede in una città e in una città possono risiedere più persone"
Si guarda il numero a destra nelle parentesi più vicine all'entità a cui ci riferiamo (max-card) e in base a quello diciamo quante istanze dell'altra entità hanno quell'associazione

- Identificatori possono essere:
  - interno semplice: una sola chiave
  - interno composto: più chiavi della stessa entità contemporaneamente
  - esterno composto (misto): nell'esempio la chiave è composta dalla matricola e dalla denominazione





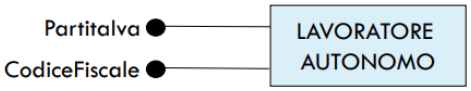




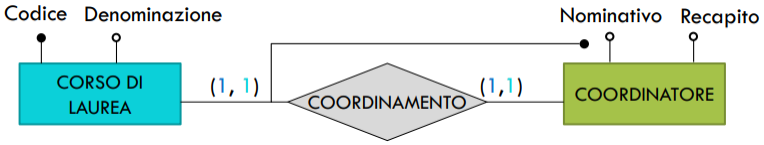
usare questo disegno più conciso

Da notare la cardinalità di uno dei due è 1 a 1 !!!

- Un'entità può avere più chiavi univoche



- Un'entità può essere debole, cioè se ha come identificatore solo quello di un'altra entità, quando l'associazione è 1 a 1+



Modello Relazionale

- Una tabella ammette duplicati mentre una relazione no

STUDENTI

Matricola	Cognome	Nome	DataNascita
2106103423	Bianchi	Giorgio	21/06/1978
2106111021	Rossi	Anna	13/04/1978
1602042312	Rossi	Anna	11/03/1978

SELECT Cognome, Nome

FROM STUDENTI ;

una tabella che non è una relazione

Cognome	Nome
Bianchi	Giorgio
Rossi	Anna
Rossi	Anna

SELECT DISTINCT Cognome, Nome

FROM STUDENTI ;

una tabella che è una relazione

Cognome	Nome
Bianchi	Giorgio
Rossi	Anna

- 1NF (Prima Forma Normale): ogni dominio di una relazione deve essere atomico (eccezione per le date, stringhe ecc…), ossia non ulteriormente decomponibili
- In una relazione se al posto di usare caratteri speciali per dire per esempio che un valore non è presente si usa NULL

STUDENTI

Matricola	CodiceFiscale	Cognome	Nome	DataNascita
210629323	BNCGRG78L21A944Z	Bianchi	Giorgio	21/07/1978
216635467	RSSNNA78A53A944N	Rossi	Anna	13/01/1978
160239654	VRDMRC79H20F839U	Verdi	Marco	20/06/1979
214842132	VRDMRC79H20G125T	Verdi	Marco	20/06/1979

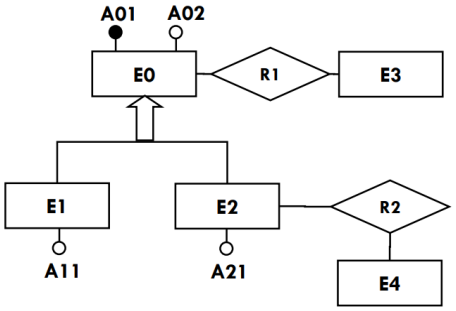
- {Matricola} e {Codice Fiscale} sono due chiavi
- {Matricola, Cognome} e {CodiceFiscale, Nome} sono solo superchiavi
- {Cognome, Nome, Data Nascita} non è superchiave

- La Primary Key si sottolinea (tipo <u>Matricola</u>) e non ammette NULL

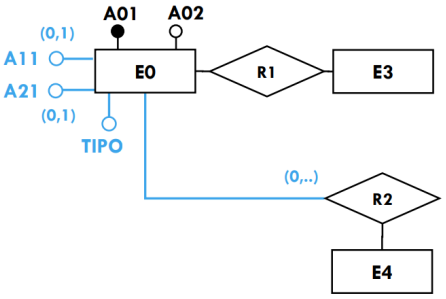
Progettazione Logica

- Si effettua la ristrutturazione (eliminazione dei costrutti che non possono essere rappresentati nel modello logico):
  - Eliminazione degli attributi multivalore
  - Eliminazione delle gerarchie di generalizzazione
  - Accorpamento o partizionamento di entità/associazioni
  - Scelta Identificatori principali
- 3 possibili modi di eliminare gerarchie:
  - Accorpare entità figlie nel genitore (collasso verso l'alto)
  - Accorpare il genitore nelle entità figlie (collasso verso il basso)
  - Sostituire la generalizzazione con associazioni

COLLASSO VERSO L'ALTO

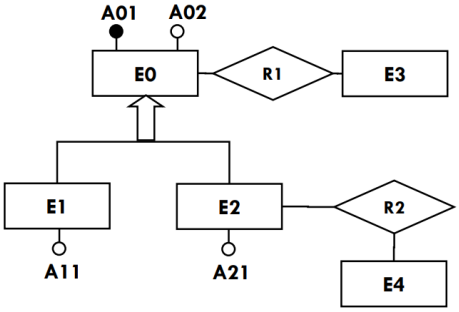


→

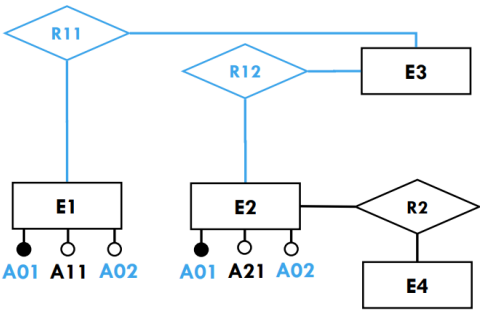


- **TIPO** è un attributo selettore che indica se l'entità appartiene a una sua sottoidentità
- Se la copertura è **totale esclusiva**: TIPO assume **N** valori, quante le sottoentità
- Se la copertura è **parziale esclusiva**: TIPO assume **N+1** valori per le istanze che non appartengono a nessuna sottoentità
- Se la copertura è **sovrapposta**: Servono **tanti selettori** quante le sottoentità
- Le eventuali associazioni connesse alle sotto-entità si trasportano su E
- Le eventuali cardinalità minime diventano 0
- Conviene se gli accessi all'entità padre e alle entità figlie sono contestuali

COLLASSO VERSO IL BASSO

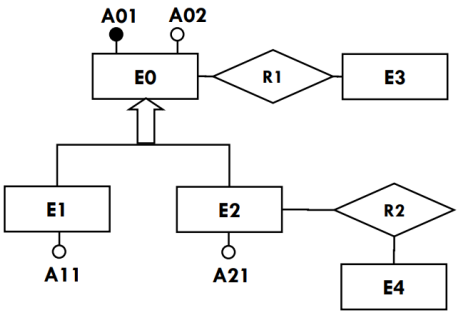


→

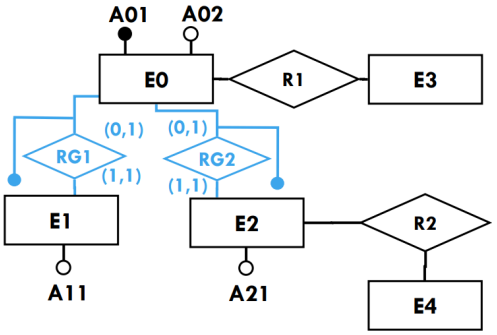


- Se copertura **non è totale**, NON si può fare
- Se copertura **non è esclusiva** introduce ridondanza
- Conviene se gli accessi alle entità figlie sono distinti, ma d'altra parte è possibile solo con generalizzazioni totali

