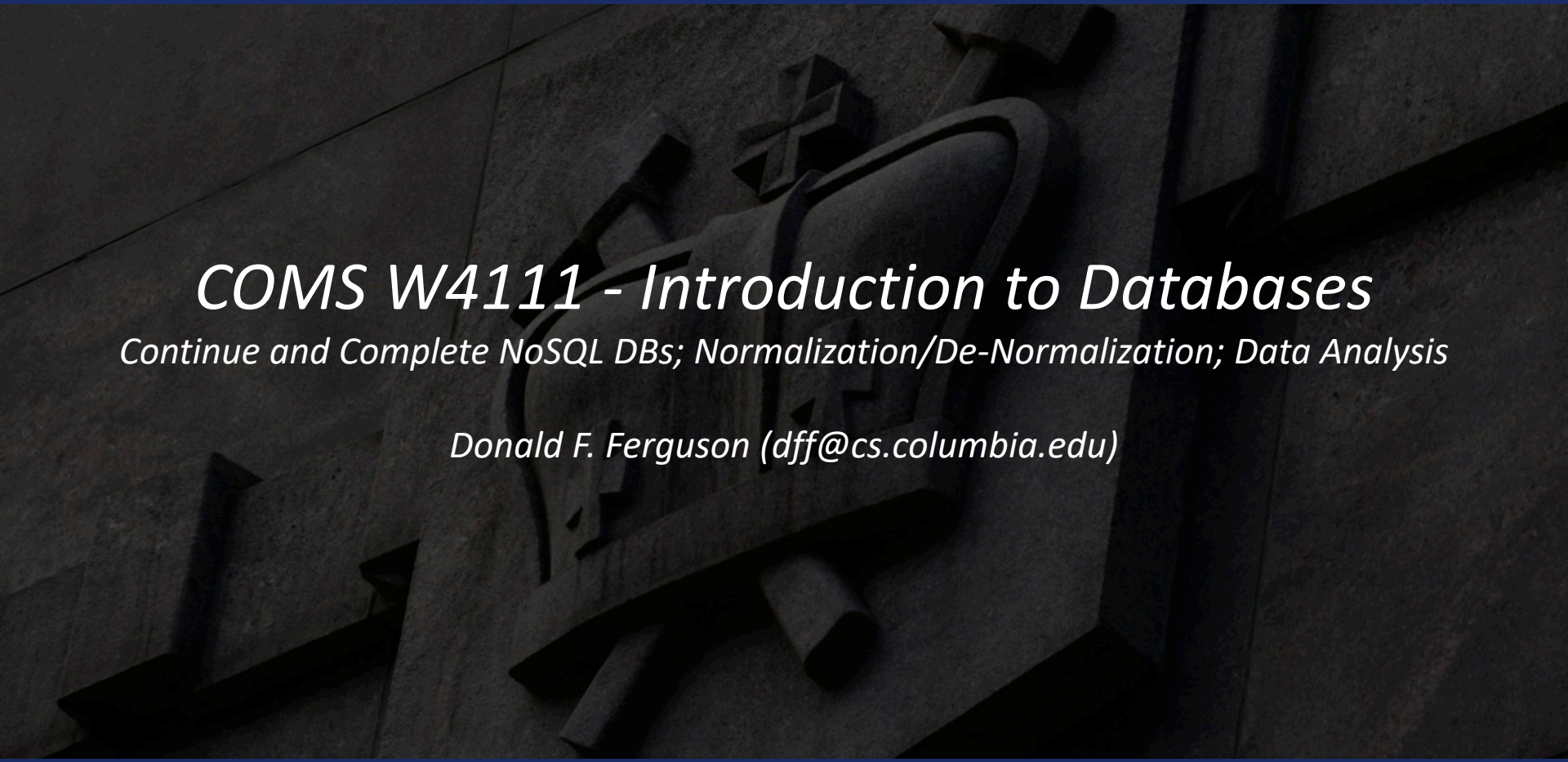


COMS W4111 - Introduction to Databases

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

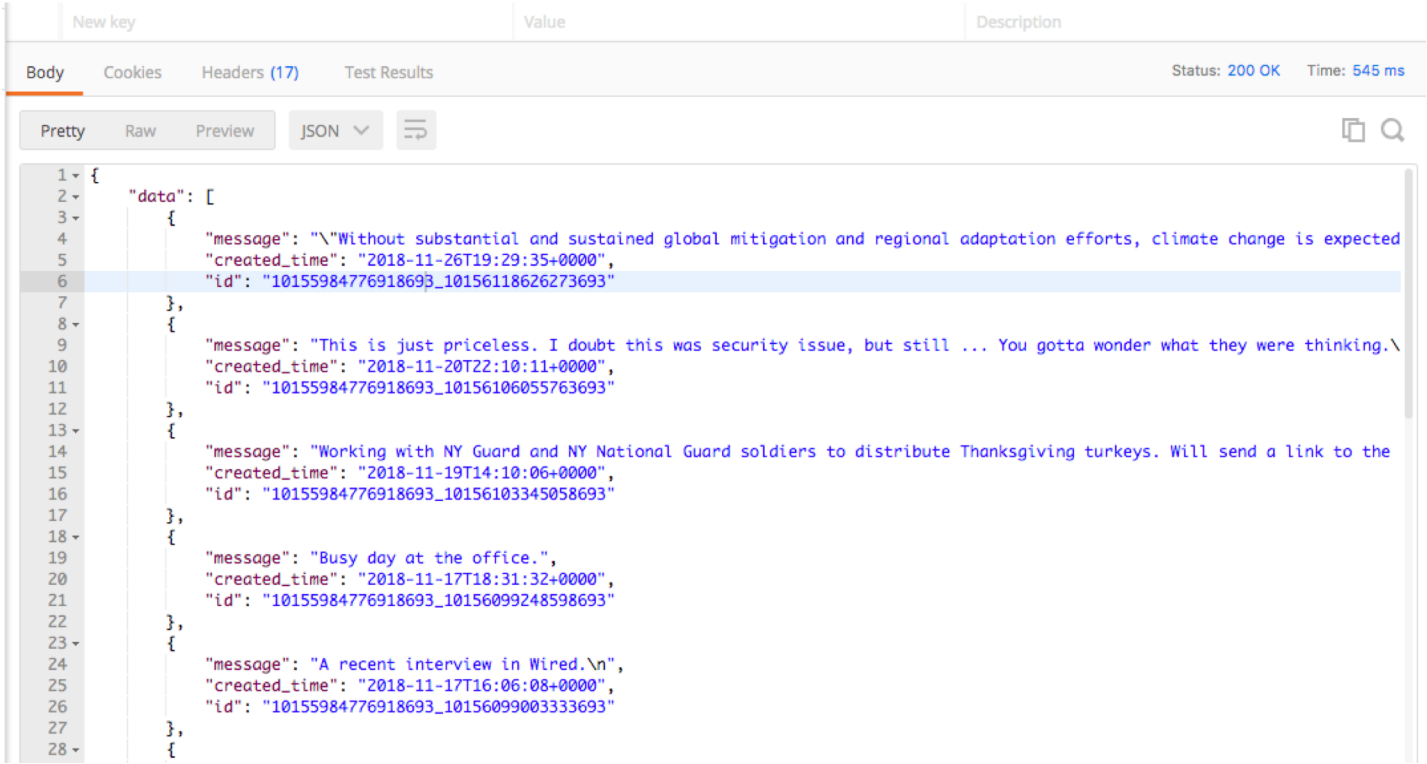
Full list at:
https://developers.facebook.com/docs/graph-api/reference

... ...

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 `native offer` represents an Offer on Facebook. The `/{offer_id}` node returns a single `native offer`. Each `native offer` requires a `view` to be rendered to users.
