More ETL, Data Marts, Metadata preaching, Another Elaborate HW Assignment

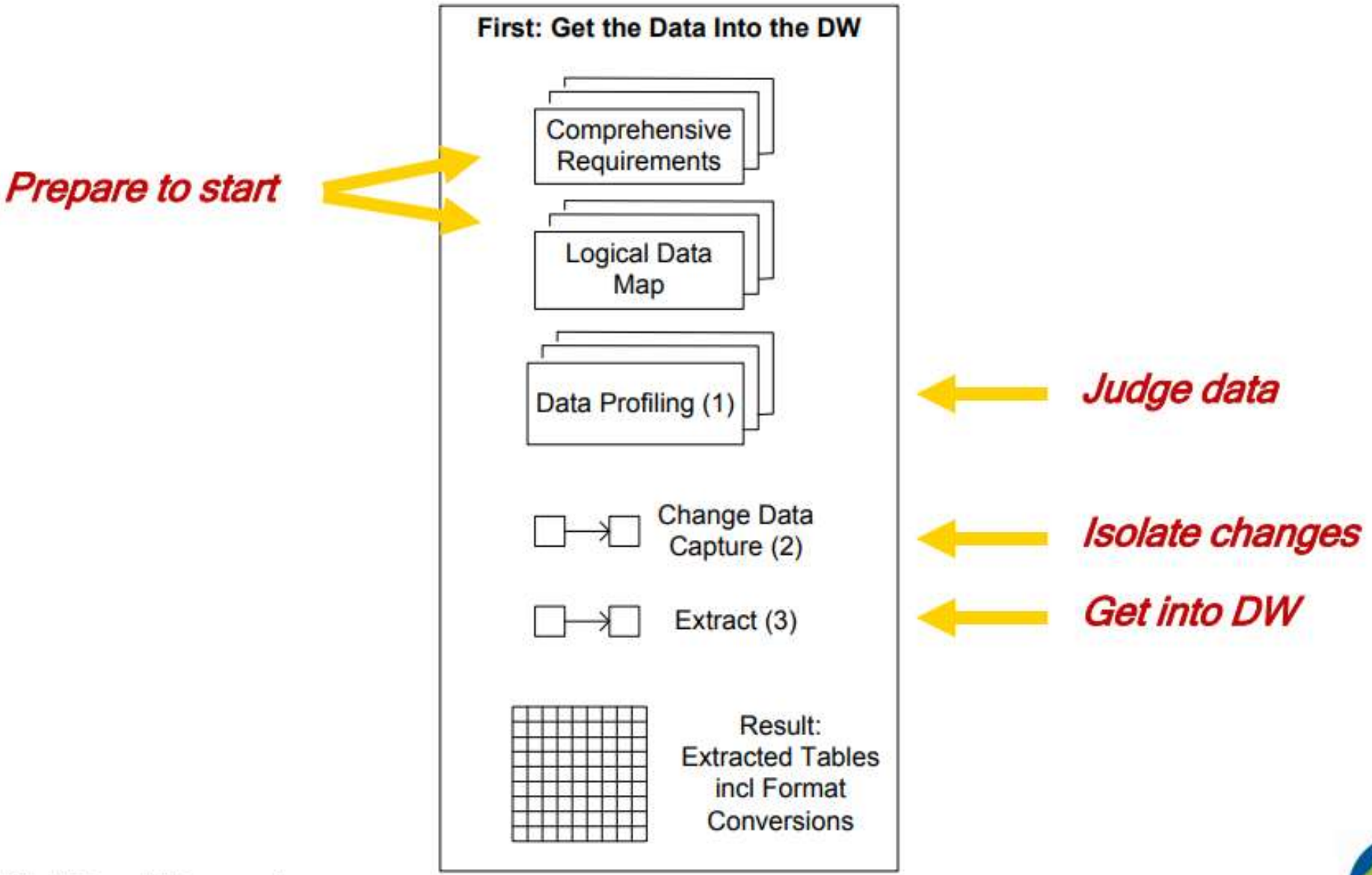
Breitzman 2/26/2025

34 ETL Subsystems

- According to Ralph Kimball, there are 34 subsystems related to the ETL process. (Actually his process is ETLM – Extract, Transform, Load, Manage)
- Three subsystems focus on extracting data from source systems.
- Five subsystems deal with value-added cleaning and conforming, including dimensional structures to monitor quality errors.
- Thirteen subsystems deliver data as dimensional structures to the final BI layer, such as a subsystem to implement slowly changing dimension techniques.
