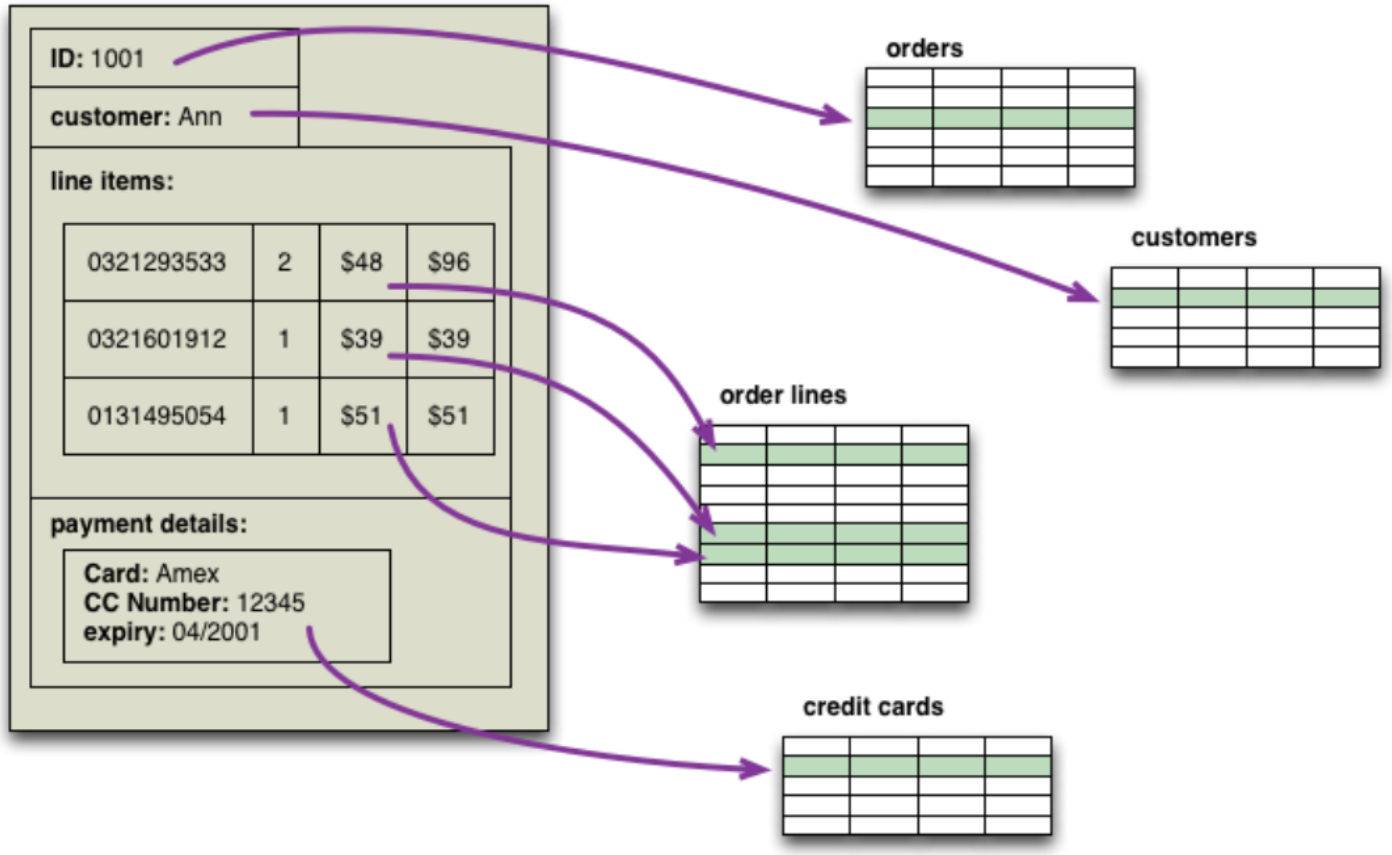
NoSQL

- Riferimento: P. J. Sadalage, M Fowler, NoSQL Distilled, Addison Wesley

PERCHÉ NoSQL

- 20 anni di successi della tecnologia relazionale che ha garantito:
  - Persistenza
  - Concurrency control
  - Meccanismi di integrazione
- Ma gli sviluppatori di applicazioni hanno sofferto lo impedance mismatch tra il modello relazionale e le strutture dati in-memory
- Tendenza a non utilizzare più come punto di integrazione i database e a realizzare l'integrazione mediante i servizi
- Fattore essenziale tecnologico: la necessità di usare clusters, che non si prestano a implementazioni efficienti dei database relazionali

IMPEDANCE MISMATCH



PERCHÉ NoSQL (cont.)