SOSTITUIRE CON ASSOCIAZIONI



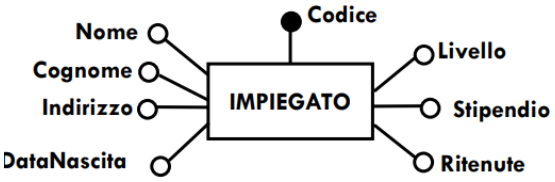
→



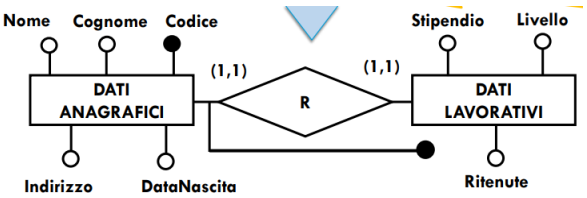
- Le entità figlie sono **identificate esternamente**
- Sostituzione con associazioni è **sempre possibile**
- Conviene se gli accessi alle entità figlie sono separati dagli accessi al padre

- Accorpamento / Partizionamento di entità e associazioni con:
  - **Partizionamento verticale** di entità
  - **Partizionamento orizzontale** di associazioni
  - Eliminazione di **attributi multivalore**
  - **Accorpamenti** di entità e associazioni

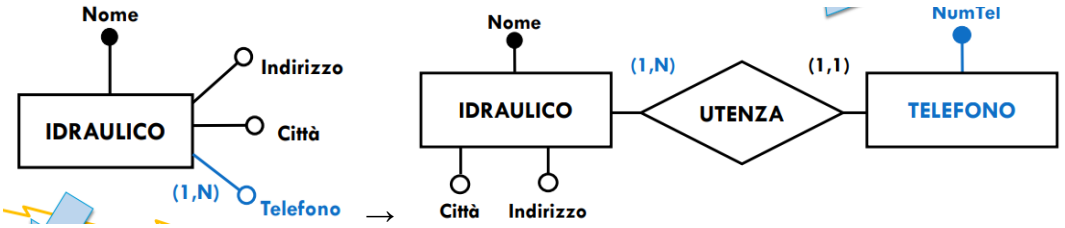
PARTIZIONAMENTO VERTICALE



→

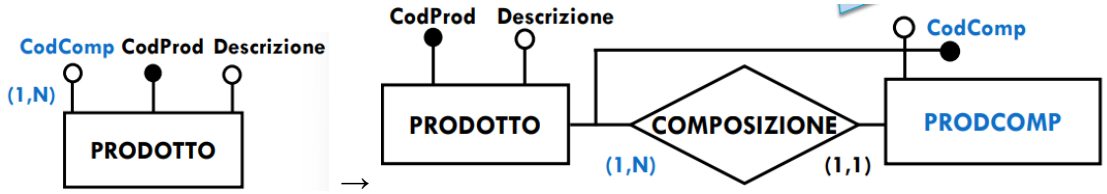


ELIMINAZIONE DI ATTRIBUTI MULTIVALORE

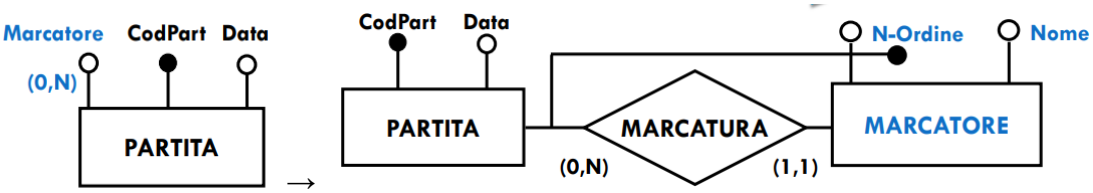


- Oppure se si sa la max_card, si usano max_card attributi

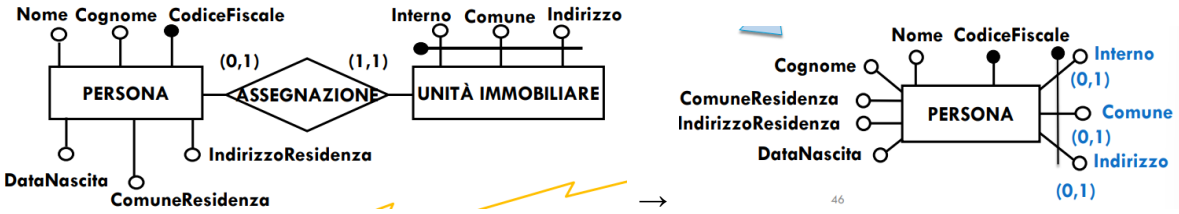
OPPURE SE VALORE DELL'ATTRIBUTO COMPARE UNA SOLA VOLTA:



OPPURE SE VALORE DELL'ATTRIBUTO COMPARE PIU' DI UNA VOLTA:

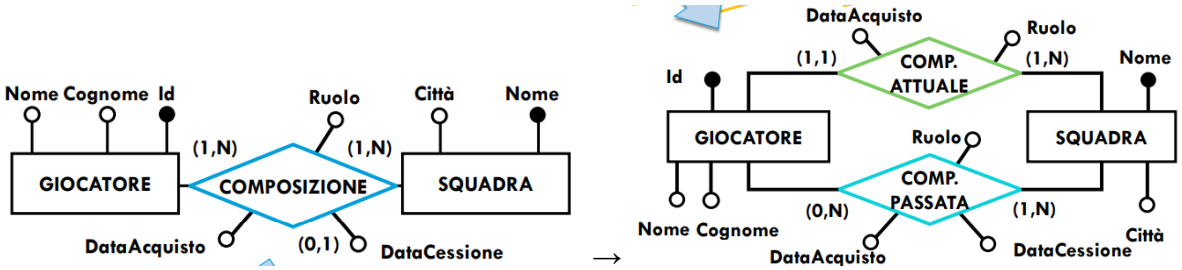


ACCORPAMENTO DI ENTITA'



- Sensato se e solo se non si vuole mantenere neanche in futuro Unità Immobiliare

PARTIZIONAMENTO ORIZZONTALE DI ASSOCIAZIONI



- Scelta degli identificatori principali (scelta della chiave primaria):
    - Assenza di opzionalità
    - Semplicità
    - Utilizzo nelle operazioni più frequenti o importanti
- Se nessuno gli rispetta, se ne creano nuovi
- Attributi composti si scompongono in attributi normali

