

# *COMS W4111 - Introduction to Databases*

*Continue and Complete NoSQL DBs; Normalization/De-Normalization; Data Analysis*

*Donald F. Ferguson (dff@cs.columbia.edu)*

# *Contents*

# Contents

- A Diversion: Facebook Graph API
- Putting the pieces together: SQL and NoSQL example.
- Relational Normalization/De-normalization.
- Setup for HW5

# *A Little More on NoSQL*

# *Facebook Graph API*

# Facebook Graph API

## Overview

<https://developers.facebook.com/docs/graph-api/overview>

The Graph API is the primary way to get data into and out of the Facebook platform. It's an HTTP-based API that apps can use to programmatically query data, post new stories, manage ads, upload photos, and perform a wide variety of other tasks.



### The Basics

The Graph API is named after the idea of a "social graph" — a representation of the information on Facebook. It's composed of:

- **nodes** — basically individual objects, such as a User, a Photo, a Page, or a Comment
- **edges** — connections between a collection of objects and a single object, such as Photos on a Page or Comments on a Photo
- **fields** — data about an object, such as a User's birthday, or a Page's name

Typically you use nodes to get data about a specific object, use edges to get collections of objects on a single object, and use fields to get data about a single object or each object in a collection.

# Facebook Graph API

## Graph API Root Nodes

This is a full list of the Graph API root nodes. The main difference between a root node and a non-root node is that root nodes can be queried directly, while non-root nodes can be queried via root nodes or edges. If you want to learn how to use the Graph API, read our [Using Graph API guide](#), and if you want to know which APIs can solve some frequent issues, try our [Common Scenarios guide](#).

Node	Description
Achievement	Instance for an achievement for a user.
Achievement Type	Graph API Reference Achievement Type /achievement-type
Album	A photo album
App Link Host	An individual app link host object created by an app
App Request	An individual app request received by someone, sent by an app or another person
Application	A Facebook app
Application Context	Provides access to available social context edges for this app
Async Session	Represents an async job request to Graph API
Audience Insights Rule	Definition of a rule
CTCert Domain	A domain name that has been issued new certificates.
Canvas	A canvas document
Canvas Button	A button inside the canvas
Canvas Carousel	A carousel inside a canvas

Full list at:

<https://developers.facebook.com/docs/graph-api/reference>

... ... ...

Life Event	Page milestone information
Link	A link shared on Facebook
Live Encoder	An EntLiveEncoder is for the live encoders that can be associated with video broadcasts. This is part of the reference live encoder API
Live Video	A live video
Mailing Address	A mailing address object
Message	An individual message in the Facebook messaging system.
Milestone	Graph API Reference Milestone /milestone
Native Offer	A <code>native offer</code> represents an Offer on Facebook. The <code>/{offer_id}</code> node returns a single <code>native offer</code> . Each <code>native offer</code> requires a <code>view</code> to be rendered to users.
Notification	Graph API Reference Notification /notification
Object Comments	This reference describes the <code>/comments</code> edge that is common to multiple Graph API nodes. The structure and operations are the same for each node.

# Facebook Graph API – Examples

[https://graph.facebook.com/v3.2/me/posts?access\\_token=xxxxxxxx](https://graph.facebook.com/v3.2/me/posts?access_token=xxxxxxxx)

The screenshot shows a REST client interface with the following details:

- URL:** https://graph.facebook.com/v3.2/me/posts?access\_token=xxxxxxxx
- Status:** 200 OK
- Time:** 545 ms
- Body:** JSON response (Pretty printed)
- Headers:** (17) - This tab is selected.
- Cookies:** - This tab is unselected.
- Test Results:** - This tab is unselected.

The JSON response body contains the following data:

```
1 {  
2   "data": [  
3     {  
4       "message": "\"Without substantial and sustained global mitigation and regional adaptation efforts, climate change is expected  
5         \"created_time": "2018-11-26T19:29:35+0000",  
6         "id": "10155984776918693_10156118626273693"  
7     },  
8     {  
9       "message": "This is just priceless. I doubt this was security issue, but still ... You gotta wonder what they were thinking.\\"  
10      "created_time": "2018-11-20T22:10:11+0000",  
11      "id": "10155984776918693_10156106055763693"  
12    },  
13    {  
14      "message": "Working with NY Guard and NY National Guard soldiers to distribute Thanksgiving turkeys. Will send a link to the  
15        \"created_time": "2018-11-19T14:10:06+0000",  
16        "id": "10155984776918693_10156103345058693"  
17    },  
18    {  
19      "message": "Busy day at the office.",  
20      "created_time": "2018-11-17T18:31:32+0000",  
21      "id": "10155984776918693_10156099248598693"  
22  },  
23  {  
24    "message": "A recent interview in Wired.\n",  
25    "created_time": "2018-11-17T16:06:08+0000",  
26    "id": "10155984776918693_10156099003333693"  
27 },  
28 }  
}
```

# Facebook Graph API – Examples

[https://graph.facebook.com/v3.2/me/likes?access\\_token=xxxxxxxx](https://graph.facebook.com/v3.2/me/likes?access_token=xxxxxxxx)

The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: https://graph.facebook.com/v3.2/me/likes?access\_token=EAAE2HWkr2kIBANW8p4YZCiZA82WiRkqabbKnw5GEA...
- Params: None
- Send and Save buttons

The response body is displayed in JSON format, showing a list of likes with their IDs and creation times, followed by a "paging" object containing cursors and a "next" link.

```
100    "id": "120885117945175",
101    "created_time": "2010-09-04T18:57:10+0000"
102  },
103  {
104    "name": "Making it Big in Software: Get the job. Work the org. Become great.",
105    "id": "337756296552",
106    "created_time": "2010-03-27T10:54:20+0000"
107  },
108  {
109    "name": "Tim's Sounds",
110    "id": "311405670729",
111    "created_time": "2010-02-06T14:56:33+0000"
112  },
113  {
114    "name": "Erasmus Mundus IMSE",
115    "id": "186881751274",
116    "created_time": "2009-12-06T14:18:14+0000"
117  }
118 ],
119 "paging": {
120   "cursors": {
121     "before": "MTE1MDM3NTQ4NTE0NzMx",
122     "after": "MTg20DgxNzUxMjc0"
123   },
124   "next": "https://graph.facebook.com/v3.2/10155984776918693/likes?access_token=EAAE2HWkr2kIBANW8p4YZCiZA82WiRkqabbKnw5GEAM2AORuqJ"
125 }
126 }
```

Notice use of paging links.

# Facebook Applications

- Facebook Application
  - Define an application at [developers.facebook.com/apps/](https://developers.facebook.com/apps/)
  - Users log into your application via Facebook.
  - This grants your application permission to access the user's Facebook content.
  - User must approve access on login and Facebook must "approve" the application for non-profile information.
- Once logged onto your application via Facebook, you can
  - Get basic information about people.
  - Enable people to share, etc. information from your application to Facebook.
  - Perform analytics on users and their networks, subject to granted permissions.