Notification	Graph API Reference Notification /notification
Object Comments	This reference describes the `/comments` 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=xxxxxxx



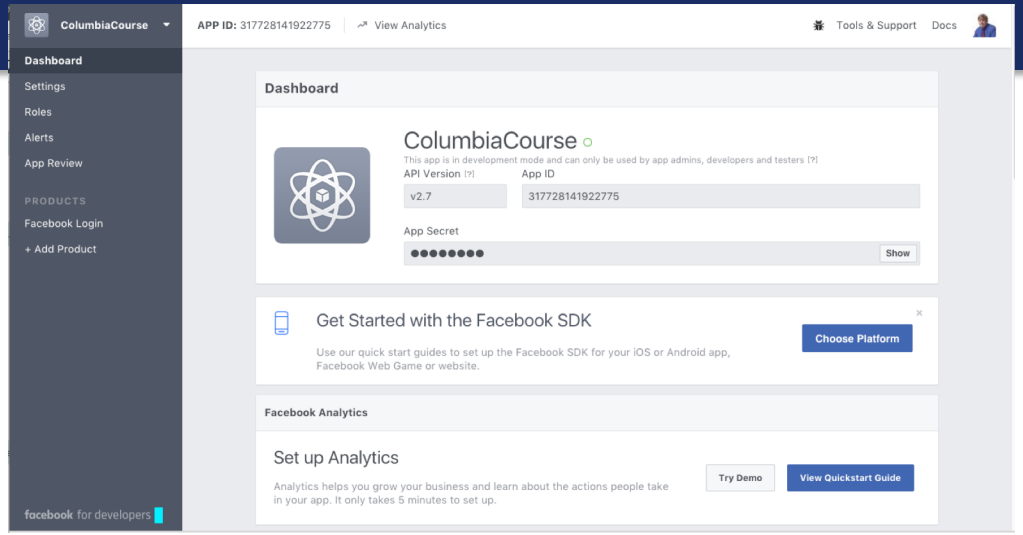

```
{
    "data": [
        {
            "message": "\"Without substantial and sustained global mitigation and regional adaptation efforts, climate change is expected
            "created_time": "2018-11-26T19:29:35+0000",
            "id": "10155984776918693_10156118626273693"
        },
        {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

Facebook Graph API – Examples

https://graph.facebook.com/v3.2/me/likes?access_token=xxxxxxx

Notice use of paging links.

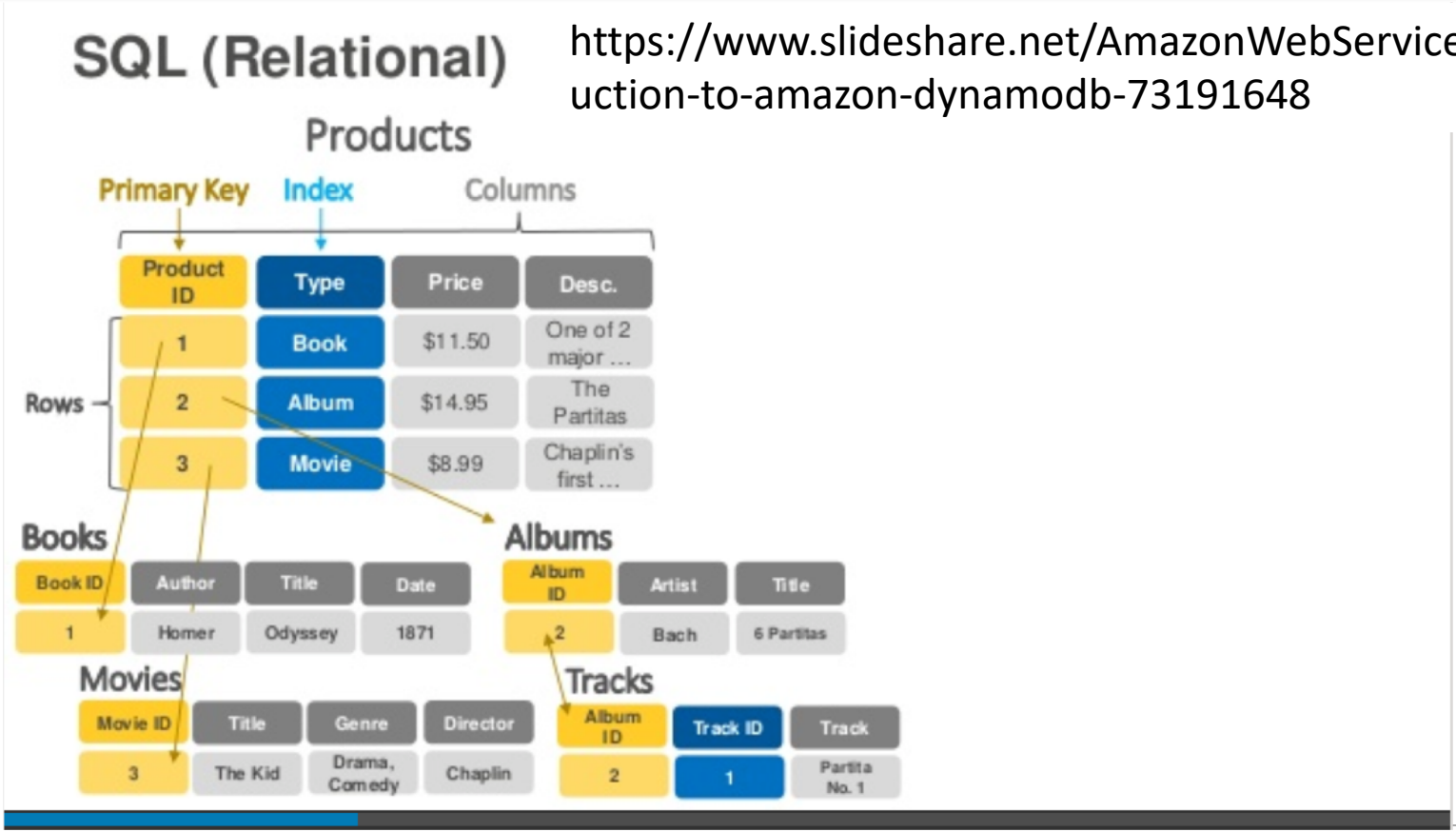
Facebook Applications



- Facebook Application
  - Define an application at 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.