Redazione Schema Logico

- Fatta la ristrutturazione dell'E/R si procede con lo schema logico
- Traduzione delle associazioni:
    - Ogni associazione diventa una relazione
    - Gli ID delle entità collegate sono superchiavi
- Se associazione **molti a molti**:
    - Associazione diventa entità con due FK delle entità
    - Posso rinominare le FK
- Se associazione **uno a molti**:
    - Si mette come FK nell'entità (1,1) l'identificatore dell'altra entità
    - Se associazione **ad anello**:
        - Nell'entità si mette una FK opzionale con il nome di uno dei due collegamenti
    - Se associazione con **id esterno**:
        - Si mette come FK nell'entità con id esterno l'id dell'altra entità
- Se associazione **uno a uno**:
    - Ho 3 possibilità: traduzione con 1, 2, o 3 relazioni
    - Se **3 rel**: Associazione diventa relazione e si sceglie come ID uno dei due, l'altro è UNIQUE
    - Se **2 rel**: In una delle due entità metto come FK l'ID dell'altra ed è UNIQUE
    - Se **1 rel**: Scelgo quale delle due entità far diventare relazione e accorpo tutti gli attributi, l'altro id è UNIQUE
    - Associazioni **uno a uno con opzionalità**:
        - Se **entrambe le min card** sono **0** si introduce un nuovo codice
        - Se **solo una** delle due **min card** è **0**: si sceglie come PK quella dell'entità con cardinalità (1,1)
    - Associazione **ad anello**:
        - Uso due relazioni, l'associazione diventa relazione con due FK con il nome del collegamento, una è anche PK e l'altra UNIQUE

Tavola dei Volumi

- Obiettivo è l'efficienza, che si valuta avendo la tabella dei carichi di lavoro e delle frequenze delle operazioni
- Si costruisce la tavola degli accessi (per valutare il mantenimento di ridondanze):
  - Per ogni operazione che riguarda il dato ridondante si costruisce la tabella
  - Tabella con nelle colonne: Nome Entità, Costrutto(E/A), Accessi, Tipo(L/S)
  - Per ogni modifica di un dato si fa una lettura e una scrittura
  - La scrittura vale 2
  - Alla fine si fa tabella finale:
    - Nelle colonne: Num.Operazione, Num.Accessi, Frequenza, Accessi/Giorno
    - Accessi/Giorno è dato da Num.Accessi * Frequenza
  - Si fa la stessa cosa nel caso di non ridondanza e si prende il valore più piccolo

Normalizzazione

- Proprietà di uno schema relazionale che ne garantisce la "qualità"

1NF (Prima Forma Normale):

- Se il dominio di ciascun attributo comprende solo **valori atomici** (semplici, indivisibili)

CodDip	Nome	CodDir	SediDip
D0001	Amministrazione	33301	(Milano, Napoli, Roma)
D0005	Produzione	18007	Aprilia
D0003	Ricerca	33010	Napoli

Non è in 1NF perchè SediDip contiene insiemi di valori

- Si rimuove l'attributo SediDip e lo si pone in un'altra relazione separata con chiave combinata di CodDip e SedeDip

DIPARTIMENTI(CodDip, Nome, CodDir)

CodDip	Nome	CodDir
D0001	Amministrazione	33301
D0005	Produzione	18007
D0003	Ricerca	33010

SEDI(CodDip:DIPARTIMENTI, SedeDip)

CodDip	SedeDip
D0001	Milano
D0001	Napoli
D0001	Roma
D0005	Aprilia
D0003	Napoli

2NF (Seconda Forma Normale):

- Se non c'è **dipendenza parziale** di un attributo non-primo (non fa parte della chiave) da una chiave
- Uno schema in 1NF le cui chiavi sono formate da un solo attributo, è anche in 2NF
- Esempio:
  - MAGAZZINI(Articolo, Magazzino, Quantità, Indirizzo)
  - Dipendenze funzionali (FD):
    - Articolo, Magazzino → Quantità, Indirizzo (AM → QI)
    - Magazzino → Indirizzo (M → I)
  - Problemi causati da M → I : Indirizzo dipende parzialmente dalla chiave
- Si estrae la dipendenza funzionale che crea problemi e si generano gli schemi
  - ARTICOLI_IN_MAGAZZINI(Articolo, Magazzino, Quantità) (AM → QI)
  - INDIRIZZI_MAGAZZINI(Magazzino, Indirizzo) (M → I)

3NF (Terza Forma Normale):

- Se non c'è **dipendenza transitiva** di un attributo non-primo da una chiave

IdImpiegato	Cognome	Nome	Agenzia	Luogo
001	Rossi	Carlo	02400	Bologna
002	Verdi	Maria	53880	Bergamo
003	Bianchi	Giulia	04826	Cagliari
004	Neri	Franco	23900	Cesena
005	Gialli	Marco	02400	Bologna

IdImp → Cognome, Nome, Agenzia, Luogo
Agenzia → Luogo
Dipendenza transitiva: Agenzia dipende dalla chiave IdImp e Luogo dipende da Agenzia. L dipende transitivamente da A

- Si estrae la dipendenza funzionale che crea problemi e si generano gli schemi
  - IMPIEGATI_AGENZIE(IdImpiegato, Cognome, Nome, Agenzia : AGENZIE)
  - AGENZIE(Agenzia, Luogo)

Esercizio:
Si consideri lo schema relazionale:

- TEST_LAB (MatrStudente, NomeStudente, CodCorso, NomeCorso, CodTitolare, NomeTitolare, CodEsaminatore, NomeEsaminatore, DataProva, Voto)
- Sono registrate anche eventuali prove non superate.
- Un corso ha un solo professore titolare che non coincide necessariamente con il professore esaminatore.

Scrivere le FD e normalizzarlo in 3NF

MatrStudente → NomeStudente Dipendenza Parziale
CodCorso → NomeCorso Dipendenza Parziale
CodCorso → CodTitolare Dipendenza Parziale
CodTitolare → NomeTitolare Dipendenza Transitiva
CodEsaminatore → NomeEsaminatore Dipendenza Transitiva

Poiché ci sono dipendenze funzionali parziali la relazione non è in 2NF.
Per ottenere una relazione in 2NF si devono spezzare le dipendenze parziali.
Per ottenere una relazione in 3NF si devono poi risolvere le dipendenze funzionali transitive.

Normalizzo in 2NF:

PROVE_LAB (MatrStudente: STUDENTI, CodCorso: CORSI, CodEsaminatore, NomeEsaminatore, DataProva, Voto)
STUDENTI (MatrStudente, NomeStudente)
CORSI (CodCorso, NomeCorso, CodTitolare, NomeTitolare)

Normalizzo in 3NF:

PROVE_LAB (MatrStudente: STUDENTI, CodCorso: CORSI, CodEsaminatore : PROFESSORI, DataProva, Voto)
PROFESSORI(CodProfessore, NomeProfessore)
STUDENTI (MatrStudente, NomeStudente)
CORSI (CodCorso, NomeCorso, CodTitolare : PROFESSORI)

Algebra Relazionale

- Operazioni unarie: $Selezione$: σ $Proiezione$: π $Ridenominazione$: ρ
- Operazioni Binarie: $Unione$: $\cup$ $Differenza$: $-$ $Join$:
- Selezione σ :
  - Seleziona un sottoinsieme di tuple di una relazione applicando una formula booleana F
  - F si compone di predicati connessi da AND OR NOT

ESAMI

Matricola	CodCorso	Voto	Lode
29323	483	28	no
39654	729	30	sì
29323	913	26	no
35467	913	30	no
31283	729	30	no

$\sigma_{(Voto = 30) \text{ AND } (Lode = 'no')}(ESAMI)$

$\sigma_{(CodCorso = 729) \text{ OR } (Voto = 30)}(ESAMI)$

- Proiezione π :
  - Seleziona solo delle colonne da una relazione
  - Se in delle tuple ci sono duplicati, vengono eliminati

CORSI

CodCorso	Titolo	CodDocente	Anno
483	Analisi	0201	1
729	Analisi	0021	1
913	Sistemi Informativi	0123	2