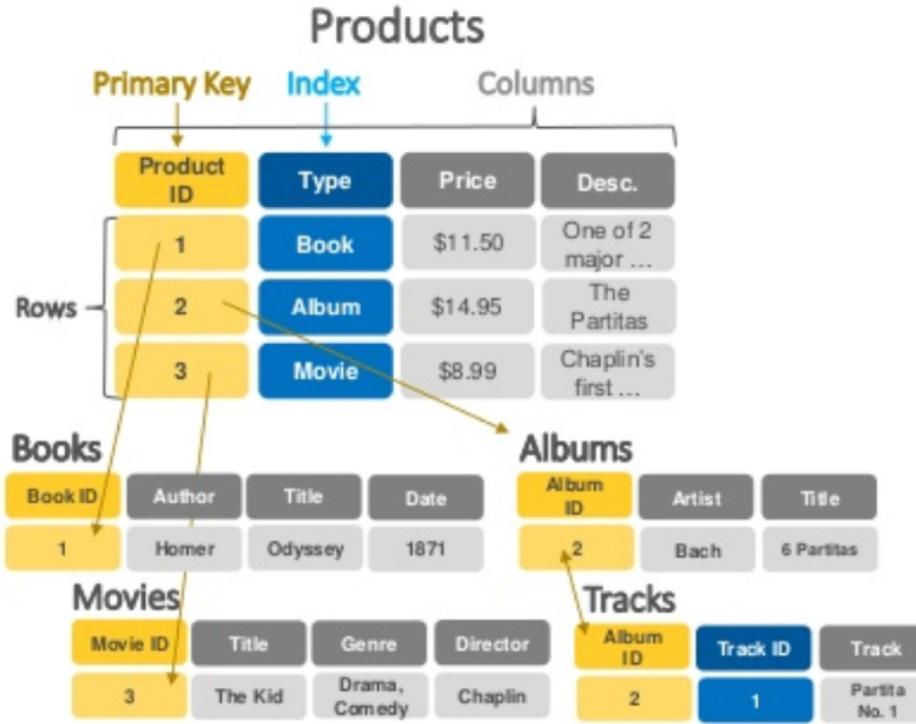
The screenshot shows the Facebook Developers Dashboard for the 'ColumbiaCourse' app. At the top, it displays the APP ID (317728141922775) and a 'View Analytics' link. On the left, a sidebar menu includes 'Dashboard', 'Settings', 'Roles', 'Alerts', 'App Review', 'PRODUCTS', 'Facebook Login', and '+ Add Product'. Below the sidebar is a 'facebook for developers' logo. The main dashboard area has a header 'Dashboard' and features a large 'ColumbiaCourse' logo with a green 'o'. It states: 'This app is in development mode and can only be used by app admins, developers and testers [?]' and 'API Version [?] v2.7'. It also shows the 'App ID' (317728141922775) and 'App Secret' (redacted). A 'Get Started with the Facebook SDK' section provides quick start guides for iOS, Android, and Facebook Web Game/website, with a 'Choose Platform' button. Another section for 'Facebook Analytics' includes a 'Set up Analytics' link and a 'Try Demo' button. A 'View Quickstart Guide' button is also present.

# *DynamoDB: A Little More*

# DynamoDB Data Model

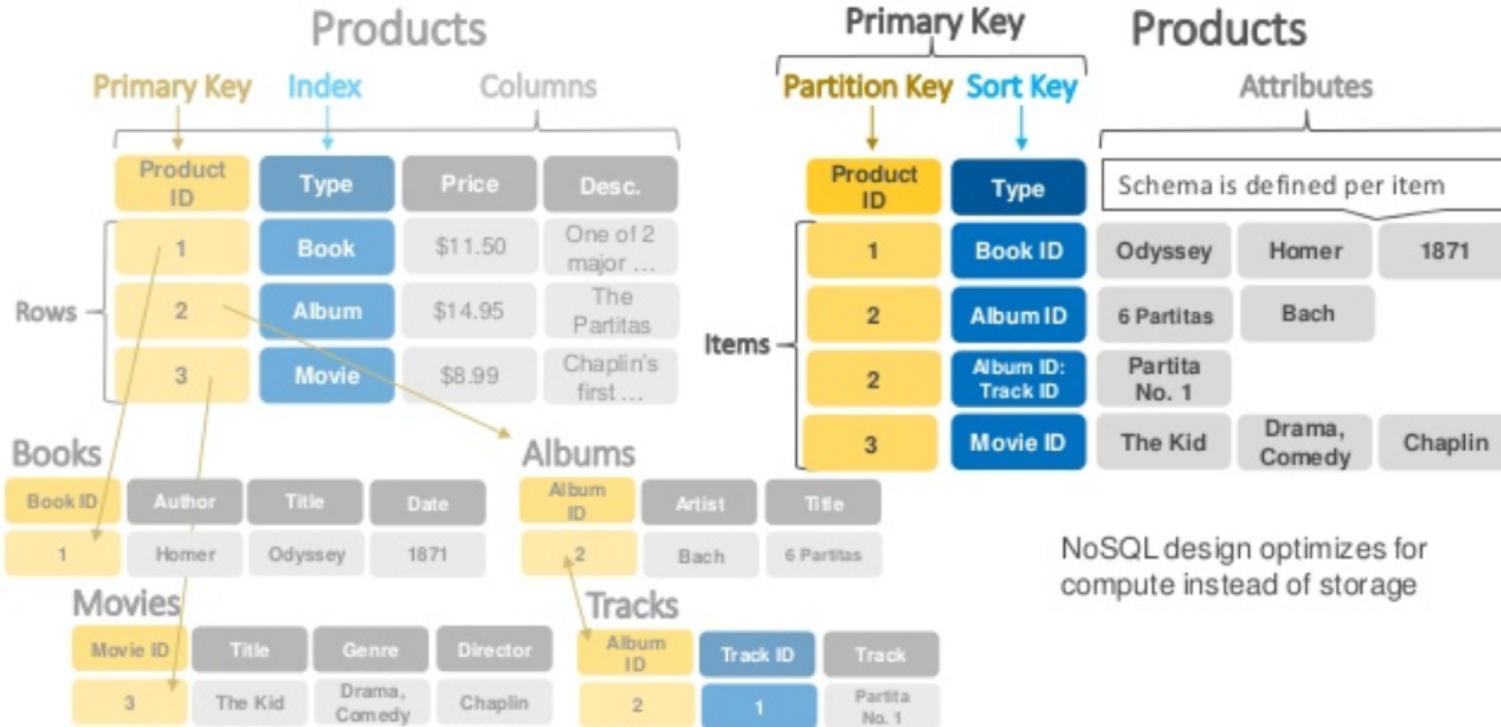
## SQL (Relational)

<https://www.slideshare.net/AmazonWebServices/introduction-to-amazon-dynamodb-73191648>



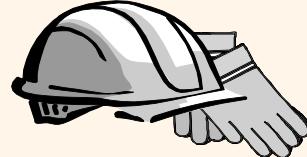
# DynamoDB Data Model

## SQL (Relational) vs. NoSQL (Non-relational)



# Normalization/De-normalization

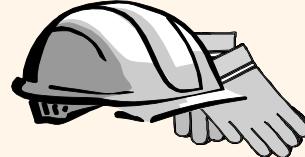
# *Normal Forms*



# *Schema Refinement and Normal Forms*

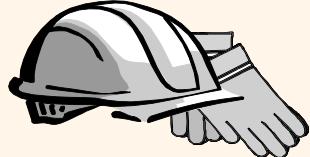
## Chapter 19

- There is a theoretical/formal basis for normalization in the relational model.
- Has practical application to real scenarios and data models.
- Primarily applied intuitively and based on common sense, not theory.
- But, like relational algebra, I have to give you an overview of the theory.



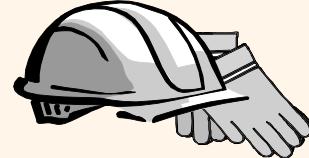
# The Evils of Redundancy

- ❖ *Redundancy* is at the root of several problems associated with relational schemas:
  - redundant storage, insert/delete/update anomalies
- ❖ Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
- ❖ Main refinement technique: decomposition (replacing ABCD with, say, AB and BCD, or ACD and ABD).
- ❖ Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?



# Functional Dependencies (FDs)

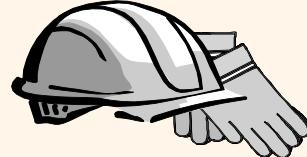
- ❖ A functional dependency  $X \rightarrow Y$  holds over relation  $R$  if, for every allowable instance  $r$  of  $R$ :
  - $t1 \in r, t2 \in r, \pi_X(t1) = \pi_X(t2)$  implies  $\pi_Y(t1) = \pi_Y(t2)$
  - i.e., given two tuples in  $r$ , if the  $X$  values agree, then the  $Y$  values must also agree. ( $X$  and  $Y$  are *sets* of attributes.)
- ❖ An FD is a statement about *all* allowable relations.
  - Must be identified based on semantics of application.
  - Given some allowable instance  $r1$  of  $R$ , we can check if it violates some FD  $f$ , but we cannot tell if  $f$  holds over  $R$ !
- ❖  $K$  is a candidate key for  $R$  means that  $K \rightarrow R$ 
  - However,  $K \rightarrow R$  does not require  $K$  to be *minimal*!