- Thirteen subsystems help manage the production ETL environment.
- Don’t worry, we’re not going to talk about all of them today, but we’ll talk about the ones we need for the next phase of our grocery data warehouse.
- Next 4 slides are ‘borrowed’ from Kimball group website

E: Getting the Data Into the DW

Note: Numbers in the parentheses refer to Kimball's 34 ETL subsystems.



KIMBALL GROUP

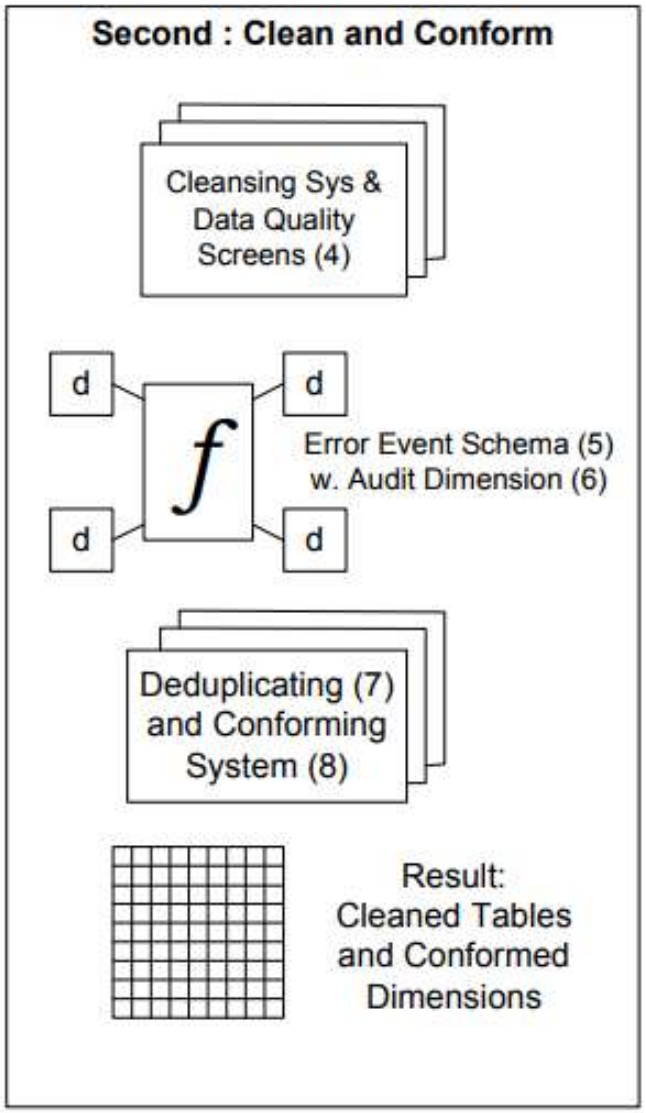
T: Clean and Conform

Note: Numbers in the parentheses refer to Kimball's 34 ETL subsystems.