$\pi_{CodCorso,CodDocente}(CORSI)$

$\pi_{CodCorso,Anno}(CORSI)$

- Join Naturale:
  - Fa join tra relazioni. Duplicati sempre rimossi
  - Commutativo e associativo: r1><r2 = r2><r1 e r1><r2><r3 = (r1><r2)><r3
  - $0 \leq |r1 >< r2| \leq |r1| * |r2|$ cardinalità dei join

ESAMI

Matricola	CodCorso	Voto	Lode
29323	483	28	no
39654	729	30	sì
29323	913	26	no
35467	913	30	no

CORSI

CodCorso	Titolo	CodDocente	Anno
483	Analisi	0201	1
729	Analisi	0021	1
913	Sistemi Informativi	0123	2

match

$X_1 \cap X_2$ ={CodCorso}
$X_1 \cup X_2$ ={Matricola,CodCorso,Voto,Lode,Titolo, CodDocente,Anno}

ESAMI ⋈ CORSI

Matricola	CodCorso	Voto	Lode	Titolo	CodDocente	Anno
29323	483	28	no	Analisi	0201	1
39654	729	30	sì	Analisi	0021	1
29323	913	26	no	Sistemi Informativi	0123	2
35467	913	30	no	Sistemi Informativi	0123	2

- Ridenominazione ρ:
  - Cambia il nome di 1 o più attributi

REDDITI

CF	Imponibile
BNCGRG84H21A944K	27000

VOLI_NON_STOP

Numero	Giorno
FR278	28/01/2018
FR315	30/01/2018

$\rho_{CodiceFiscale \leftarrow CF}$(REDDITI)

CodiceFiscale	Imponibile
BNCGRG84H21A944K	27000

$\rho_{Codice,Data \leftarrow Numero,Giorno}$(VOLI_NON_STOP)

Codice	Data
FR278	28/01/2018
FR315	30/01/2018

- Divisione: ÷
  - Trova le istanze a cui appartengono tutti gli elementi delle relazioni:

VOLI

Codice	Data
AZ427	21/07/2017
AZ427	23/07/2017
AZ427	24/07/2017
AA056	21/07/2017
AA056	24/07/2017
AA056	25/07/2017

LINEE_VOLI

Codice
AZ427
AA056

VOLI ÷ LINEE_VOLI

Data
21/07/2017
24/07/2017

- La divisione trova le date con voli effettuati da tutte le linee aeree.

(VOLI ÷ LINEE_VOLI) ⋈ LINEE_VOLI

Codice	Data
AZ427	21/07/2017
AZ427	24/07/2017
AA056	21/07/2017
AA056	24/07/2017

- Theta-join:
  - Usato quando due relazioni non hanno attributi comuni

PARTECIPAZIONI

CodRicercatore	CodProgetto
115623	HK27
100104	HAL2000
116232	HK27
100104	HK28
201401	HAL2000

PARTECIPAZIONI $\bowtie_{CodProgetto=Sigla}$ PROGETTI

CodRicercatore	CodProgetto	Sigla	CodResponsabile
115623	HK27	HK27	116232
100104	HAL2000	HAL2000	201401
116232	HK27	HK27	116232
100104	HK28	HK28	100104
201401	HAL2000	HAL2000	201401

PROGETTI

Sigla	CodResponsabile
HK27	116232
HAL2000	201401
HK28	100104

PARTECIPAZIONI $\bowtie_{(CodProgetto=Sigla)\ AND\ (CodRicercatore \neq CodResponsabile)}$ PROGETTI

CodRicercatore	CodProgetto	Sigla	CodResponsabile
115623	HK27	HK27	116232
100104	HAL2000	HAL2000	201401

- Semi-join:
  -

IMPIEGATI

IdImp	Cognome	Nome	Qualifica
100	Bianchi	Mario	1
200	Neri	Carlotta	2
250	Rossi	Giorgio	1
300	Verdi	Maria	2

QUAL_STIP

Qualifica	Stipendio
1	18000
2	22500
3	30000

Il semijoin QUAL_STIP $\ltimes$ IMPIEGATI con schema IQ(Qualifica,Stipendio) corrisponde alle qualifiche per le quali vi è almeno un impiegato che percepisce stipendio.

Qualifica	Stipendio	IdImp	Cognome	Nome
1	18000	100	Bianchi	Mario
1	18000	250	Rossi	Giorgio
2	22500	200	Neri	Carlotta
2	22500	300	Verdi	Maria

QUAL_STIP $\bowtie$ IMPIEGATI

IQ

Qualifica	Stipendio
1	18000
2	22500

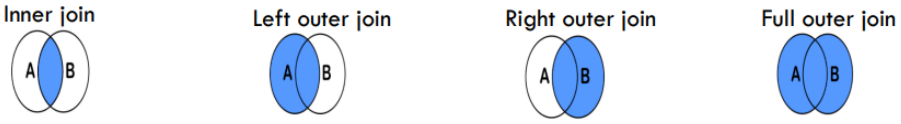
$\pi_{Qualifica,Stipendio}$ (QUAL_STIP $\bowtie$ IMPIEGATI)

$\equiv$

QUAL_STIP $\ltimes$ IMPIEGATI

- Se ci sono NULL, per operarci si usa A IS NULL o A IS NOT NULL

- Esistono 3 varianti di outer join:
  - Left outer join =><
  - Right outer join ><=
  - Full outer join =><=



CLIENTI

CF	CodComune
BNCGRG84H21A944K	A347
TRLFNC60L31G713L	A944
MAIMRA61P18F839O	A347
RLFRC72L60G713J	M185

FORNITORI

CodFornitore	CodComune
F001	A347
F002	G125
F003	A944
F004	H501

CLIENTI =⊳⊲ FORNITORI

CF	CodComune	CodFornitore
BNCGRG84H21A944K	A347	F001
TRLFNC60L31G713L	A944	F003
MAIMRA61P18F839O	A347	F001
RLFRC72L60G713J	M185	NULL

SQL

DDL Data Definition Language:

- Definisce schemi di relazioni, li modifica e li elimina
- Specifica vincoli (integrità referenziale)
- Definisce nuovi domini (int, stringa …)
- Definisce delle viste (view) e indici per accedere efficientemente ai dati
- **CREATE TABLE**
  - Definisce struttura della tabella e ne crea un'istanza vuota
  - Per ogni attributo va specificato il dominio, un eventuale valore di default e eventuali vincoli

```
CREATE TABLE Impiegati (
   CodImp     char(4)          PRIMARY KEY,              -- chiave primaria
   CF         char(16)         NOT NULL UNIQUE,          -- chiave
   Cognome    varchar(60)      NOT NULL,
   Nome       varchar(30)      NOT NULL,
   Sede       char(3)          REFERENCES Sedi(Sede),    -- FK
   Ruolo      char(20)         DEFAULT 'Programmatore',
   Stipendio int               CHECK (Stipendio > 0),
   UNIQUE (Cognome, Nome)                                 -- chiave  )
```