# Example: Constraints on Entity Set

- ❖ Consider relation obtained from Hourly\_Emps:
  - Hourly\_Emps (*ssn, name, lot, rating, hrly\_wages, hrs\_worked*)
- ❖ Notation: We will denote this relation schema by listing the attributes: **SNLRWH**
  - This is really the *set* of attributes {S,N,L,R,W,H}.
  - Sometimes, we will refer to all attributes of a relation by using the relation name. (e.g., Hourly\_Emps for SNLRWH)
- ❖ Some FDs on Hourly\_Emps:
  - *ssn* is the key:  $S \rightarrow \text{SNLRWH}$
  - *rating* determines *hrly\_wages*:  $R \rightarrow W$

# Example (Contd.)



- ❖ Problems due to  $R \rightarrow W$ :
  - Update anomaly: Can we change  $W$  in just the 1st tuple of SNLRWH?
  - Insertion anomaly: What if we want to insert an employee and don't know the hourly wage for his rating?
  - Deletion anomaly: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

Hourly\_Emps2

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

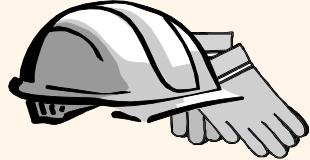
Wages

R	W
8	10
5	7

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Will 2 smaller tables be better?

# Example: Constraints on Entity Set

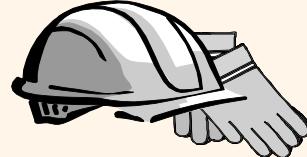


- ❖ Consider relation obtained from Hourly\_Emps:
  - Hourly\_Emps (ssn, name, lot, rating, hrly\_wages, hrs\_worked)
- ❖ Notation: We will denote this relation schema by  
~~Hourly\_Emps(ssn, name, lot, rating, hrly\_wages, hrs\_worked)~~

A more intuitive example – Consider a table of addresses.

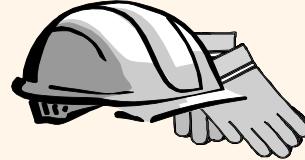
- Addresses(id, street\_no, street\_name, city, country, zipcode)
- (city, country) are *functionally dependent* on *zipcode*.

- ❖ Some FDs on Hourly\_Emps:
  - *ssn is the key:*  $S \rightarrow SNLRWH$
  - *rating determines hrly\_wages:*  $R \rightarrow W$



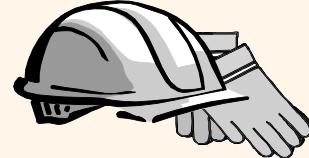
# Reasoning About FDs

- ❖ Given some FDs, we can usually infer additional FDs:
  - $ssn \rightarrow did$ ,  $did \rightarrow lot$  implies  $ssn \rightarrow lot$
- ❖ An FD  $f$  is implied by a set of FDs  $F$  if  $f$  holds whenever all FDs in  $F$  hold.
  - $F^+ = \text{closure of } F$  is the set of all FDs that are implied by  $F$ .
- ❖ Armstrong's Axioms ( $X, Y, Z$  are sets of attributes):
  - Reflexivity: If  $X \subseteq Y$ , then  $Y \rightarrow X$
  - Augmentation: If  $X \rightarrow Y$ , then  $XZ \rightarrow YZ$  for any  $Z$
  - Transitivity: If  $X \rightarrow Y$  and  $Y \rightarrow Z$ , then  $X \rightarrow Z$
- ❖ These are *sound* and *complete* inference rules for FDs!



# Normal Forms

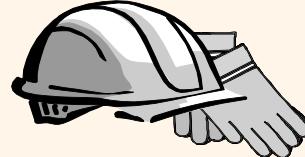
- ❖ Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- ❖ If a relation is in a certain *normal form* (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.
- ❖ Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No FDs hold: There is no redundancy here.
    - Given A     B: Several tuples could have the same A value, and if so, they'll all have the same B value!



# Boyce-Codd Normal Form (BCNF)

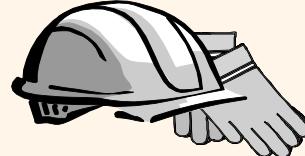
- ❖ Reln R with FDs  $F$  is in BCNF if, for all  $X \rightarrow A$  in  $F^+$ 
  - $A \in X$  (called a *trivial* FD), or
  - $X$  contains a key for R.
- ❖ In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints.
  - No dependency in R that can be predicted using FDs alone.
  - If we are shown two tuples that agree upon the X value, we cannot infer the A value in one tuple from the A value in the other.
  - If example relation is in BCNF, the 2 tuples must be identical (since X is a key).

X	Y	A
x	y1	a
x	y2	?



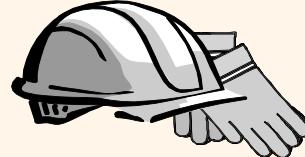
# Third Normal Form (3NF)

- ❖ Reln R with FDs  $F$  is in **3NF** if, for all  $X \rightarrow A$  in  $F^+$ 
  - $A \in X$  (called a *trivial FD*), or
  - $X$  contains a key for R, or
  - A is part of some key for R.
- ❖ *Minimality* of a key is crucial in third condition above!
- ❖ If R is in BCNF, obviously in 3NF.
- ❖ If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no “good” decompr, or performance considerations).
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*



# Example Decomposition

- ❖ Decompositions should be used only when needed.
  - SNLRWH has FDs  $S \rightarrow S$  and  $R \rightarrow W$
  - Second FD causes violation of 3NF;  $W$  values repeatedly associated with  $R$  values. Easiest way to fix this is to create a relation  $RW$  to store these associations, and to remove  $W$  from the main schema:
    - i.e., we decompose  $SNLRWH$  into  $SNLRH$  and  $RW$
- ❖ The information to be stored consists of  $SNLRWH$  tuples. If we just store the projections of these tuples onto  $SNLRH$  and  $RW$ , are there any potential problems that we should be aware of?



# Problems with Decompositions

- ❖ There are three potential problems to consider:
  - Some queries become more expensive.
    - e.g., How much did sailor Joe earn? ( $\text{salary} = \text{W} * \text{H}$ )
  - Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
    - Fortunately, not in the SNLRWH example.
  - Checking some dependencies may require joining the instances of the decomposed relations.
    - Fortunately, not in the SNLRWH example.
- ❖ Tradeoff: Must consider these issues vs. redundancy.