Cleaning machinery →

Cleaning control →

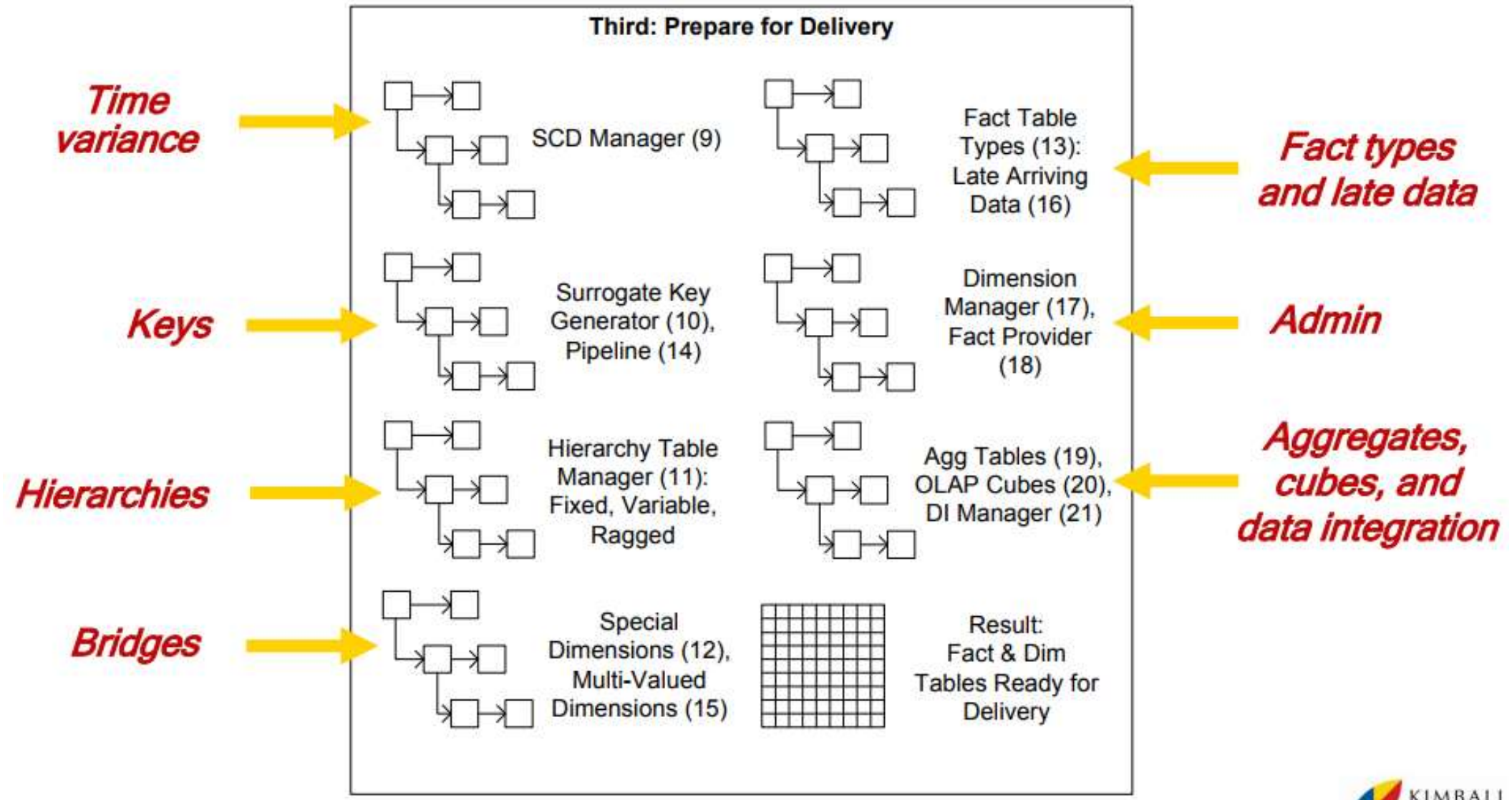
Integration →



KIMBALL GROUP

L: Prepare for Presentation

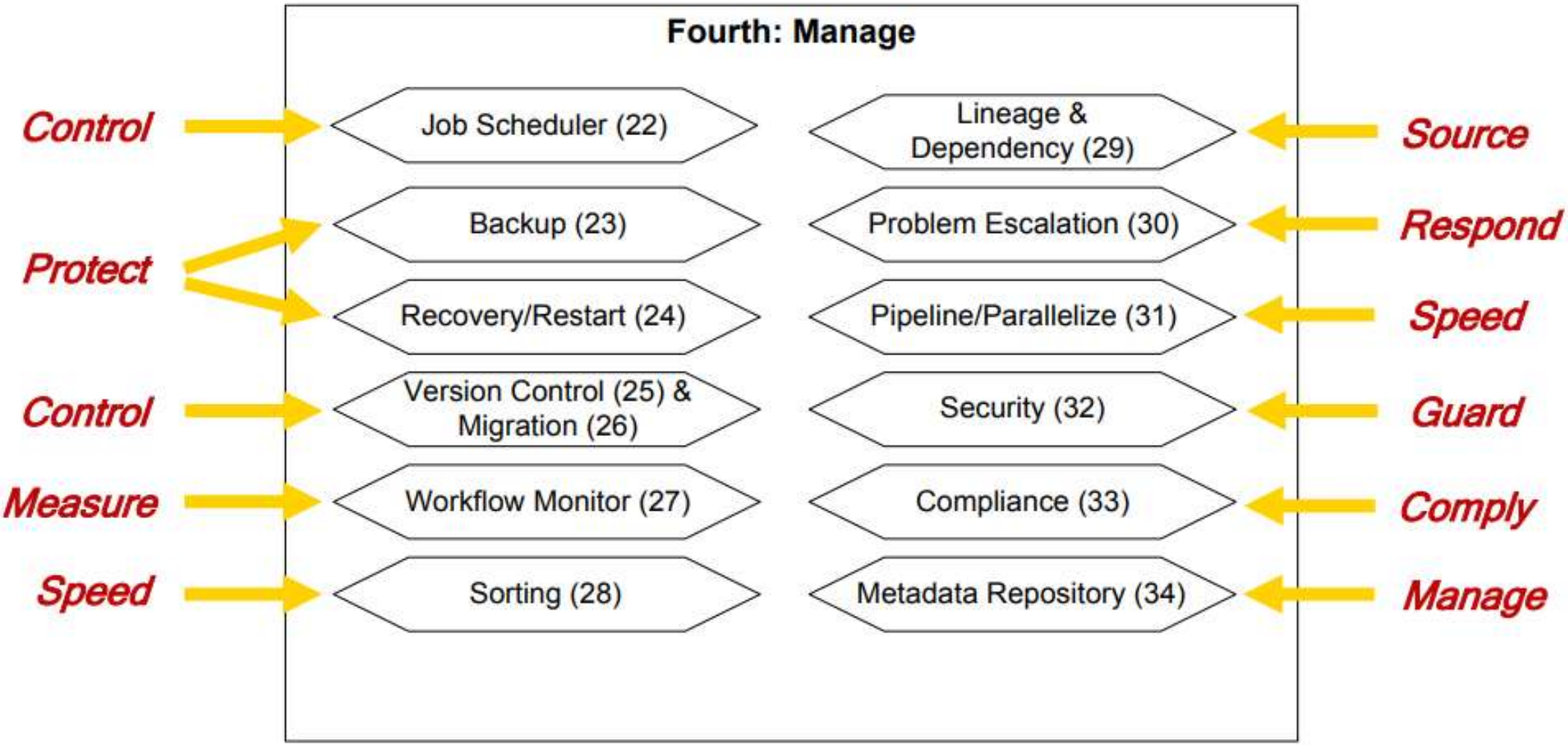
Note: Numbers in the parentheses refer to Kimball's 34 ETL subsystems.



KIMBALL GROUP

M: Manage All the Processes

Note: Numbers in the parentheses refer to Kimball's 34 ETL subsystems.