- **DROP TABLE** <table>: elimina lo schema di una tabella
- **ALTER TABLE**
  - Modifica schema di una tabella
  - Aggiunge attributi, aggiunge / rimuove vincoli
  - Se si aggiunge un attributo con vincolo NOT NULL, si deve prevedere un valore di default che il sistema assegnerà a tutte le tuple già presenti

```
ALTER TABLE Impiegati
ADD COLUMN Sesso char(1) CHECK (Sesso in ('M', 'F'))
ADD CONSTRAINT StipendioMax CHECK (Stipendio < 4000)
DROP CONSTRAINT StipendioPositivo
DROP UNIQUE(Cognome,Nome);
```

```
ADD COLUMN Istruzione char(10) NOT NULL DEFAULT 'Laurea'
```

Domini:

- Domini elementari
- Domini definiti dall'utente:

```
CREATE DOMAIN Voto AS SMALLINT
DEFAULT NULL
CHECK ( value >=18 AND value <= 30 )
```

Vincoli:

- **NOT NULL**
- **DEFAULT '<nome>'**
- **UNIQUE:**
  - Definisce una chiave
  - Posso scriverlo in linea o in seguito con UNIQUE(<att>, <att>)
- **PRIMARY KEY:**
  - Definisce chiave primaria
  - Non obbligatoria, al massimo una per tabella
- **FOREIGN KEY:**
  - Definisce chiave esterna
  - Politiche di reazione in caso di cancellazione/modifiche:

```
FOREIGN KEY (Sede)  REFERENCES Sedi(Sede)
```

```
ON DELETE CASCADE        -- cancellazione in cascata
ON UPDATE NO ACTION      -- modifiche non permesse
```

- **CHECK (<condizione>)**: esprime un vincolo

SELECT:

```
SELECT [ALL|DISTINCT][TOP(n)[PERCENT][WITH TIES]]
A1,A2,..,Am
FROM   R1,R2,..,Rn
[WHERE   <condizione>]
[GROUP BY <listaAttributi>]
[HAVING <condizione>]
[ORDER BY <listaAttributi>]
```

- TOP:
  - Specifica quante righe deve restituire
  - Può essere specificato con un numero o come percentuale
  - WITH TIES indica eventuali pareggi: voglio impiegato con stipendio più alto
  - SQLServer / Access:

```
SELECT  TOP(numero|percentuale) [PERCENT] [WITH TIES] A1,A2,..,Am
FROM R1,R2,..,Rn
```

  - MySQL

```
SELECT  A1,A2,..,Am
FROM R1,R2,..,Rn
...
LIMIT numero
```

  - Oracle

```
SELECT  A1,A2,..,Am
FROM R1,R2,..,Rn
...
AND ROWNUM <= numero
```

- Nel WHERE posso usare anche AND OR NOT
- **BETWEEN:**
  - Esprime condizioni di appartenenza a un intervallo
  - WHERE Stipendio BETWEEN 1300 AND 2000
- **IN**:
  - Esprime condizioni di appartenenza a un insieme
  - WHERE Sede IN (‘S02’, ‘S03’)
- **LIKE**:
  - **_** indica un solo carattere
  - **%** indica una stringa intera
  - Nomi impiegati che terminano con i e hanno i in 2° posizione
  - WHERE Nome LIKE ‘_i%i’ (Bianchi, Gialli, Violetti)
- Espressioni nella select list:
  - SELECT CodImp, **Stipendio * 12**
  - Dato che quella colonna non avrà nome, gli diamo un alias con **AS**
    - Select CodImp **AS** Codice, Stipendio*12 **AS** StipendioAnnuo
    - Posso anche omettere l’AS
- **IS**:
  - Per verificare se valore è NULL
  - WHERE Stipendio IS NULL oppure IS NOT NULL
- **ORDER BY:**
  - Ordina i valori della query
  - ORDER BY Stipendio **DESC** oppure **ASC**
- Interrogazioni su più tabelle:
  - Nel FROM indico le tabelle che mi servono (le rinomino anche)
  - Nella select list se ho attributi con stesso nome, devo rinominarle

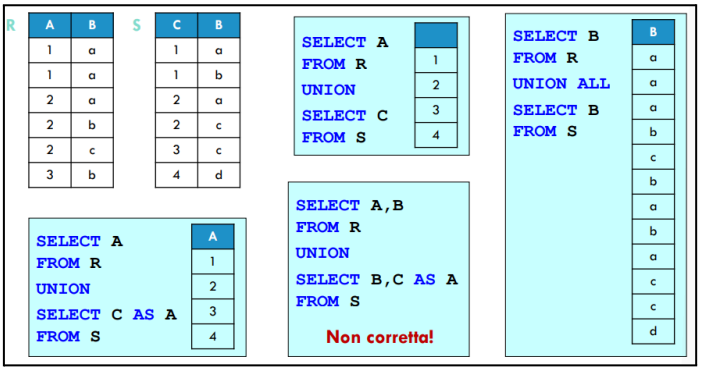
```
SELECT    I.Nome, I.Sede, S.Città
FROM      Impiegati I, Sedi S
WHERE     I.Sede = S.Sede
AND       I.Ruolo = 'Programmatore'
```

- Join:
  - Left outer join dice che nell’operatore di sinistra tengo tutte le tuple indipendentemente dal join

```
SELECT I.Nome, I.Sede, S.Città
FROM   Impiegati I JOIN Sedi S ON (I.Sede = S.Sede)
WHERE  I.Ruolo = 'Programmatore'
```

```
LEFT [OUTER] JOIN
RIGHT [OUTER] JOIN
FULL [OUTER] JOIN
NATURAL JOIN
```

- Operatori insiemistici:
  - UNION, INTERSECT, EXCEPT: eliminano duplicati
  - UNION ALL, INTERSECT ALL, EXCEPT ALL: non eliminano duplicati
  - Collega due SELECT: nelle due select devono esserci gli stessi elementi



Data Manipulation Language DML:

- **INSERT**: inserisce nuove tuple
  - Caso singolo:

```
INSERT INTO Sedi(Sede, Responsabile, Città)
VALUES            ('S04','Bruni','Firenze')
```

  - Posso non inserire attributi opzionali o di default
  - Caso multiplo:

```
INSERT INTO SediBologna(SedeBO,Resp)
SELECT Sede,Responsabile
FROM   Sedi
WHERE  Città = 'Bologna'
```

- **DELETE**: cancella tuple

```
DELETE FROM   Sedi    -- elimina le sedi di Bologna
WHERE         Città = 'Bologna'
```

- **UPDATE**: modifica tuple

```
UPDATE   Sedi
SET      Responsabile = 'Bruni',
         Città = 'Firenze'
WHERE    Sede = 'S01'

UPDATE Impiegati
SET      Stipendio = 1.1*Stipendio
WHERE    Ruolo = 'Programmatore'
```

Raggruppamenti:

- Funzioni aggregate:
  - MIN, MAX, SUM, AVG, STDEV, VARIANCE, COUNT

```
SELECT    SUM(Stipendio) AS ToTStipS01
FROM      Impiegati
WHERE     Sede = 'S01'
```

- **GROUP BY**:
  - Suddivide in gruppi le tuple risultanti da una SELECT
  - Quando c'è il PER OGNI

```
SELECT    Sede, COUNT(*) AS NumProg
FROM      Impiegati
WHERE     Ruolo = 'Programmatore'
GROUP BY  Sede
```

Sede	NumProg
S01	2
S03	1
S02	1

Le tuple che soddisfano la clausola WHERE...

CodImp	Nome	Sede	Ruolo	Stipendio
E003	Bianchi	S01	Programmatore	1000
E004	Gialli	S03	Programmatore	1000
E007	Violetti	S01	Programmatore	1000
E008	Aranci	S02	Programmatore	1200

...sono raggruppate per valori uguali della/e colonna/e presenti nella clausola GROUP BY...