# *Denormalization Example*

# Classic Models (Cars) Database

<http://www.mysqltutorial.org/mysql-sample-database.aspx>

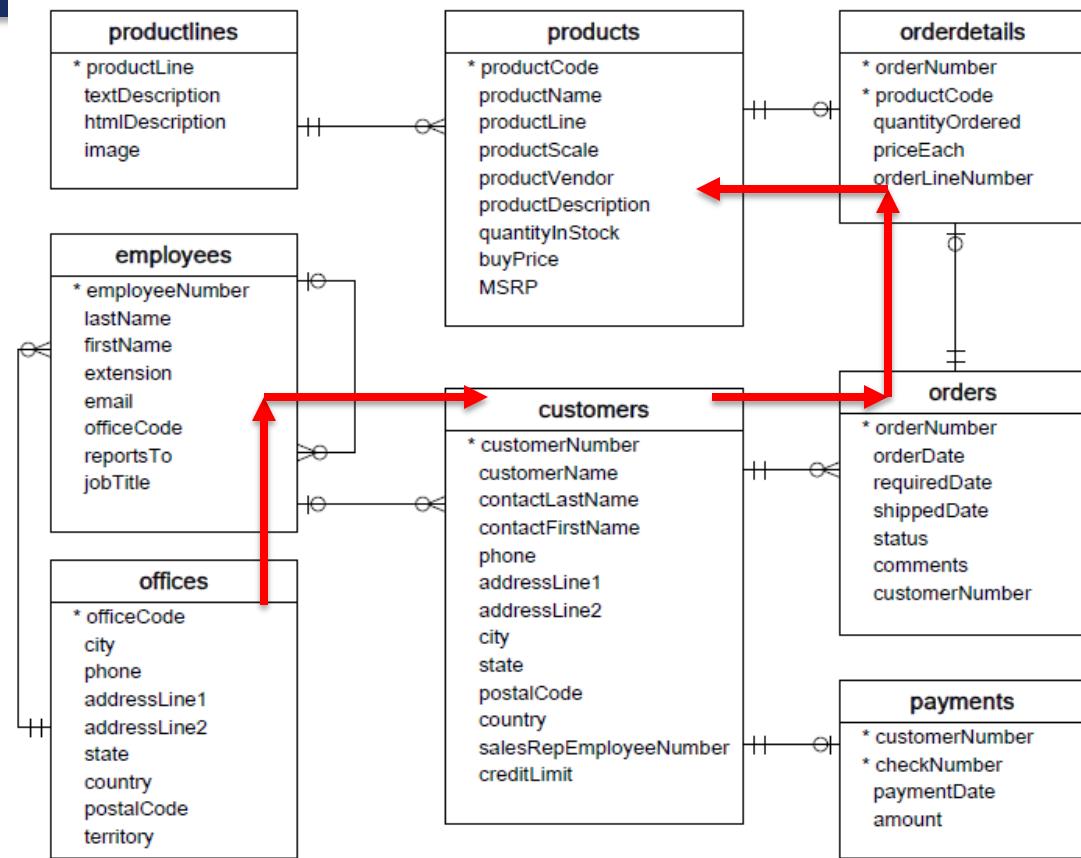
The classic models (cars) database we have seen has the following tables:

- **Customers**: stores customer's data.
- **Products**: stores a list of scale model cars.
- **ProductLines**: stores a list of product line categories.
- **Orders**: stores sales orders placed by customers.
- **OrderDetails**: stores sales order line items for each sales order.
- **Payments**: stores payments made by customers based on their accounts.
- **Employees**: stores all employee information as well as the organization structure such as who reports to whom.
- **Offices**: stores sales office data.

# Normalized Schema

- The normalization helps prevent update errors:
  - I cannot create an order without a valid customer no.
  - An order cannot have an invalid product ID.
  - etc.
- But some interesting data insight questions require complicated joins,

“Please show me the total number of instances of products from vendor XXX that employees in office YYY have sold.”