- NoSQL è un termine vago che si riferisce a molti approcci diversi alla gestione di database.
- Le caratteristiche condivise dai database NoSQL sono
  - Non usano il modello relazionale
  - Sono eseguiti efficientemente sui clusters
  - Open-source
  - Costruiti per le web estates sel 21-esimo secolo: Google, Amazon ets.
  - Senza schema (schemaless)

DATA MODEL

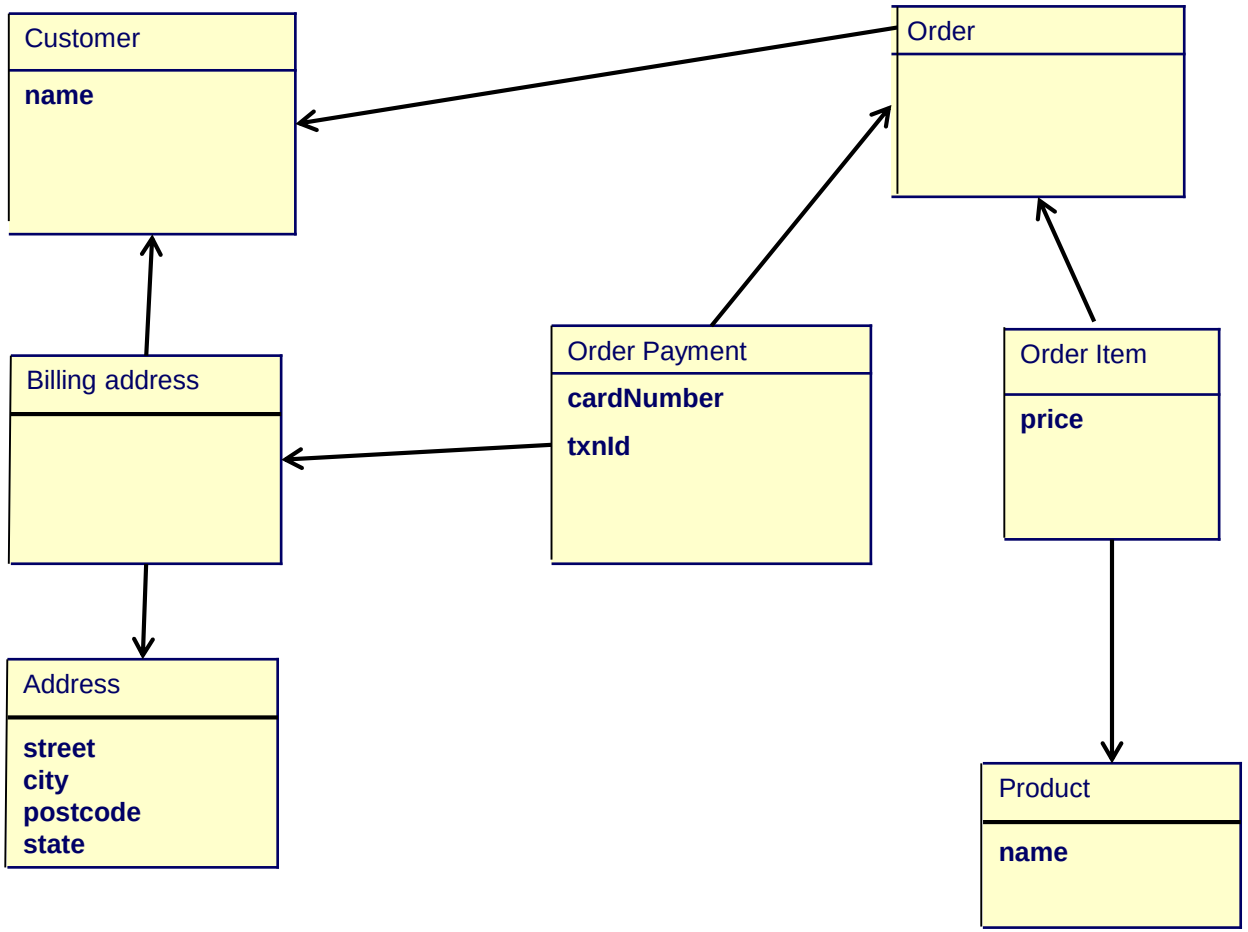
- I database NoSQL abbandonano il modello relazionale delle tabelle e dei riferimenti tramite valori e sono raggruppabili in accordo ai seguenti data model:
  - Aggregate orientation model
    - Key-value (coppie chiave-valore)
    - Documenti
    - Column-family
  - Graph data model