CodImp	Nome	Sede	Ruolo	Stipendio
E003	Bianchi	S01	Programmatore	1000
E007	Violetti	S01	Programmatore	1000
E004	Gialli	S03	Programmatore	1000
E008	Aranci	S02	Programmatore	1200

...e infine a ciascun gruppo si applica la funzione aggregata.

Sede	NumProg
S01	2
S03	1
S02	1

- **HAVING**:
  - Filtra alcuni gruppi sulla base di condizioni
  - Si usa SEMPRE con il GROUP BY

```
SELECT    Sede, COUNT(*) AS NumImp
FROM      Impiegati
GROUP BY  Sede
HAVING    COUNT(*) > 2
```

Sede	NumImp
S01	4
S02	3

Query Innestate:

```
SELECT    CodImp    -- impiegati delle sedi di Milano
FROM      Impiegati
WHERE     Sede IN (SELECT    Sede
                   FROM      Sedi
                   WHERE     Città = 'Milano')
```

Sede
S01
S03

```
SELECT    Responsabile
FROM      Sedi
WHERE     Sede = (SELECT Sede -- al massimo una sede
                  FROM   Impiegati
                  WHERE  CodImp = 'E001')
```

Sede
S01

- **EXISTS**: verifica se risultato di una subquery restituisce almeno una tupla

```
SELECT Sede -- sedi con almeno un programmatore
FROM   Sedi S
WHERE  EXISTS  (SELECT   *
                FROM     Impiegati
                WHERE    Ruolo = 'Programmatore'
                AND      Sede = S.Sede)
```

Esempio complesso:

Sede che ha il numero maggiore di impiegati con ruolo 'Programmatore'

```
SELECT Sede, COUNT(CodImp) AS NumProg
FROM Impiegati I1
WHERE  I1.Ruolo = 'Programmatore'
GROUP BY I1.Sede
HAVING COUNT(CodImp) >= ALL
       (SELECT COUNT(CodImp)
        FROM Impiegati I2
        WHERE I2.Ruolo = 'Programmatore'
        GROUP BY I2.Sede)
```

Sede	NumProg
S01	2

In alternativa...

```
SELECT TOP(1) WITH TIES Sede, COUNT(CodImp)
FROM Impiegati
WHERE  Ruolo = 'Programmatore'
GROUP BY Sede
ORDER BY COUNT(CodImp) DESC
```

Viste e Tabelle Derivate:

```
CREATE VIEW ProgSedi(CodProg,CodSede)
AS     SELECT P.CodProg,S.Sede
       FROM   Prog P, Sedi S
       WHERE  P.Città = S.Città
```

ProgSedi

CodProg	CodSede
P01	S01
P01	S03
P01	S02
P02	S02

```
SELECT *
FROM   ProgSedi
WHERE  CodProg = 'P01'
```

CodProg	CodSede
P01	S01
P01	S03
P01	S02

- Esempio: sede con il massimo numero di impiegati:

```
CREATE VIEW NumImp(Sede,Nimp)
AS     SELECT    Sede, COUNT(*)
       FROM      Impiegati
       GROUP BY  Sede

SELECT Sede
FROM   NumImp
WHERE  Nimp = (SELECT MAX(NImp)
               FROM   NumImp)
```

NumImp

Sede	NImp
S01	4
S02	3
S03	1

- Non posso aggiornare viste se nel blocco esterno ho GROUP BY, DISTINCT, join e funzioni aggregate

- **WITH CHECK OPTION:**
  - Garantisce che ogni tupla inserita nella vista sia anche restituita dalla vista
  - WITH **CASCADED** CHECK OPTION: vale il CHECK OPTION anche per le viste create dalla vista
  - WITH **LOCAL** CHECK OPTION: non vale il CHECK OPTION per le viste create da questa vista

- Table Expressions:
  - Subquery utili

Per ogni sede, lo stipendio massimo e quanti impiegati lo percepiscono

```
SELECT SM.Sede,SM.MaxStip, COUNT(*) AS NumImpMaxStip
FROM Impiegati I,(SELECT   Sede, MAX(Stipendio)
                  FROM     Impiegati
                  GROUP BY Sede) AS SM(Sede,MaxStip)
WHERE     I.Sede = SM.Sede
AND       I.Stipendio = SM.MaxStip
GROUP BY SM.Sede,SM.MaxStip
```

SM

Sede	MaxStip
S01	2000
S02	2500
S03	1000

87

  - Common table expression: table expression temporanea:

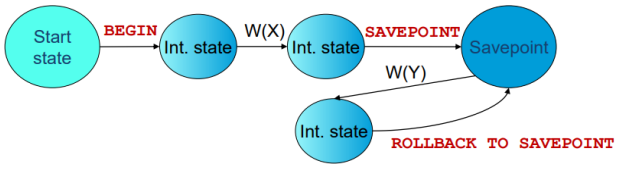
```
WITH SediStip(Sede, TotStip)
AS   (SELECT   Sede, SUM(Stipendio)
      FROM     Impiegati
      GROUP BY Sede)

SELECT Sede
FROM   SediStip
WHERE  TotStip = (SELECT MAX(TotStip)
                  FROM SediStip)
```

common table expression (validità all'interno di una singola query)

Transazioni

- Unità logica di elaborazione che corrisponde a un insieme di operazioni fisiche elementari (letture/scritture) sul DB
- DBMS garantisce che ogni transazione sia ACID:
  - **Atomicity**: o tutti gli effetti della trans. sono registrati o nessuno
  - **Consistency**: nessuno dei vincoli di integrità del DB deve essere violato
  - **Isolation**: transazione si esegue indip. dalle altre
  - **Durability**: DBMS deve proteggere il DB a fronte di guasti
- Moduli DBMS:
  - **Query Manager**: gestisce le richieste
  - **Storage Manager**: gestisce dispositivi di storage
  - **Transaction Manager**: Coordina esecuzione delle transazioni
  - **DDL Compiler**: Garantisce Consistency
  - **Logging & Recovery Manager**: Garantisce Atomicity e Durability
  - **Concurrency Manager**: Gestisce concorrenza e garantisce Isolation
- Esiti di una transazione:
  - Transazione inizia con **BEGIN**
  - Termina correttamente: quando l'app esegue il **COMMIT**
  - Termina non correttamente: usa istruzione **ROLLBACK** e torna a start state
- Possibile definire dei savepoint usati da trans. per disfare parzialmente lavoro svolto



- Gestione concorrenza:
  - **Lost Update** (dipendenza write → write):
    - Perdita di modifiche da parte di una transazione a causa di un aggiornamento effettuato da un'altra transazione
  - **Dirty Read** (dipendenza write → read):
    - Dipendenza di una transazione da un'altra non ancora completata con successo e lettura di dati inconsistenti
  - **Unrepeatable Read** (dipendenza write → write):
    - Analisi inconsistente di dati da parte di una transazione causata da aggiornamenti prodotti da un'altra
  - **Phantom Row**: quando vengono inserite o cancellate tuple che un'altra trans. dovrebbe considerare. Un tipo di Dirty Read
- **Cascading Abort**: catena di fallimenti delle transazioni alterate da T e delle transazioni eventualmente alterate da queste, e così via.
- **Recoverability**:
  - Transazione è recoverable se le viene impedito di fare commit prima che tutte le transazioni, che scrivono valori letti da T, eseguano commit o abortiscano
- Per evitare il cascading abort si impedisce che una transazione esegua dirty read