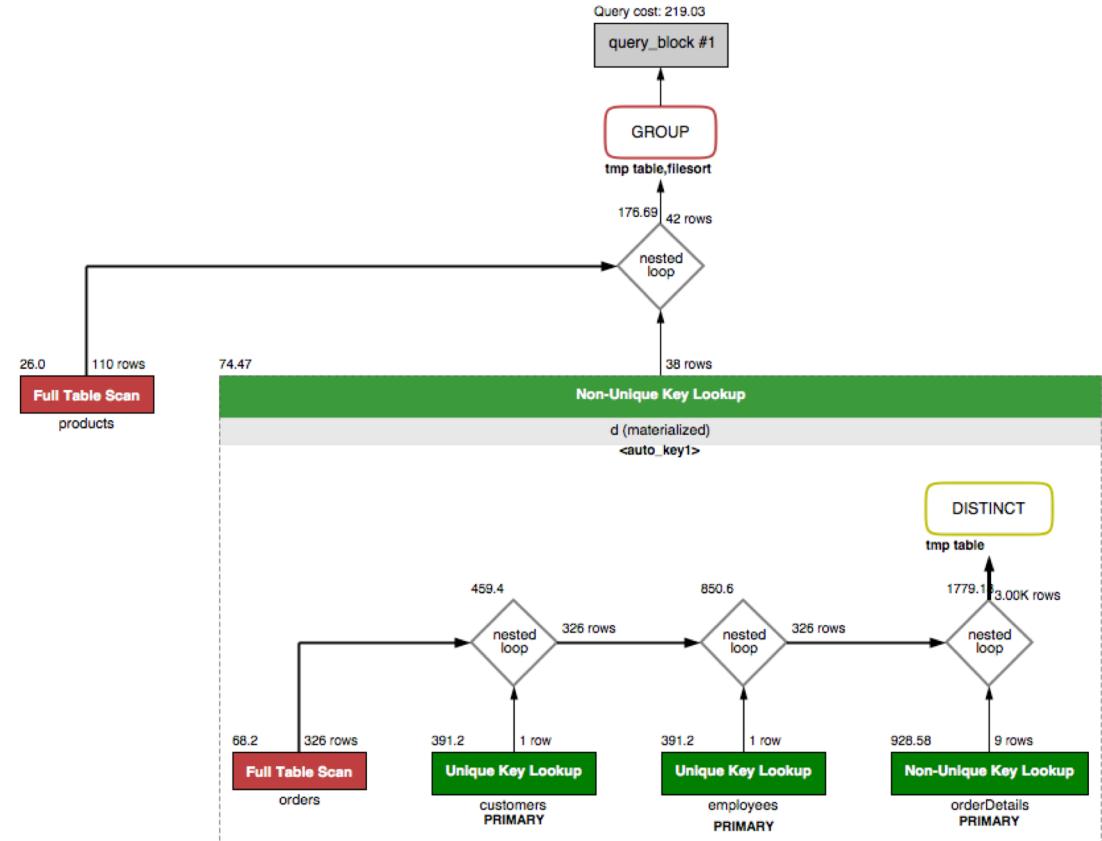


# Potential Query

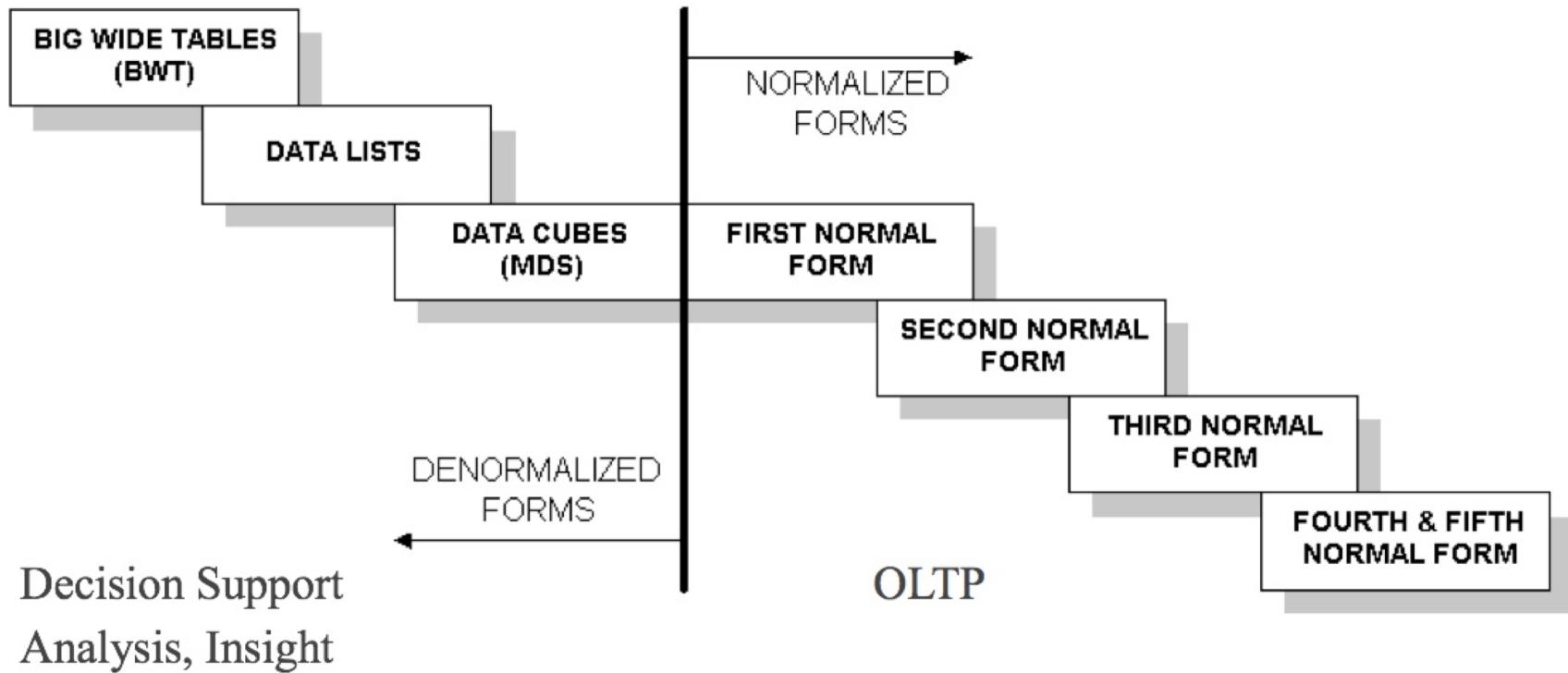
```
select officeCode, productVendor, productCode, sum(quantityOrdered) from
    (select employeeNumber, officeCode, customerNumber, orderNumber, orderLineNumber,
        quantityOrdered, productCode, productVendor
    from (select * from
        (
            SELECT distinct * FROM
                (
                    select * from
                        (select * from employees join customers on
                            salesRepEmployeeNumber=employees.employeeNumber) as a
                        join
                            orders using(customerNumber)
                ) as b
                join
                    orderDetails using(orderNumber)) as d) as e
            join
                products using(productCode)
            where officeCode=3 and productVendor='Motor City Art Classics') as f
        group by officeCode, productCode;
```

# Execution Plan

- I am not completely sure that query is correct.
- I am sure of two things, however:
  - I do not want to write queries like this over and over for each question.
  - The DB engine really, really does not like running queries like this one.



# Decision Support versus OLTP



# Fact Table

customerNumber	orderNumber	status	quantityOrdered	orderLineNumber	priceEach	MSRP	buyPrice	productCode	salesRepEmployeeNumber	geo_dimension_id	shipped_dimension_id	ordered_dimension_id	product_dimension_id
363	10100	Shipped	30	3	136.00	170.00	86.70	S18_1749	1216	78	3	1	89
363	10100	Shipped	50	2	55.09	60.54	33.30	S18_2248	1216	78	3	1	90
363	10100	Shipped	22	4	75.46	92.03	43.26	S18_4409	1216	78	3	1	99
363	10100	Shipped	49	1	35.29	41.03	21.75	S24_3969	1216	78	3	1	107
128	10101	Shipped	25	4	108.06	127.13	58.48	S18_2325	1504	7	266	2	91
128	10101	Shipped	26	1	167.06	168.75	72.56	S18_2795	1504	7	266	2	92
128	10101	Shipped	45	3	32.53	33.19	22.57	S24_1937	1504	7	266	2	102
128	10101	Shipped	46	2	44.35	44.80	20.61	S24_2022	1504	7	266	2	103
181	10102	Shipped	39	2	95.55	102.74	60.62	S18_1342	1286	9	267	3	87
181	10102	Shipped	41	1	43.13	53.91	24.26	S18_1367	1286	9	267	3	88
121	10103	Shipped	26	11	214.30	214.30	98.58	S10_1949	1504	4	268	4	1
121	10103	Shipped	42	4	119.67	147.74	103.42	S10_4962	1504	4	268	4	3
121	10103	Shipped	27	8	121.64	136.67	77.90	S12_1666	1504	4	268	4	76
121	10103	Shipped	35	10	94.50	116.67	58.33	S18_1097	1504	4	268	4	78
121	10103	Shipped	22	2	58.34	60.77	24.92	S18_2432	1504	4	268	4	80
121	10103	Shipped	27	12	92.19	101.31	60.78	S18_2949	1504	4	268	4	93

Field	Type
customerNumber	int(11)
orderNumber	int(11)
status	varchar(15)
quantityOrdered	int(11)
orderLineNumber	smallint(6)
priceEach	decimal(10,2)
MSRP	decimal(10,2)
buyPrice	decimal(10,2)
productCode	varchar(15)
salesRepEmployeeNumber	int(11)
geo_dimension_id	int(11)
shipped_dimension_id	int(11)
ordered_dimension_id	int(11)
product_dimension_id	int(11)

- Fact
  - What was sold.
  - To whom.
  - For how much.
  - And when.
- Dimensions tell you how to position in space, time and your product inventory.

# Classic Cars Facts

```
drop table if exists customer_order_temp;
create temporary table customer_order_temp as
    select * from customers join orders using(customerNumber);

drop table if exists customer_order_details_temp;
create table customer_order_details_temp as
    select * from customer_order_temp join orderdetails
        using (orderNumber);

drop table if exists customer_order_product_facts;
create table customer_order_product_facts as
    select * from
        products join customer_order_details_temp
            using (productCode);

drop table customer_order_temp;
drop table customer_order_details_temp;
```

- Pre-compute the joins and make a copy of the data.
- Can ask questions using SELECT, GROUP BY and WHERE.

Field	Type	Null	Key	Default	Extra
productCode	varchar(15)	NO		NULL	
productName	varchar(70)	NO		NULL	
productLine	varchar(50)	NO		NULL	
productScale	varchar(10)	NO		NULL	
productVendor	varchar(50)	NO		NULL	
productDescription	text	NO		NULL	
quantityInStock	smallint(6)	NO		NULL	
buyPrice	decimal(10,2)	NO		NULL	
MSRP	decimal(10,2)	NO		NULL	
orderNumber	int(11)	NO		NULL	
customerNumber	int(11)	NO		NULL	
customerName	varchar(50)	NO		NULL	
contactLastName	varchar(50)	NO		NULL	
contactFirstName	varchar(50)	NO		NULL	
phone	varchar(50)	NO		NULL	
addressLine1	varchar(50)	NO		NULL	
addressLine2	varchar(50)	YES		NULL	
city	varchar(50)	NO		NULL	
state	varchar(50)	YES		NULL	
postalCode	varchar(15)	YES		NULL	
country	varchar(50)	NO		NULL	
salesRepEmployeeNumber	int(11)	YES		NULL	
creditLimit	decimal(10,2)	YES		NULL	
orderDate	date	NO		NULL	
requiredDate	date	NO		NULL	
shippedDate	date	YES		NULL	
status	varchar(15)	NO		NULL	
comments	text	YES		NULL	
quantityOrdered	int(11)	NO		NULL	
priceEach	decimal(10,2)	NO		NULL	
orderLineNumber	smallint(6)	NO		NULL	

Orders

orderNumber	orderDate	customerNumber	orderPriority
10223	2004-02-20	114	1
10361	2004-12-17	282	1
10263	2004-06-28	175	1
10388	2005-03-03	462	1
10309	2004-10-15	121	1
10134	2003-07-01	250	1
10285	2004-08-27	286	1
10201	2003-12-01	129	1
10168	2003-10-28	161	1
10107	2003-02-24	131	1
10375	2005-02-03	119	1
10251	2004-05-18	328	1
10275	2004-07-23	119	1
10121	2003-05-07	353	1
10237	2004-04-05	181	1
10329	2004-11-15	131	1
10211	2004-01-15	406	1
10341	2004-11-24	382	1