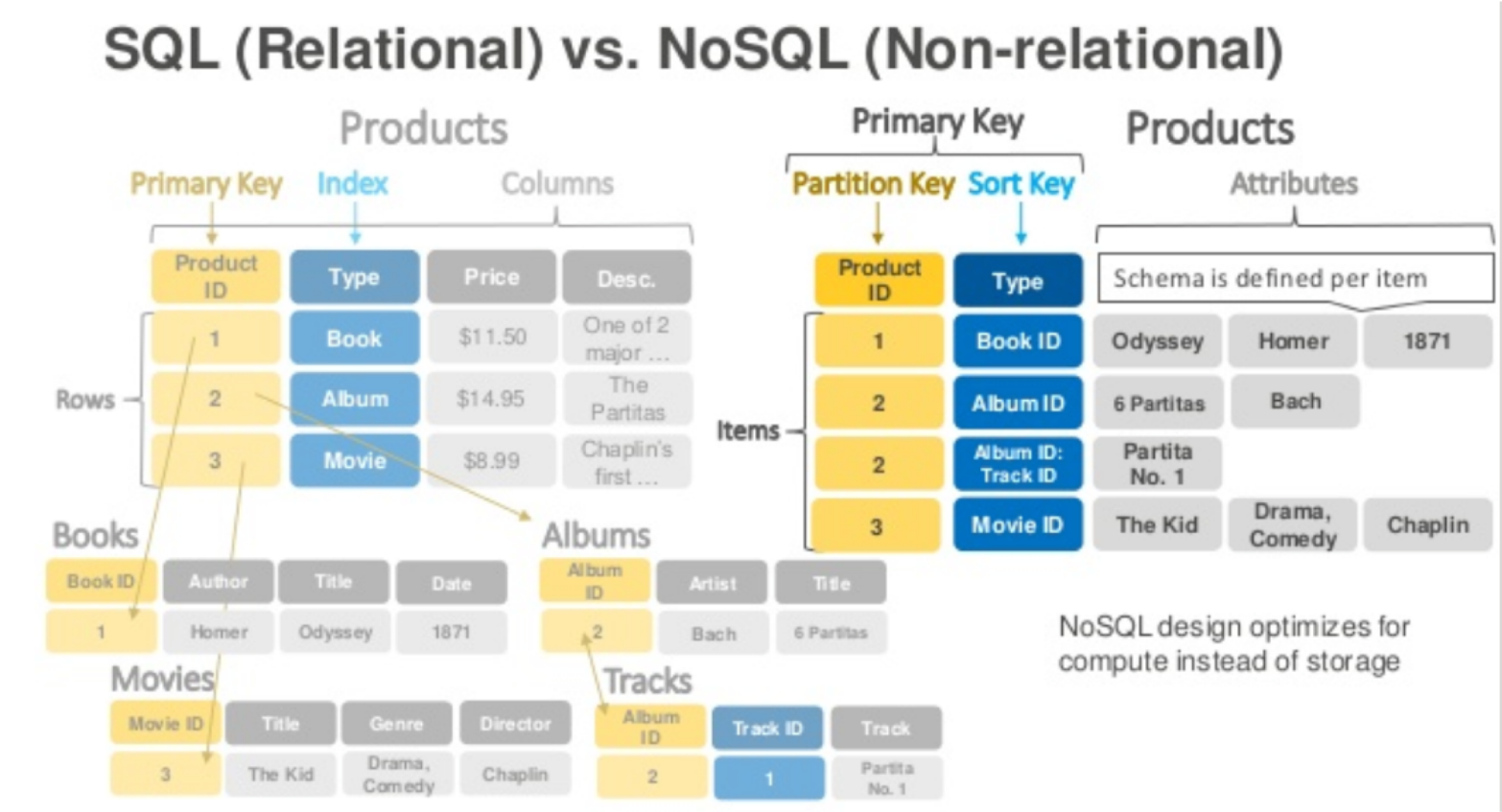
DynamoDB: A Little More

DynamoDB Data Model



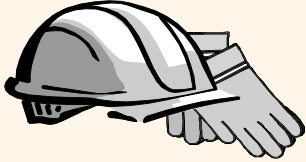
https://www.slideshare.net/AmazonWebServices/introduction-to-amazon-dynamodb-73191648

DynamoDB Data Model



Normalization/De-normalization

Normal Forms



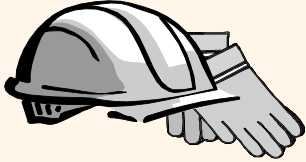
Schema Refinement and Normal Forms

Chapter 19

Note

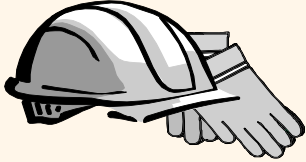
- There is a theoretical/formal basis for normalization in the relational model.
- Has practical application to real scenarios and data models.
- Primarily applied intuitively and based on common sense, not theory.
- But, like relational algebra, I have to give you an overview of the theory.

The Evils of Redundancy



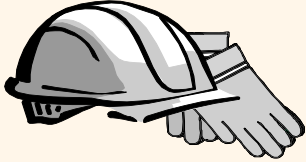
- *Redundancy* is at the root of several problems associated with relational schemas:
  - redundant storage, insert/delete/update anomalies
- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
- Main refinement 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)



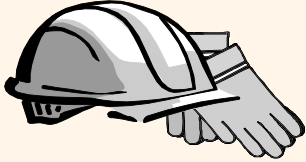
- A functional dependency X→ Y holds over relation R if, for every allowable instance *r* of R:
  - $t1 \in r,\ t2 \in r, \pi_X(t1) = \pi_X(t2)$ implies $\pi_Y(t1) = \pi_Y(t2)$
  - i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)
- 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→ R
  - However, K→ 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 → SNLRWH
  - *rating* determines *hrly_wages*: R → W

Example (Contd.)



Wages

R	W
8	10
5	7

Hourly_Emps2

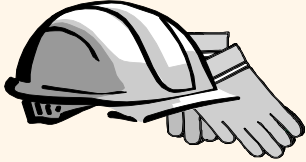
S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

- Problems due to R → 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!