AGGREGATI

- I database relazionali operano su ennuple di valori elementari non consentendo strutturazioni più complesse
- La caratterisitica primaria dei sistemi aggregate oriented è di consentire strutture complesse.
- L'aggregato si presenta come l'unità di manipolazione dati e di mantenimento della consistenza
- L'organizzazione dei dati in aggregati rende naturale organizzare il calcolo sui cluster, poiché gli aggregati si presentano come l'unità naturale per le duplicazioni e a frammentazione (sharding)
- L'organizzazione dei dati in aggregati facilita le applicazioni che possono manipolare aggregati

ESEMPIO DI RIFERIMENTO: SITO DI E-COMMERCE

- Vesndita di item via web
- Informazione da gestire:
  - Utenti
  - Catalogo dei prodotti
  - Ordini
  - Indirizzi di spedizione
  - Dati sul pagamento


Customer
name
Order
Billing address
Order Payment
cardNumber
txnId
Order Item
price
Address
street
city
postcode
state
Product
name

OSSERVAZIONE

- Il database è normalizzato ovvero nessun dato è ripetuto su più tabelle
- Anche l'integrità referenziale è garantita

TIPICI DATI PER IL DBMS RELAZIONALE

Customer

Id	Name
1	Martin

Order

Id	Customer Id	SchippingAddress Id
99	1	77

Product

Id	Name
127	NoSQL book

BillingAddress

Id	Customer Id	AddressId
99	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