KIMBALL GROUP

Extract Subsystem

- Data Profiling – talked about that a couple weeks ago. We'll talk about it again today.

- Change Data Capture - This is a fancy way of talking about updating the data warehouse. In some data warehouses there are elaborate methods to see what needs to be updated.
  - For ours, we don't need anything elaborate, because updates will be based on dates. Initial load will contain the whole year data; update will contain data after 12/31/2024
  - Other data warehouses, such as those that allow for returned products, have to go back in time and change things.
  - Ours is simple unless we notice an error in the data warehouse. We will want to fix the error in the source system and flag it somehow so that the next load replaces the old data in the warehouse with the updated data
  - There are also large data warehouses, where not all data can be refreshed at once. A system is needed to make sure the relevant data gets updated and that integrity errors are not introduced by only updating pieces

Extract Subsystem (2)/Transform Subsystem

- Extract – Move data from the source systems to staging database for further processing.

- Cleaning and Transformation – There's a lot to talk about. Here's a sample:
  - De-duping: Vendor lists for multiple stores may have names slightly different. When merging we need to eliminate duplicates
  - Same with customers (e.g. is A. Britzman the same customer as Tony Breitzman?)
  - Conforming data sources (e.g. Prices in Euros for the European operations and Prices in dollars for the US operations. Need to be standardized in one or the other or both)
  - Missing values – need to be filled in with default values, or someone needs to find the right values
  - Wrong values – 4 digit zipcodes need to be fixed, phone numbers in the wrong format etc.

Load and Management Subsystems

- We'll talk about many of these at a future date
- The SCD manager refers to slowly changing dimensions. For example if our Gallons of milk change to 120 ounces, that's a slowly changing dimension and there are strategies to deal with those. (But we don't need to talk about them today)

- Surrogate key generator – we will talk about this. We need to generate keys to map our dimension tables to fact table. They need to be random/meaningless and should be integers. It's the ETL systems job to generate them

- OLAP/Cube Aggregator – that's a subject for another day

- Metadata Repository – that's another subject for another day, but we will have to keep track of where our data comes from, so we will talk about it a little bit

The best way to learn about ETL…

- The best way to learn about ETL is to do ETL, so I have an elaborate team HW assignment to lay out.

- Basically, we will make a data-mart of 3-4 regional grocery stores, which will eventually get rolled up into an Enterprise Data Warehouse

Data Mart Teams

- We already have teams that built stores 1-10
- Now we'll roll those up into regions A=1-4, B=5-7, C=8-10

Key Steps

- Step 1 - Data Profiling: You will want to do a series of profiling steps to make sure your store databases are reasonable
  - For example if Froot Loops are the 10th best selling item in store 1, but they're the 900th ranked item in store 2 then there is something wrong with one of the programs.
  - (By the way, each store should have a name, and the individual team member(s) who generated the data will be listed as the store manager (e.g. the Bryan-Johann Market). You can make up an address and other info as shown in the table several slides ahead.)
  - Also, profile sales, customers per day, total transactions and make sure differences among stores make sense

Key Steps (2)

- Step 2 – Fix individual programs
  - If there are problems with one or more programs, then team members should help other team members fix any bugs
  - Do not just replace all programs with the best working one. We want different programs to have different output formats etc. to mimic how actual chains of stores work. So, fix the existing programs unless they're completely hopeless.
    - The idea here is that chains of stores will have scanners from different manufacturers or different eras and will consequently have different feed formats. ETL teams must deal with data in multiple formats
  - Next re-run profiling exercise.
    - Check for missing values, missing dates, etc.
    - Recheck top selling items as before
    - Deliverable 1 should look something like the next slide

Deliverable One

Combines Sales (December)			Joe's Market		Sue's Market		Jane's Market		Steve's Market	
$ 13,043,675.00			$ 3,069,100.00		$ 3,682,920.00		$ 3,376,010.00		$ 2,915,645.00	
Combined Customer Count (December)			20400		24480		22440		19380	
Best Selling Items Combined		# Sold	# Sold	Rank	# Sold	Rank	# Sold	Rank	# Sold	Rank
SKU 1	Product 1	130050	30600	1	36720	3	33660	2	29070	1
SKU 2	Product 2	127415	29980	2	35976	2	32978	1	28481	3
.										
.										
.										
SKU 25	Product 25	63707.5	14990	25	17988	24	16489	28	14240.5	23

This will be an actual product name like Whole Milk

Note the rankings will differ by store, but not wildly.