Will 2 smaller tables be better?

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

Example: Constraints on Entity Set



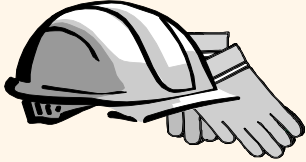
- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (*ssn, name, lot, rating, hrly_wages, hrs_worked*)
- *Notation*: We will denote this relation schema by

A more intuitive example – Consider a table of addresses.

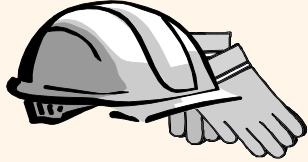
- Addresses(id, street_no, street_name, city, country, zipcode)
- (city, country) are *functionally dependent* on *zipcode.*

- Some FDs on Hourly_Emps:
  - *ssn* is the key: S → SNLRWH
  - *rating* determines *hrly_wages*: R → W

Reasoning About FDs



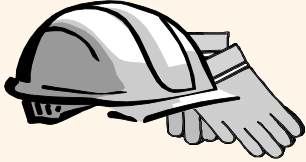
- Given some FDs, we can usually infer additional FDs:
  - *ssn* $\rightarrow$ *did*, *did* $\rightarrow$ *lot* implies *ssn* $\rightarrow$ *lot*
- An FD *f* is *implied by* a set of FDs *F* if *f* holds whenever all FDs in *F* hold.
  - F^+ = *closure of F* is the set of all FDs that are implied by *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

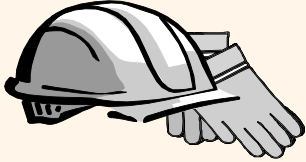
- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- If a relation is in a certain *normal form* (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation 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→ A in F^+
  - A ∈ 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).

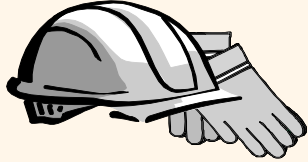
X	Y	A
x	y1	a
x	y2	?



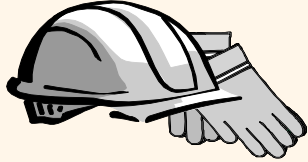
Third Normal Form (3NF)