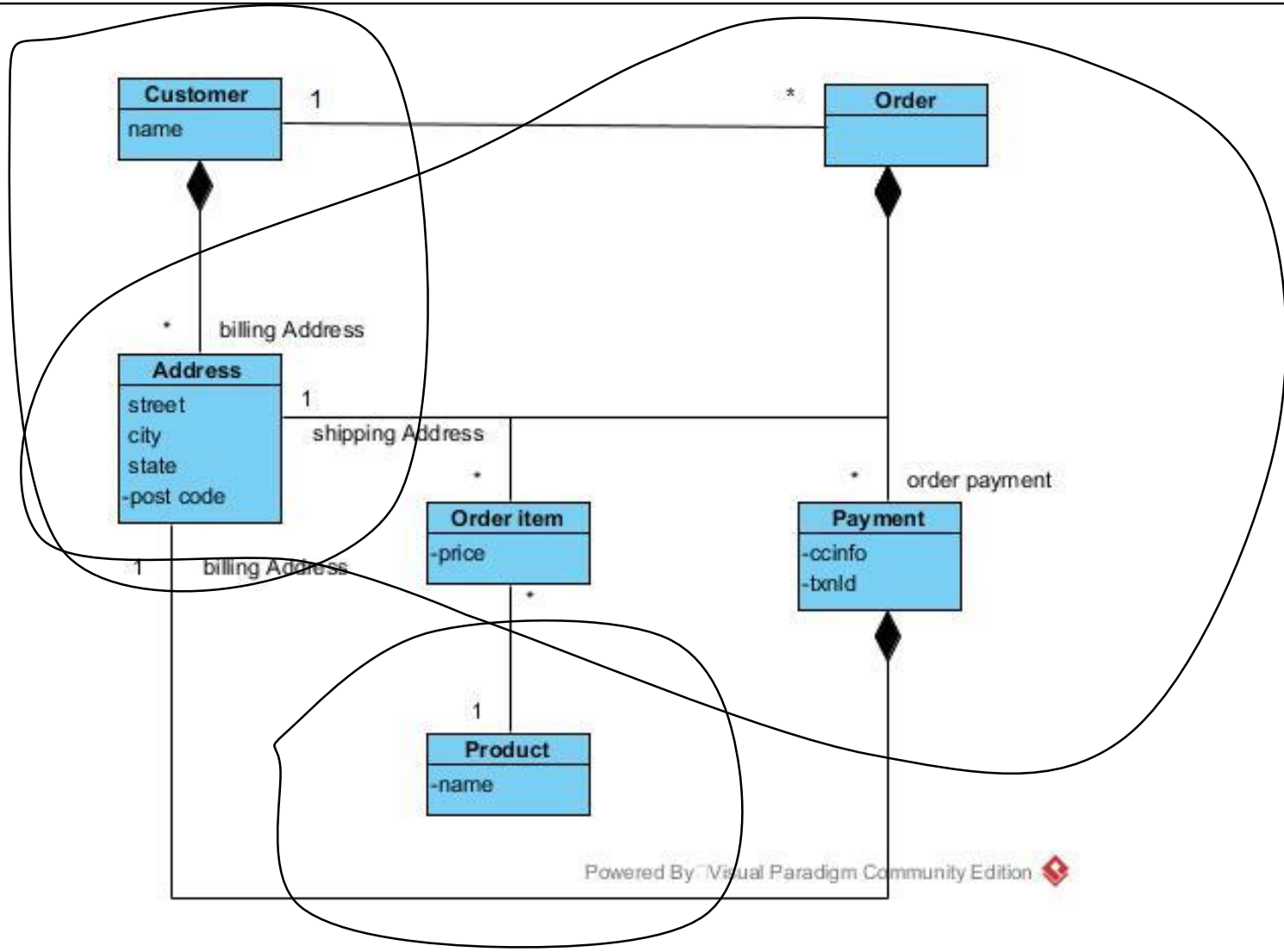
Address

Id	ity
77	Chicago

OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

VISTA AGGREGATED-ORIENTED



VISTA AGGREGATED-ORIENTED

- Due aggregati: customer e order
- Customer *contiene* una lista di billing addresses
- Order *contiene* una lista di order items, uno shipping address e pagamenti. Il pagamento stesso contiene un billing address per quel pagamento
- Un singolo record logico per l'indirizzo appare tre volte, ma invece di essere gestito tramite ID è trattato come un valore e copiato ogni volta. Questo garantisce che il valore dell'indirizzo non cambierà.

ESEMPIO DI DATI IN JSON (JavaScript Object Notation)