- **Lock**:
  - Tecnica dei DBMS per evitare problemi visti
  - Per eseguire un'operazione è prima necessario "acquisire" un lock sulla risorsa interessata
  - La richiesta di lock è implicita
  - Vari tipi:
    - **S (Shared)**: lock condiviso necessario per leggere
    - **X (eXclusive):** lock esclusivo necessario per scrivere / modificare
- **Lock Manager:**
  - Tiene traccia delle risorse in uso e delle trans. che le usano e che le hanno richieste
  - Quando T vuole operare su Y, inviata richiesta al lock manager
  - Lock viene accordato a T in base alla tabella di compatibilità:

		Su Y un'altra transazione detiene un lock di tipo	
		S	X
T richiede su Y un lock di tipo	S	OK	NO
	X	NO	NO

  - Quando T termina di usare Y può rilasciare il lock (unlock(Y))
  - Gestisce una lock table rispondendo a richieste tipo:
    - LOCK (transaction_id, data_item, lock_mode)
    - UNLOCK (transaction_id, data_item)
- **Strict 2-phase locking (Strict 2PL):**
  - Protocollo più usato per lo scheduler nei DBMS centralizzati.
  - A seguito di una richiesta di operazione su un certo dato da parte della transazione T, viene assegnato il lock corrispondente solo se non vi sono lock incompatibili sul dato stesso detenuti da altre transazioni.
  - Un lock acquisito dalla transazione T non può essere rilasciato almeno fino alla conferma dell'avvenuta esecuzione da parte di DM dell'operazione corrispondente.
  - Lo scheduler rilascia i lock acquisiti da T in una volta sola, al termine della transazione; più precisamente, quando il Data Manager conferma l'avvenuta esecuzione di COMMIT o ABORT.
- Atomicity e Durability:
  - **Transaction Failure**: quando transazione abortisce
  - **System Failure**: interruzione di transazioni attive da anomalia HW o SW
  - **Media Failure**: il contenuto persistente del DB
- Per far fronte a malfunzionamenti il DBMS usa diversi strumenti:
  - **DataBase Dump**: copia del DB
  - **Log File**: file in cui vengono registrate le operazioni di modifica fatte dalle tran
    - Se pagina P del DB viene modificata da T, il log contiene un record:(LSN, T, PID, before(P), after(P), prevLSN)
      - LSN = Log Sequence Number (numero del record)
      - T = Id della transazione
      - PID = id della pagina modificata
      - before(P) = contenuto di P prima della modifica e uguale after
      - prevLSN = LSN del precedente record del Log relativo a T

- **WAL Protocol**:
  - Se bisogna disfare transazione, le before image devono essere registrate nel Log prima che le relative pagine del DB siano effettivamente modificate
  - Se fallisce T, la lettura del log viene fatta a ritroso fino al BEGIN
  - Se ne occupa il Buffer Manager
- Quando una T modifica P il Buffer Manager può:
  - Politica **NO-STEAL**: Mantiene P nel buffer e attende il COMMIT prima di scriverla nel disco
  - Politica **STEAL**: Scrive P quando conviene

Lost Update

T1	X	T2
R(X)	1	
X=X-1	1	
	1	R(X)
	1	X=X-1
W(X)	0	
Commit	0	
	0	W(X)
	0	Commit

Dirty Read

T1	X	T2
R(X)	0	
X=X+1	0	
W(X)	1	
	1	R(X)
Rollback	0	
	0	...
	0	Commit

Unrepeatable Read

T1	X	T2
R(X)	0	
	0	R(X)
	1	X=X+1
	1	W(X)
	1	Commit
R(X)	1	
Commit	1	

Livello Fisico

- Dati vengono salvati in memoria centrale raggruppati in pagine
- A livello di applicazione si lavora su record logici mentre in quello di sistema di archiviazione e dispositivo su blocchi di byte
- DB consiste in file visto come collezione di pagine che memorizza più record composto da campi che rappresentano gli attributi
- Spazio fisico organizzato in Tablespaces, un insieme di extent che contengono pagine. Un insieme di extent è detto segment
- Nell header del tablespace ci sono tutte le info sugli extent
- Organizzazione a slot delle pagine:
  - Directory contiene puntatore per ogni record della pagina
  - L'ID del record, RID, è formato da: PID e Slot

Funzionalità DBMS

- Un DBMS è un insieme di programmi che permettono agli utenti di definire, costruire, manipolare e condividere una base di dati.
- Caratteristiche fondamentali:
    - Supporto di viste multiple
    - Indipendenza tra programmi e dati
    - Gestione efficiente ed efficace di dati persistenti e condivisi
    - Linguaggi di alto livello per definizione, amministrazione e controllo dei dati
- Modello dei dati: collezione di concetti usati per descrivere i dati, le loro associazioni e i vincoli che devono rispettare
- La descrizione della struttura di un DB è memorizzata nel catalogo che contiene info come la struttura dei file, il tipo e formato di memorizzazione dei dati e i vincoli
- Le info nel catalogo sono dette metadati
- Obiettivo di un DBMS è fornire indipendenza logica e fisica:
    - Indipendenza Fisica:
        - La riorganizzazione fisica dei dati non deve modificare lo schema logico del DB
    - Indipendenza Logica:
        - Modifiche allo schema logico non comportano aggiornamenti degli schemi esterni e delle applicazioni
    - Soluzione porta all'architettura a 3 livelli di un DBMS:
        - Livello Fisico
        - Livello delle Viste, create in base alle esigenze dell'utente:
            - Ristrutturazione dello schema integrato
            - Regolare controllo degli accessi
            - Calcolare dinamicamente nuovi dati dal DB
- Per gestire autorizzazioni sono previsti i Data Base Administrator
- Applicazioni basate su DBMS, 2 livelli:
    - Livelli Architetturali:
        - Riguarda struttura logica dell'app
        - Focus: organizzazione funzionale
        - Tipi: 1-tier, 2-tier, 3-tier
    - Livelli di distribuzione:
        - Riguarda dove e come sono fisicamente distribuiti i dati e i DBMS
        - Focus: Topologia fisica e geografica dei dati
        - Tipi: Centralizzato, Distribuito, Cloud, Replicato, Federato
- Architettura 1-Tier (utente → sistema locale):
    - Tutto eseguito sullo stesso livello, utente interagisce direttamente con DB
    - Semplice da implementare e usare, bassa latenza
    - Non scalabile, sicurezza debole, no separazione tra UI e logica del DB
- Architettura 2-Tier (utente → client → database server):
    - Separazione tra client e server, client gestisce UI, server gestisce DBMS
    - Migliore separazione delle responsabilità, più sicuro e organizzato
    - Scalabilità limitata: ogni client apre connessione diretta al server
    - Difficile da gestire se ci sono molti utenti