- 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'' decomp, 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 → SNLRWH and R → 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? (salary = W*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

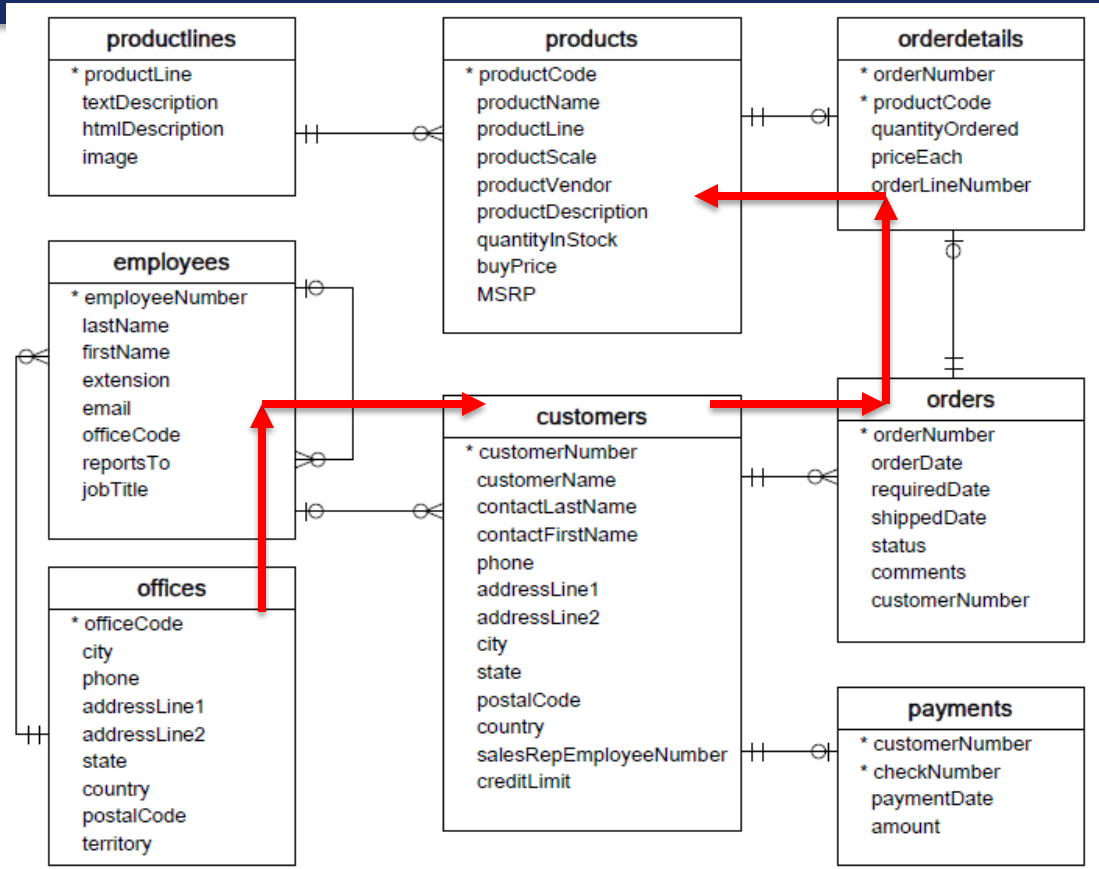
The classic models (cars) database we have seen has the following tables:

- **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.

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."



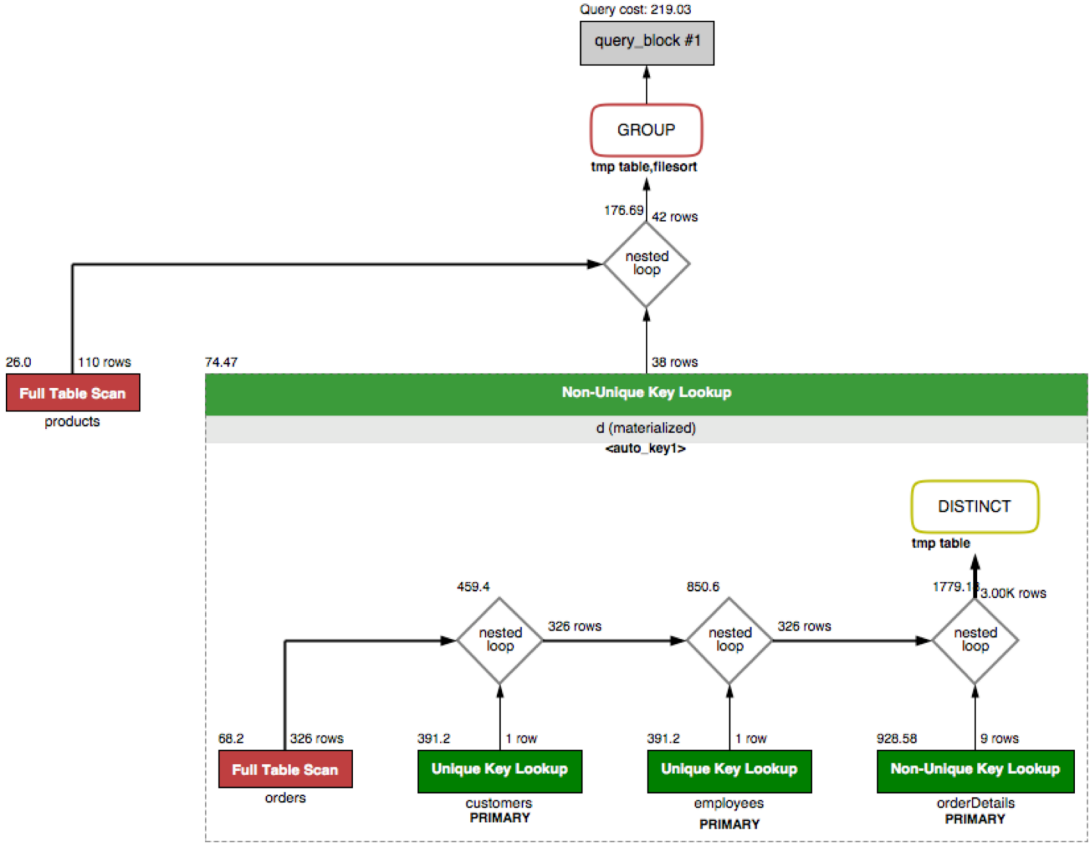
Potential Query

```
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;
```

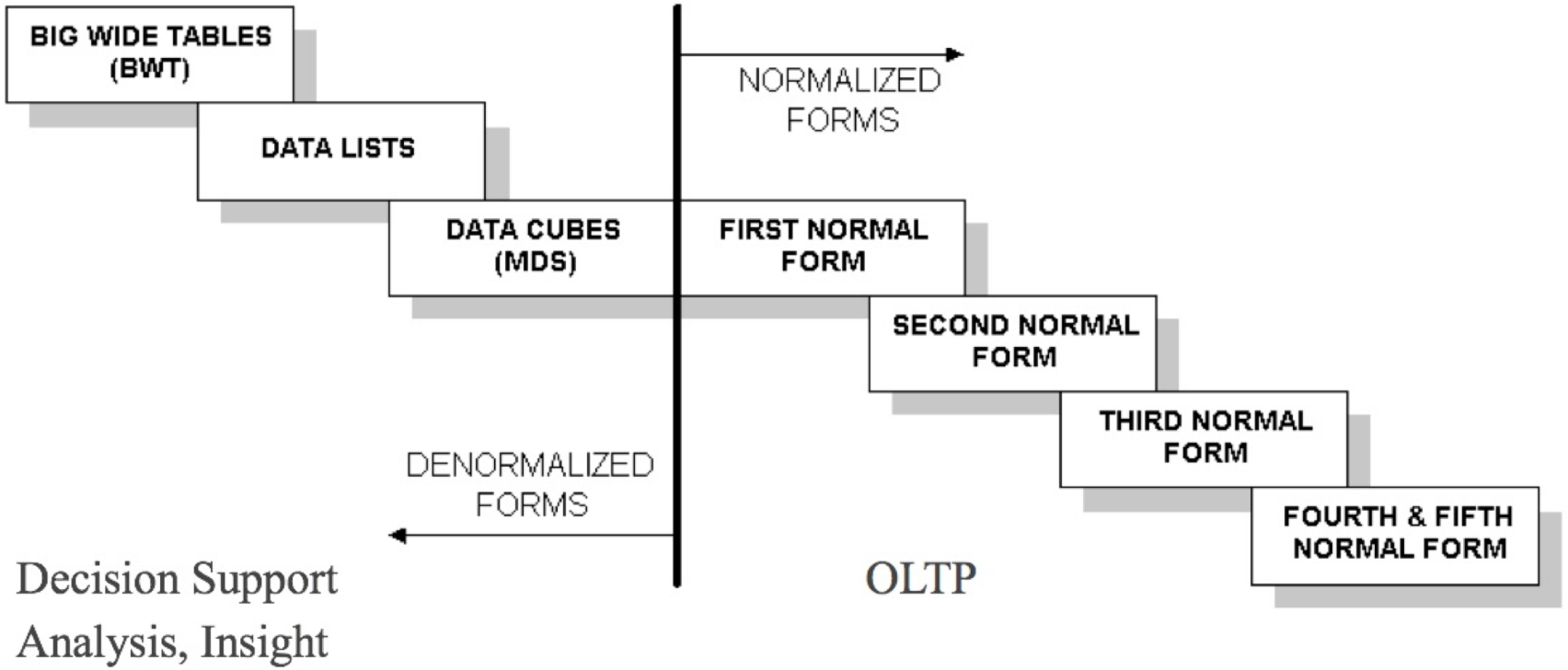
Execution Plan

- 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



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;
```

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	

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