A Common ETL Issue

- Missing Data/Conforming Data – The products table has multiple issues
  - Some ItemTypes are missing
  - Some ItemTypes are too broad
  - Some ItemTypes are too fine

- Upper Management has decided that every SKU should be mapped to one of the 114 product subcategories in product_class.txt

- This happens all the time. Companies get reorganized into new business units and the Data Warehouse must get reorganized as well
  - It's a pain, but remember a Data Warehouse is for business users in management. They don't care about inconveniencing the ETL team

Fixing the Products Table

- The Good News: 1319 of the 2076 products have an itemType that maps directly to the subcategory

- The Bad News: 294 items in the product table have null values. These will have to be assigned to one of the new subcategories
  - is there a way to automate this, by string searching or is it a data entry problem? ETL team decides.
  - Are there any products that don't fit into a category? Suggest a new category to management (me) if you find one that doesn't fit.

- More bad news: There are 597 items with item types, but those item types are obsolete (either too broad or too fine).
  - For example there are a bunch of Potato Chips currently called 'Snacks' that want to go to 'Chips' and a bunch of 'Bread' that want to go to 'Sliced Bread'
  - I'll bet there is a way to automate these changes

Deliverable 2a

- Deliverable 2a: A new products table that contains: Manufacturer, Product Name, SKU, Size, Product Class ID, etc as shown
  - This should have no null values (and only Product Class IDs that exist in the Product Class Table or new ones that Management has agreed to add)

Product Dimension	
Product key	int
SKU	ShortText
Product Name	ShortText
Product Class ID	int
Subcategory	ShortText
Category	ShortText
Department	ShortText
Product Family	ShortText
Size	ShortText
#Per Case	int
Brand Name	ShortText
Manufacturer	ShortText
Supplier	ShortText

- Note supplier is always Rowan Warehouse for everything but milk. Milk comes from Rowan Dairy
- Note the product key is a random integer for each SKU. (You might want to ask other teams how they are generating theirs, otherwise when we roll up all the data-marts into an Enterprise Data Warehouse, we will have an issue)

Deliverable 2b

- Deliverable 2b: We haven't talked about meta-data much, but every record needs a source and a reason.
  - So we'll need another table that contains each SKU and source#
  - Source# can just be 1,2,3,4,5, etc. where 1 means it came from original product table, 2 means it was mapped by hand by Jane Doe, 3 means it was done by a string match like Product Name = 'Frito Lay' implies subcategory 'Chip' etc.
  - You will obviously need a source# table that contains definitions similar to above for each source#

Another Common ETL Problem

- It's likely that if you have 3 stores that you may have more than one date format (e.g. 20240101, 1/1/2024, January 1, 2024
- Make sure as part of your ETL process you conform the dates.
- Don't rewrite the individual programs to use the same format (that's cheating). Instead conform the dates in the staging database that combines the store data
- You might ask the other teams what kind of date format they're using so we don't have to do this again when we roll-up the data-marts into an Enterprise Data Warehouse (Maybe we should just go with 20240101)
- Note this is why Bill Inmon (father of data warehousing) doesn't like building Data Warehouses bottom-up. He believes the Data Warehouse should be built first, and then individual data marts should be subsets (But not everyone has 20 million dollars, so most of the time they are built bottom-up like ours)