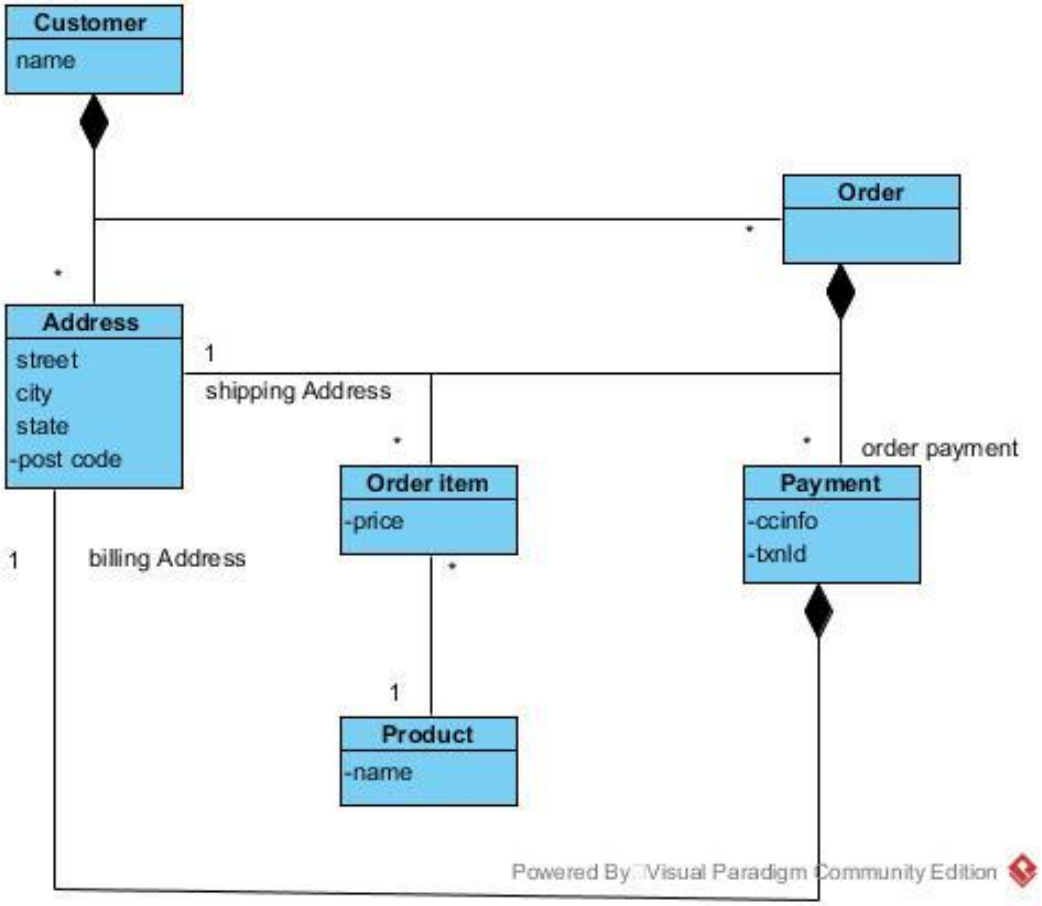
```
// in orders
{
«id»:99,
«customerId»:1,
«OrderItems»:[{
«productId»:27,
«price»:32.45,
«productName: «NoSQL Distilled»
  }]
«shippingAddress»: [{«city»:»Chicago]}
«orderPayment»:[
  {
     «ccinfo»: «1000-1000-1000-1000»,
      «txnId»: «abelif879rftZ»,
      «billingAddress»: {«city»: «Chicago»}
   }
   ],
}
```

```
// in customers
{«id»:1,
«name»: «Martin»,
«billingAddress»:[{«city»: «Chicago»}]
}
```

VISTA AGGREGATED-ORIENTED (cont.)

- Il link tra customer e Order non è registrato in nessuno dei due aggregati, ma è una relazione tra aggregati
- In modo simile il link da un order item andrebbe in un'altra struttura aggregata per i Product, ma qui il Product Name è parte dell'Order Item per minimizzare il numero di aggregati da accedere.
- La progettazione è guidata dal modo in cui i dati saranno acceduti

UN MODELLO ALTERNATIVO



DATI IN QUESTO MODELLO

```
// in customers
{
«customer»: {
{«id»:1,
«name»: «Martin»,
«billingAddress:[{«city»:»Chicago»}],
«orders»: [
  {
     «id»:99,
     «customerId»:1,
     «OrderItems»:[
      {
      «productId»:27,
      «price»:32.45,
          «productName: «NoSQL Distilled»
          }
 ]
   «ShippingAddress:[{«city»:»Chicago»}]
   «OrderPayment»: [
    {
     «ccinfo»: «1000-1000-1000-1000»,
      «txnId»: «abelif879rft»,
      «billingAddress»: {«city»:»Chicago»}
    }
    ],
}
```