OrdersDetails

productCode	quantityOrder	priceEach
S10_1678	37	80.39
S10_1678	20	92.83
S10_1678	34	89.00
S10_1678	42	80.39
S10_1678	41	94.74
S10_1678	41	90.92
S10_1678	36	95.70
S10_1678	22	82.30
S10_1678	36	94.74
S10_1678	30	81.35
S10_1678	21	76.56
S10_1678	59	93.79
S10_1678	45	81.35
S10_1678	34	86.13
S10_1678	23	91.87
S10_1678	42	80.39
S10_1678	41	90.92
S10_1678	41	84.22

Products

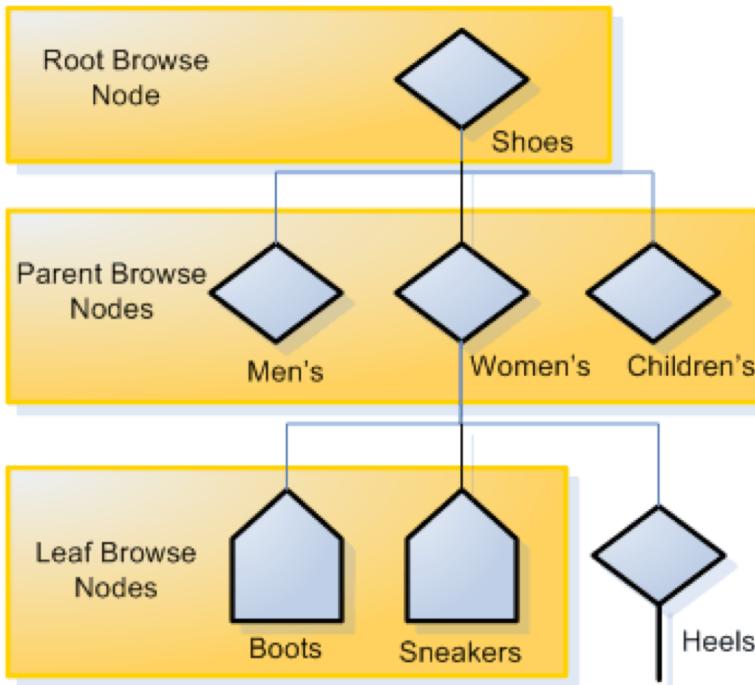
MSRP	buyPrice
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81

Computed

discount	markUp
0.1866	0.6556
0.0323	0.9015
0.0787	0.8195
0.1866	0.6556
0.0106	0.9424
0.0550	0.8605
0.0000	0.9629
0.1580	0.6761
0.0106	0.9424
0.1721	0.6761
0.2482	0.5737
0.0213	0.9219
0.1721	0.6761
0.1161	0.7580
0.0435	0.8810
0.1866	0.6556
0.0550	0.8605
0.1306	0.7171

## Amazon Hierarchy (API)

(<https://docs.aws.amazon.com/AWSECommerceService/latest/DG/BrowseNodes.html>)



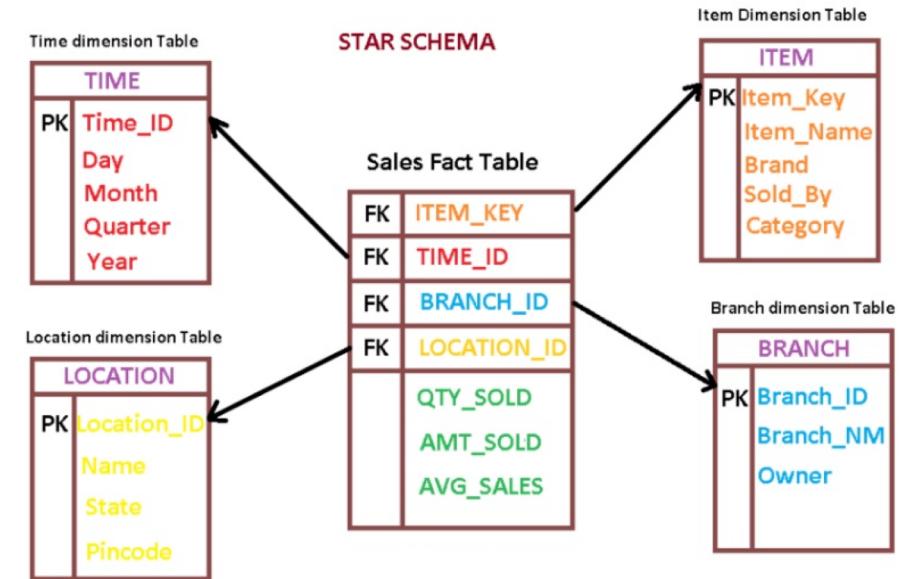
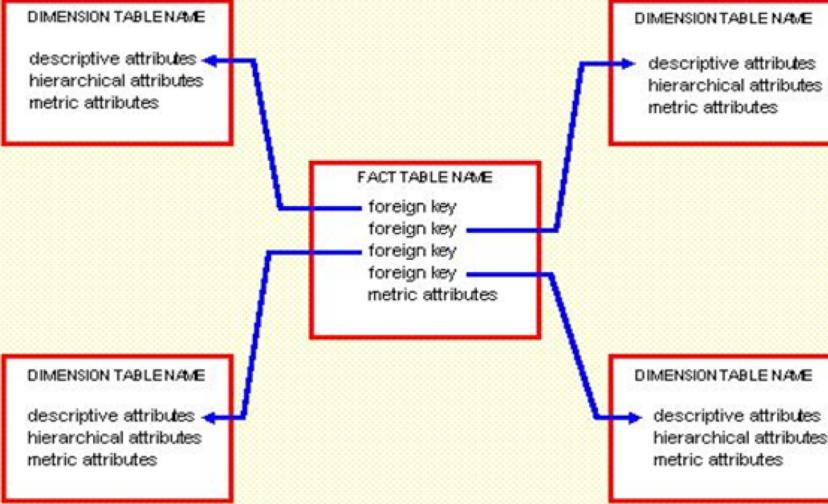
## Classic Cars “Dimensions”

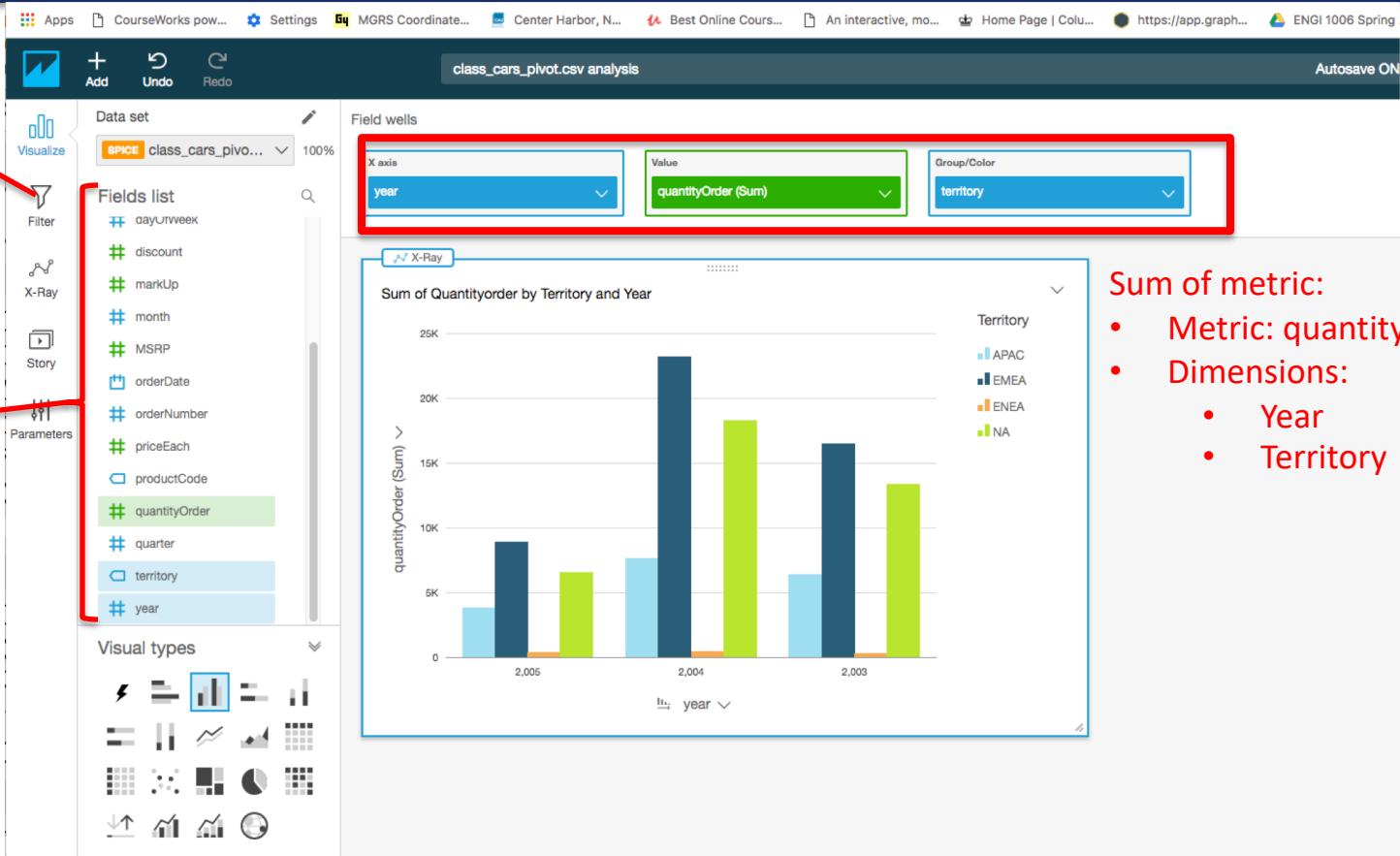
productLine
► Motorcycles
Classic Cars
Trucks and Buses
Vintage Cars
Planes
Ships
Trains