- Architettura 3-Tier (utente → client → application server → database server):
    - Introdotto app. server tra client e DBMS. client gestisce UI, App. server gestisce le API e logica applicativa, DBMS dove risiedono e gestiscono dati
    - Scalabile, separazione dei ruoli, + sicurezza, caching, load balancing
    - Complessità maggiore e richiede più risorse e infrastrutture
- Database Centralizzato:
    - Dati archiviati e gestiti in un singolo DB localizzato in un server centrale
    - Semplice gestione, coerenza
    - Bassa fault tolerance, scalabilità limitata
- Database Distribuito:
    - Dati distribuiti su + macchine ma sistema appare a utenti come unico DB
    - Buona fault tollerance e scalabilità orizzontale
    - Complessità e ritardi nella latenza
- Cloud Database:
    - Dati archiviati e gestiti su server gestiti da provider di servizi cloud (AWS)
    - Scalabilità orizzontale, alta fault tolerance, distr. geografica, consistenza e coerenza dei dati, sicurezza, prestazioni ottimizzate
- Database Replicato:
    - Dati di un DB copiati su più server
    - Alta fault tolerance e carico di lavoro bilanciato su più nodi
    - Problemi di coerenza con replica asincrona e overhead di replicazione
- Database Federato:
    - Più DB separati interagiscono tra loro e condividono dati
    - Indipendenza, Trasparenza, Interoperabilità e consistenza

Organizzazioni Primarie

Tipi di organizzazioni dei dati:

- Primaria vs Secondaria:
  - Primaria definisce un criterio di allocazione dei dati nei file
  - Secondaria definisce un altro metodo di accesso all'organizzazione primaria.
- Statica vs Dinamica:
  - Dinamica si adatta alla mole dei dati
  - Statica prevede fasi di riorganizzazione a fronte di variazioni del volume dati
- Per Chiave Primaria vs Chiave Secondaria:
  - Il valore della chiave identifica un unico record in un'organizzazione per chiave primaria, e più record nel secondo caso.

- Clustering: Presenza di addensamenti di dati nei blocchi del file
- Indexing: Riguarda accesso ai dati, possibilità di risolvere efficacemente op di ricerca
- File non strutturato: sequenza di byte (stream)
- FIle strutturato: collezione di record

Organizzazione primarie e secondarie:

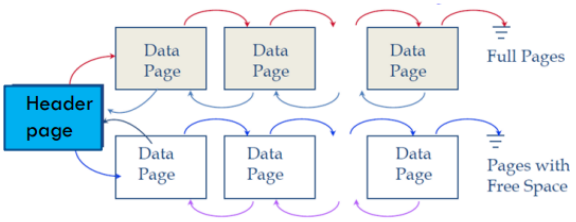
- Le primarie si suddividono in 4 categorie:
  - Strutture sequenziali non ordinate - Heap
  - Strutture sequenziali ordinate - sorted sequential file
  - Ad accesso calcolato - Hash File
  - Ad albero - indexed organization
- Le secondarie sono principalmente strutture ad albero

Organizzazioni Sequenziali:

- Sequenza logica di registrazione può rispettare:
  - Ordine temporale - Heap File
  - Ordinamento in base a una chiave - sorted sequential file
- S. sequential file facilita la ricerca su attributi di ordinamento; mantenimento oneroso

Heap File:

- Inserimento di un record sempre fatto appendendolo alla fine del file
- Permette di gestire record a lunghezza fissa o variabile
- Per organizzare pagine o si usa una doubly linked list o un altra che non so



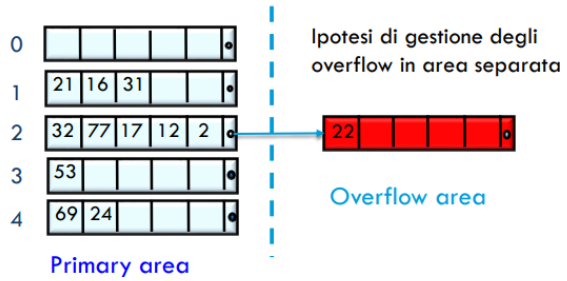
- Struttura non è efficiente per memorizzare record a lunghezza variabile
- Periodicamente deve essere fatta una riorganizzazione del file
- Costo ricerca di un valore: (NP + 1) / 2 o nel caso peggiore NP
- Costo ricerca in intervallo: NP
- Costo ricerca di più occorrenze di un valore: NP
- Costo inserimento record: 2 (lettura e scrittura pagina)
- Costo eliminazione di un record: C(ricerca) + 1
- Modifica di un record: C(ricerca) + 1

Sorted Sequential File:

- Vantaggioso se si ricerca record secondo ordine dei valori di chiave di ordinamento
- Rispetto a Heap File nessun vantaggio per ricerche su attributi non di ordinamento
- Si può usare ricerca binaria che costa log n
- Dispendioso mantenere ordinamento
- Costo ricerca di valore: log NP (NP è numero di pagine)
- Costo inserimento record: log NP + 1
- Costo eliminazione: log NP + 1 oppure C(ricerca) = 1 se noto RID
- Costo modifica record: log NP + 1 oppure C = 1 se noto RID oppure C = (NP+1)/2

Hash File:

- Uso di funzioni hash per allocare record sulla base di attributo chiave
- Indirizzi generati da H individua una pagina logica (bucket)
- Area di memoria dei bucket chiamata area primaria
- Presente anche area di overflow che possono verificarsi
- Efficiente per reperimento di record dato il suo valore chiave, inefficiente per altre ric.
- Organizzazione Hash Statica:
  - NP fissato alla creazione della struttura, se troppi overflow si fa riorganizzazi.
- Organizzazione Hash Dinamica:
  - NP è variabile, area primaria può espandersi e contrarsi; servono più funzioni
- Per queste organizzazioni bisogna:
  - Scegliere funzione Hash
  - Politica gestione di overflow
  - Capacità C dei bucket dell'area primaria
  - Capacità C_{ov} dei bucket di overflow



esempio con NP = 5 e C = 5
funzione hash H(k) = k mod 5
overflow alloca per ogni bucket uno o più bucket di overflow con C_{ov} = 5

- 2 tipi di organizzazioni hash dinamiche:
  - con directory: virtual hashing
  - senza directory: linear hashing

- Virtual Hashing:
    - Raddoppia area primaria quando si verifica overflow e ridistribuisce i record tra bucket saturo e il suo buddy usando una nuova hash function
    - Si esegue quindi lo split del bucket saturo
- Linear Hashing:
    - Non usa una struttura aggiuntiva
    - Si suddivide un altro bucket scelto secondo un criterio prefissato
    - Si allocano NP_0 bucket e si usa $H_0(k) = k \bmod NP_0$

- Costo Ricerca di valore: 1
- Costo ricerca in intervallo: NP
- Costo inserimento: 1 + 1 lettura e riscrittura
- Costo eliminazione: 1 + 1 lettura e riscrittura
- Costo modifica di record: 1 + 1 lettura e scrittura

Indici

- Struttura dati che rende disponibile un “cammino d’accesso” atto a localizzare efficientemente i record d’interesse nel rispetto di un criterio di selezione.
- Mappa che memorizza [valore di chiave di ricerca, riferimento al record o ai record]
- 2 tipi:
    - Ordered Index: organizzazione ad albero B^+-tree
    - Hash Index: organizzazione secondaria che usa hash function efficiente per ricerche su singolo valore

- NELLA PROGETTAZIONE LOGICA, QUANDO HO ASSOCIAZIONE 1 A MOLTI E IMPORTO LA CHIAVE NELL'ENTITÀ CON CARDINALITÀ 1, SE CARDINALITà è 0-1 LA CHIAVE CHE IMPORTO SARà OPZIONALE
-