AGGREGATE IGNORANT VS AGGREGATE ORIENTED

- Aggregate oriented database facilitano, tramite la nozione di aggregato, l'implementazione su cluster (un aggregato viene assegnato per intero a un nodo).
- Aggregate ignorant (relational) database sono più flessibili in quanto mantengono i dati a un livello di granularità più fine che consente la costruzione di aggregati di tipo diverso in caso di bisogni diversi, ma gli aggregati «logici» possono essere spezzati e assegnati a nodi diversi.
- Nei datatabase aggregate oriented la proprietà di atomicità delle transazioni è in genere offerta a livello di aggregato

FAMIGLIE NOSQL

- Sistemi Aggregate oriented
  - Key-value
  - Document
  - Column-family
- Graph databases

KEY-VALUE E DOCUMENT DATA MODELS

- Ogni aggregato è dotato di una chiave o id usato per ottenere i dati
- Nei database key-value oriented l'aggregato è opaco (un blob di bit)
  - Vantaggio: massima libertà di utilizzo
- Nei database document oriented la struttura dell'aggregato è visibile
  - Vantaggio: è possibile effettuare queries basate sui campi dell'aggregato, è possibile recuperare solo parte dell'aggregato e il database può creare indici basati sul contenuto dell'aggregato

COLUMN-FAMILY STORE

- Progenitore: Google BigTable
- Sistemi attuali:
  - Hbase
  - Cassandra
- Struttura: two-level map, ovvero a ogni chiave è associato un record
- Le colonne sono raggruppate in famiglie
- Contesto: poche scritture e necessità frequente di leggere alcune colonne di molte righe insieme
- Soluzione: la basic storage unit è meglio che sia per gruppi (famiglie) di colonne per tutte le righe

STRUTTURA DEI COLUMN-FAMILY STORES

- Il modello per questa classe di database è una struttura aggregata a due livelli
  - La prima chiave è un row-identifier che seleziona l'aggregato di interesse
  - Il secondo livello è un insieme di colonne
- Le operazioni consentono di accedere alla riga come un tutt'uno, ma anche di prelevare una particolare colonna

Esempio:

Per ottenere il nome di un particolare cliente: Get('1234','name')