productLine	productScale
► Classic Cars	1:10
Classic Cars	1:12
Classic Cars	1:18
Classic Cars	1:24
Motorcycles	1:10
Motorcycles	1:12
Motorcycles	1:18
Motorcycles	1:24
Motorcycles	1:32
Motorcycles	1:50
Planes	1:18
Planes	1:72
Planes	1:24
Planes	1:700
Ships	1:18
Ships	1:24
Ships	1:700
Ships	1:72

## Star Schema Template





Sum of metric:

- Metric: `quantityOrdered`
- Dimensions:
  - Year
  - Territory

## Facts and derived facts

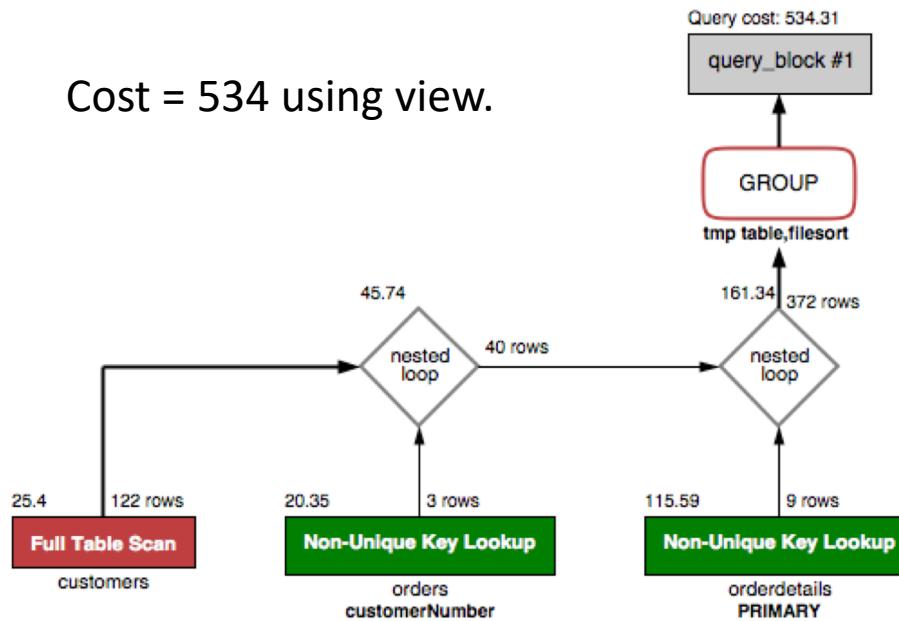
X11	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
	orderNumber	orderDate	customer	productCode	quant	priceEach	MSRP	buyPrice	discount	markUp	revenue	Unit Profit	Profit	year	quarter	month	dayOfWk	day	city	country	territory
1	10223	2/20/04	114	S10_1678	37	80.39	95.7	48.81	0.19	0.66	\$2,974.43	31.58	\$1,168.4	2004	1	2	6	2	Melbourne	Australia	APAC
2	10361	12/17/04	282	S10_1678	20	92.83	95.7	48.81	0.03	0.90	\$1,856.60	44.02	\$880.4	2004	4	12	6	1	Chatswood	Australia	APAC
3	10263	6/28/04	175	S10_1678	34	89	95.7	48.81	0.08	0.82	\$3,026.00	40.19	\$1,366.4	2004	2	6	2	2	San Rafael	USA	NA
4	10388	3/3/05	462	S10_1678	42	80.39	95.7	48.81	0.19	0.66	\$3,376.38	31.58	\$1,326.3	2005	1	3	5	1	New Bedford	USA	NA
5	10309	10/15/04	121	S10_1678	41	94.74	95.7	48.81	0.01	0.94	\$3,884.34	45.93	\$1,883.1	2004	3	10	6	1	Stavern	Norway	EMEA
6	10134	7/1/03	250	S10_1678	41	90.92	95.7	48.81	0.06	0.86	\$3,727.72	42.11	\$1,726.5	2003	2	7	3	1	Paris	France	EMEA
7	10285	8/27/04	286	S10_1678	36	95.7	95.7	48.81	0.00	0.96	\$3,445.20	46.89	\$1,688.0	2004	3	8	6	2	Cambridge	USA	NA
8	10201	12/1/03	129	S10_1678	22	82.3	95.7	48.81	0.16	0.68	\$1,810.60	33.49	\$736.7	2003	4	12	2	2	San Francisco	USA	NA
9	10168	10/28/03	161	S10_1678	36	94.74	95.7	48.81	0.01	0.94	\$3,410.64	45.93	\$1,653.4	2003	3	10	3	2	San Francisco	USA	NA
10	10375	2/3/05	119	S10_1678	21	76.56	95.7	48.81	0.25	0.57	\$1,607.76	27.75	\$582.7	2005	1	2	5	1	Nantes	France	EMEA
11	10107	2/24/03	131	S10_1678	30	81.35	95.7	48.81	0.17	0.68	\$2,440.50	32.54	\$976.2	2003	1	2	2	2	New York	USA	NA
12	10251	5/18/04	328	S10_1678	59	93.79	95.7	48.81	0.02	0.92	\$5,533.61	44.98	\$2,653.8	2004	2	5	3	1	Newark	USA	NA
13	10121	5/7/03	353	S10_1678	34	86.13	95.7	48.81	0.12	0.76	\$2,928.42	37.32	\$1,268.8	2003	2	5	4	1	Reims	France	EMEA
14	10275	7/23/04	119	S10_1678	45	81.35	95.7	48.81	0.17	0.68	\$3,660.75	32.54	\$1,464.3	2004	2	7	6	2	Nantes	France	EMEA
15	10237	4/5/04	181	S10_1678	23	91.87	95.7	48.81	0.04	0.88	\$2,113.01	43.06	\$990.3	2004	2	4	2	1	New York	USA	NA
16	10329	11/15/04	131	S10_1678	42	80.39	95.7	48.81	0.19	0.66	\$3,376.38	31.58	\$1,326.3	2004	3	11	2	1	New York	USA	NA
17	10211	1/15/04	406	S10_1678	41	90.92	95.7	48.81	0.06	0.86	\$3,727.72	42.11	\$1,726.5	2004	1	1	5	1	Paris	France	EMEA
18	10341	11/24/04	382	S10_1678	41	84.22	95.7	48.81	0.13	0.72	\$3,453.02	35.41	\$1,451.8	2004	3	11	4	2	Salzburg	Austria	EMEA
19	10417	5/13/05	141	S10_1678	66	79.43	95.7	48.81	0.20	0.64	\$5,242.38	30.62	\$2,020.9	2005	2	5	6	1	Madrid	Spain	EMEA
20	10354	12/4/04	323	S10_1678	42	84.22	95.7	48.81	0.13	0.72	\$3,537.24	35.41	\$1,487.2	2004	4	12	7	1	Auckland	New Zealand	APAC
21	10188	11/18/03	167	S10_1678	48	95.7	95.7	48.81	0.00	0.96	\$4,593.60	46.89	\$2,250.7	2003	3	11	3	1	Bergen	Norway	EMEA
22	10145	8/25/03	205	S10_1678	45	76.56	95.7	48.81	0.25	0.57	\$3,445.20	27.75	\$1,248.7	2003	3	8	2	2	Pasadena	USA	NA
23	10399	4/1/05	496	S10_1678	40	77.52	95.7	48.81	0.23	0.59	\$3,100.80	28.71	\$1,148.4	2005	2	4	6	1	Auckland	New Zealand	APAC
24	10403	4/8/05	201	S10_1678	24	85.17	95.7	48.81	0.13	0.74	\$2,044.08	36.36	\$872.6	2005	2	4	6	1	Liverpool	UK	EMEA
25	10318	11/2/04	157	S10_1678	46	84.22	95.7	48.81	0.13	0.72	\$3,874.12	35.41	\$1,628.8	2004	3	11	3	1	Allentown	USA	NA
26	10180	11/11/03	171	S10_1678	29	76.56	95.7	48.81	0.25	0.57	\$2,220.24	27.75	\$804.7	2003	3	11	3	1	Lille	France	EMEA
27	10159	10/10/03	321	S10_1678	49	81.35	95.7	48.81	0.17	0.68	\$3,986.15	32.54	\$1,594.4	2003	3	10	6	1	San Francisco	USA	NA
28	10299	9/30/04	186	S10_1678	23	76.56	95.7	48.81	0.25	0.57	\$1,760.88	27.75	\$638.2	2004	3	9	5	3	Helsinki	Finland	EMEA
29	10270	7/19/04	282	S10_1949	21	171.44	214.3	98.58	0.25	0.74	\$3,600.24	72.86	\$1,530.0	2004	2	7	2	1	Chatswood	Australia	APAC
30	10261	3/9/05	275	S10_1949	24	185.01	214.3	98.58	0.10	0.87	\$4,680.24	85.47	\$2,314.2	2005	1	3	4	1	North Sydney	Australia	APAC
31	10250	6/15/04	200	S10_1678	22	177.77	214.3	98.58	0.20	0.80	\$5,601.24	70.20	\$2,527.20	2004	3	6	3	1			