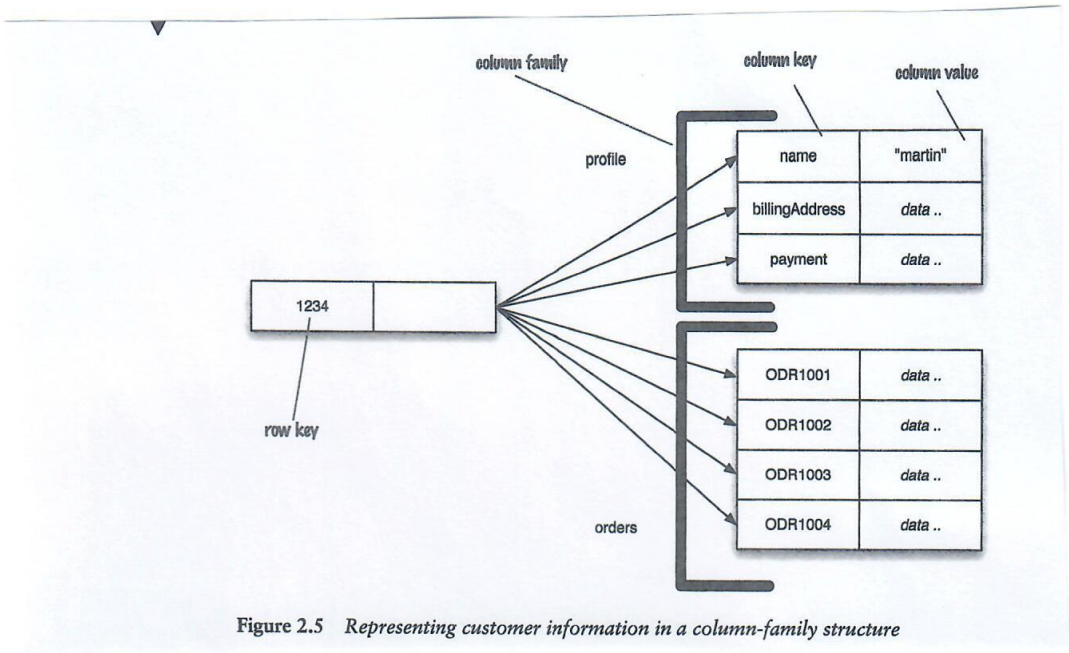


Figure 2.5 *Representing customer information in a column-family structure*

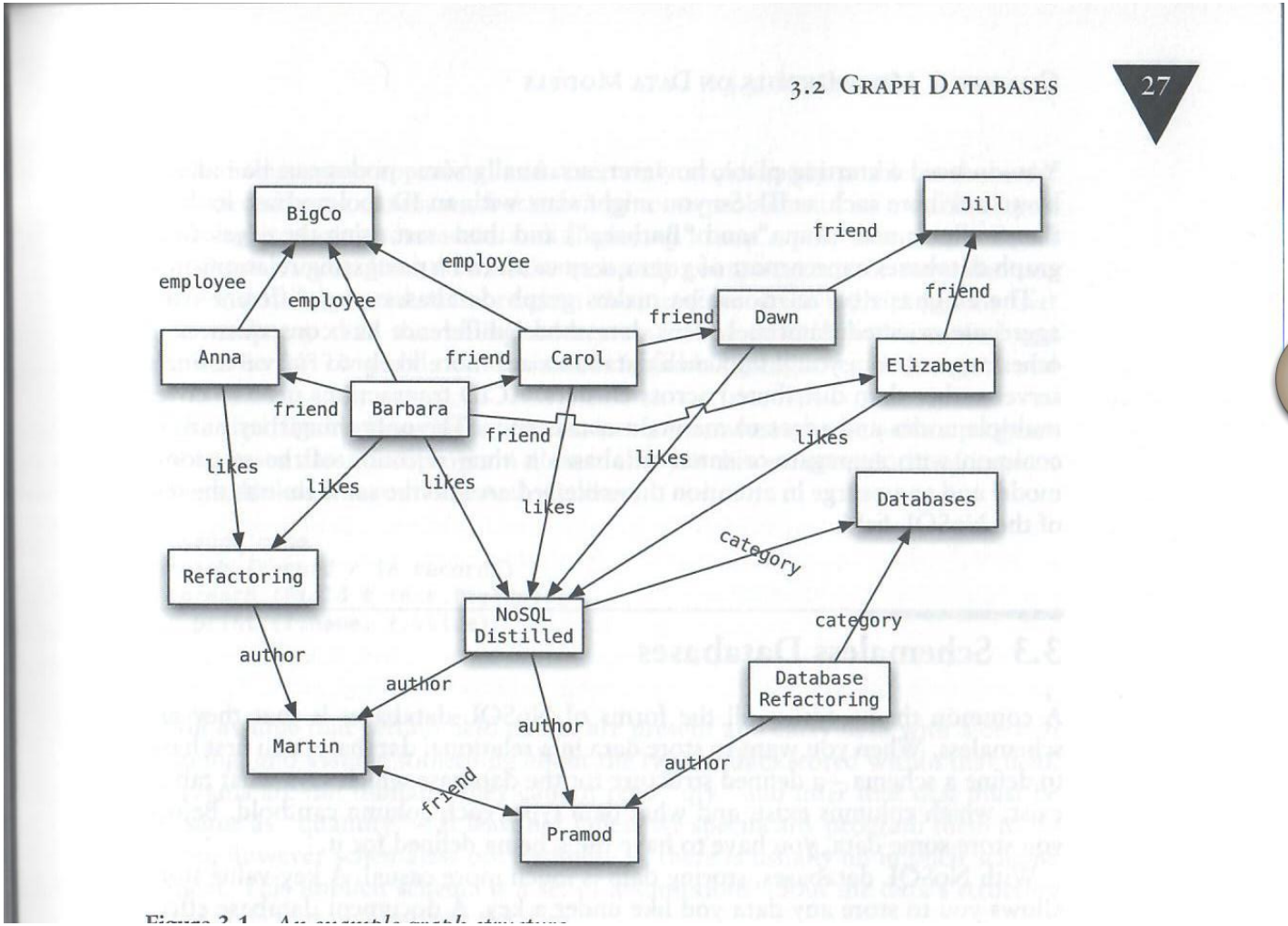
AGGREGATE ORIENTED DATABASES: SOMMARIO

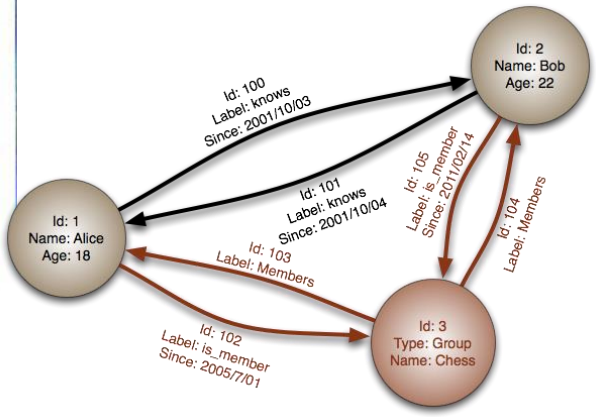
- Condividono
  - Aggregati indicizzati da una chiave da usare per il lookup
  - L'intero aggregato viene memorizzato su un nodo di un cluster
  - L'aggregato costituisce l'unità atomica per update ed è la base per il controllo delle transazioni.
- Differenze:
  - Il modello key-value tratta l'aggregato come un blob opaco
  - Il document model vede l'aggregato trasparente e consente queries sul contenuto e retrieval parziale
  - Column-family model partiziona l'aggregato in famiglie di colonne: migliora l'accessibilità al dato imponendo una strutturazione

GRAPH DATABASES

- Strumenti per memorizzare e interrogare grafi
- Strutturazione opposta agli aggregate oriented databases:
  - Record di dimensioni ridotte
  - Interconnessioni complesse
- Tipica query: «elenca i libri nella categoria Databases che sono scritti da qualcuno che piace a un mio amico»
- Possibili campi applicativi:
    - Social networks
    - Product preferences
    - Eligibility rules

Graph databases





MODELLO DI BASE

- Tutti i Graph Database Management systems condividono il modello di base Nodi e Archi
- Su questa base:
  - FlockDB usa il modello base Nodi e Archi senza meccanismi per ulteriori attributi
  - Neo4J consente di inserire oggetti Java come proprietà di nodi e archi
  - Infinite Graph memorizza oggetti Java (sottoclassi dei tipi built-in) come nodi e archi
- Tutti i sistemi offrono meccanismi per le interrogazioni che sfruttano l'idea di grafo

DIFFERENZE TRA IL MODELLO RELAZIONALE E GRAPH BASED

- Differenza fondamentale: nei graph database non è richiesto di definire a priori uno schema
- I linguaggi di navigazione dei grafi offrono costrutti di navigazione non presenti in SQL come:
  - Chiusura transitiva
  - Navigare tutte le relazioni con nome 'A*'
- I sistemi graph based sono ottimizzati per ricerche su grafo e per ricerche che partono da un singolo nodo
- Nei sistemi graph-based il concetto di 'record' non è centrale:
  - Nei DBMS si tende a leggere tutti i campi del record assieme
  - Nei graph based si accedono solo i campi necessari alla query

GRAPH DATABASE: NAVIGAZIONE

- La navigazione avviene percorrendo gli archi, ma sono necessari dei punti di partenza
- Tipicamente alcuni nodi sono indicizzati su qualche attributo identificatore.
- Continuando l'esempio:
  - Cerca tramite look-up sugli indici persone di nome «Anna» e «Barbara» e da lì inizia la navigazione sugli archi

GRAPH DATABASE: ULTERIORI CARATTERISTICHE

- Al contrario degli aggregate-oriented database, il cui progetto è motivato dalla necessità di eseguire le queries su cluster di macchine, i graph database richiedono di essere eseguiti su un singolo server
- Tipicamente transazioni ACID (Atomicity, Consistency, Isolation, e Durability) devono coinvolgere nodi e archi multipli per garantire la consistenza
- L'unica cosa in comune tra graph databases e aggregate oriented database è il rifiuto del modello relazionale.