# *Backup*

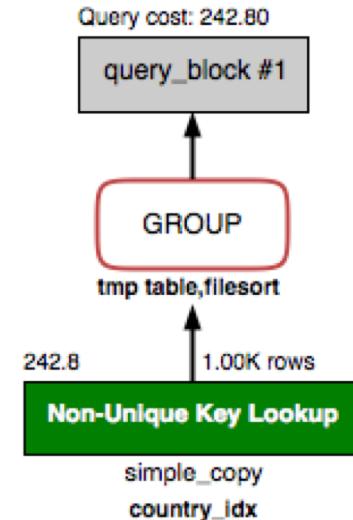
# Views versus Copy Tables

Compute the total product instances ordered by US customers, grouped by productCode.

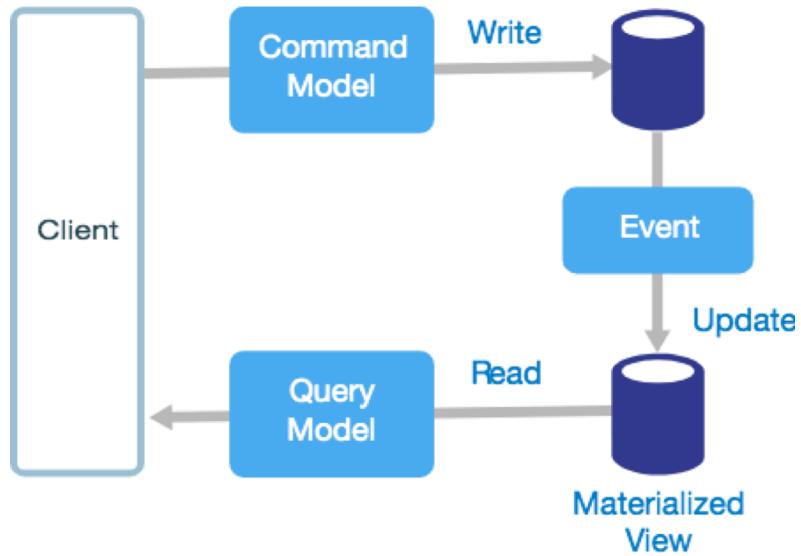
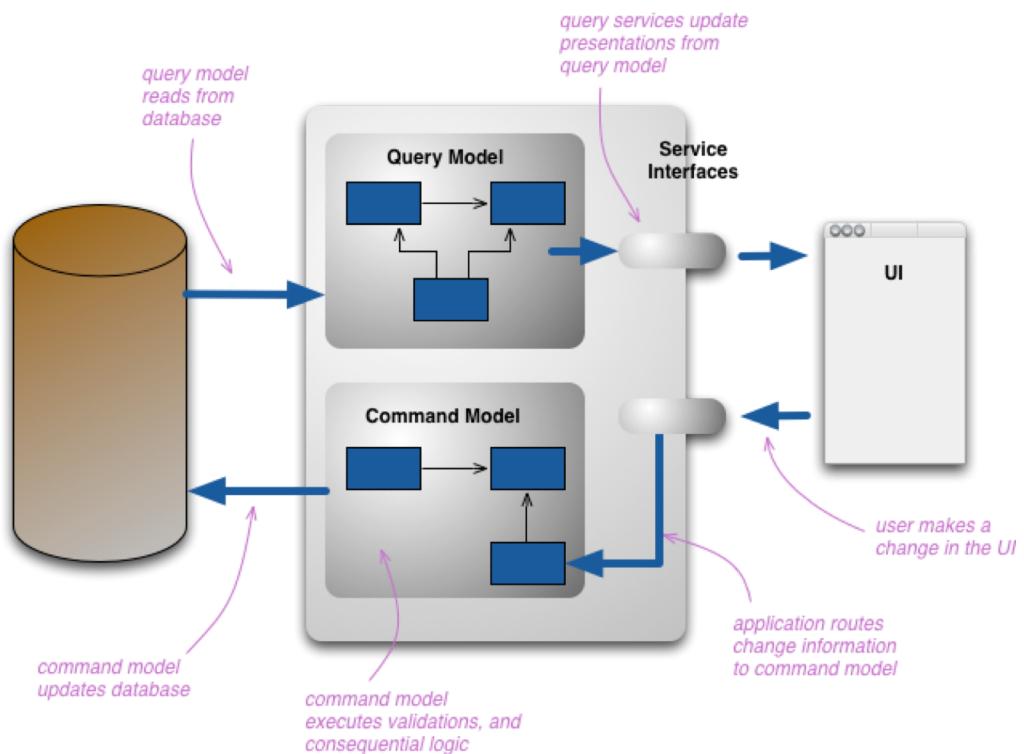
Cost = 534 using view.



Cost = 243 using copy.



# Command-Query Responsibility Segregation



<https://martinfowler.com/bliki/CQRS.html>

<https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/cqrs>