Deliverable 3

- This one's an easy one

Store Dimension	
Store key	int
Store Manager	ShortText
StoreStreetAddr	ShortText
StoreTown	ShortText
StoreZipCode	ShortText
StorePhone#	ShortText
StoreState	ShortText

- Store key is on an earlier slide
- Manager name is one of you
- Other fields can be made up

Deliverable 4

- Another easy one

Date Dimension	
DateKey	Int
Date	Date/Time
DayNumberInMonth	int
DayNumberInYear	int
WeekNumberInYea	int
MonthNum	int
MonthTxt	ShortText
Quarter	int
Year	int
Fiscal Year	int
isHoliday	Boolean
isWeekend	Boolean
Season	ShortText

- I would make the datekey 1 to 365. (a 2-byte int should allow for 50 years so that ought to be enough)
- Someone will have to look up the holidays and fill in the appropriate fields
- Base seasons on the Solstices and Equinoxes
- Fiscal Year ends in July so anything before July 31 is 2023 and anything after is 2024 (It's common for companies to have fiscal years independent of calendar year because hardly any companies are founded on January 1)

Deliverable 5

- Not Really a deliverable (don't send it to me)

Sales Fact (Transaction Level)	
DateKey	Int
DailyCust#	Int
ProductKey	Int
StoreKey	Int
QuantitySold	Int
TotalDollarSales	float
TotalCostToStore	float
GrossProfit	float

- Just do it for the month of December

- Total dollar sales is sale price*quantity; Total cost is base price*quantity; profit is Total Dollar Sales – Total Dollar Cost

- DateKey,DailyCust#,ProductKey is your composite key (it's unique unless something went wrong) and each component links back to the corresponding dimension tables with no integrity issues

Deliverable 6

- This one is a daily aggregate of the individual transaction table

Sales Fact (Daily Level)	
DateKey	Int
ProductKey	Int
StoreKey	Int
#SoldToday	int
CostOfItemsSold	float
SalesTotal	float
GrossProfit	float

- Note you can't use your Deliverable 5 for this though because that only has a month's data

- A better approach would be to aggregate from the individual stores and then append them together

Deliverable 7

- Not really a deliverable. I don't want the whole table

InventoryFact (Daily Level)	
DateKey	int
ProductKey	int
StoreKey	int
#Available	int
CostToStore (itemLevel)	float
CostToStore (caseLevel)	float
#CasesPurchasedToDate	int

Deliverables 8-11

- 4 Quarterly inventory snapshots (these we want)

InventoryFact (Quarterly Snapshot)	
ProductKey	int
StoreKey	int
Quarter and Year	ShortText
Quarter	int
Year	int
#CasesPurchasedToDate	int
#CasesPurchasedThisQuarter	int
#CasesOnHand	int
TotalCostToStoreThisQuarter	float
TotalSoldByStoreThisQuarter	float
TotalCostToStoreYTD	float
TotalSoldByStoreYTD	float

Due Dates

- Due Dates:
  - It shouldn’t take terribly long, but there is a problem whenever conforming multiple databases of false starts
  - Plus there are multiple deliverables and a midterm
  - Plus I haven’t looked at your last HW
  - So we’ll call it April 1 which is a ridiculous amount of time, but don’t wait until the last minute

Tips/Tricks (1)

- Document as you go
- We will eventually put documentation into the Metadata repository
- Note we built dimension tables first.
  - This is not an accident.
  - Keys are made up and generated at the time the dimension table is created
  - Fact tables use these keys. Make sure you have referential integrity (that is there is no dimension key in a fact table that is not in the appropriate dimension table and vice-versa)
  - While we're at it, check every field of every table and make sure there are no nulls
- There is no such thing as an ad-hoc query in a data warehouse
  - It may sound ridiculous, but every time you think of building an ad-hoc query don't do it. Every query should be called from a stored procedure or macro or trigger or batch file, or from a glue language like PowerShell or Python or from an integration tool like Pentaho Data Integration or Luigi

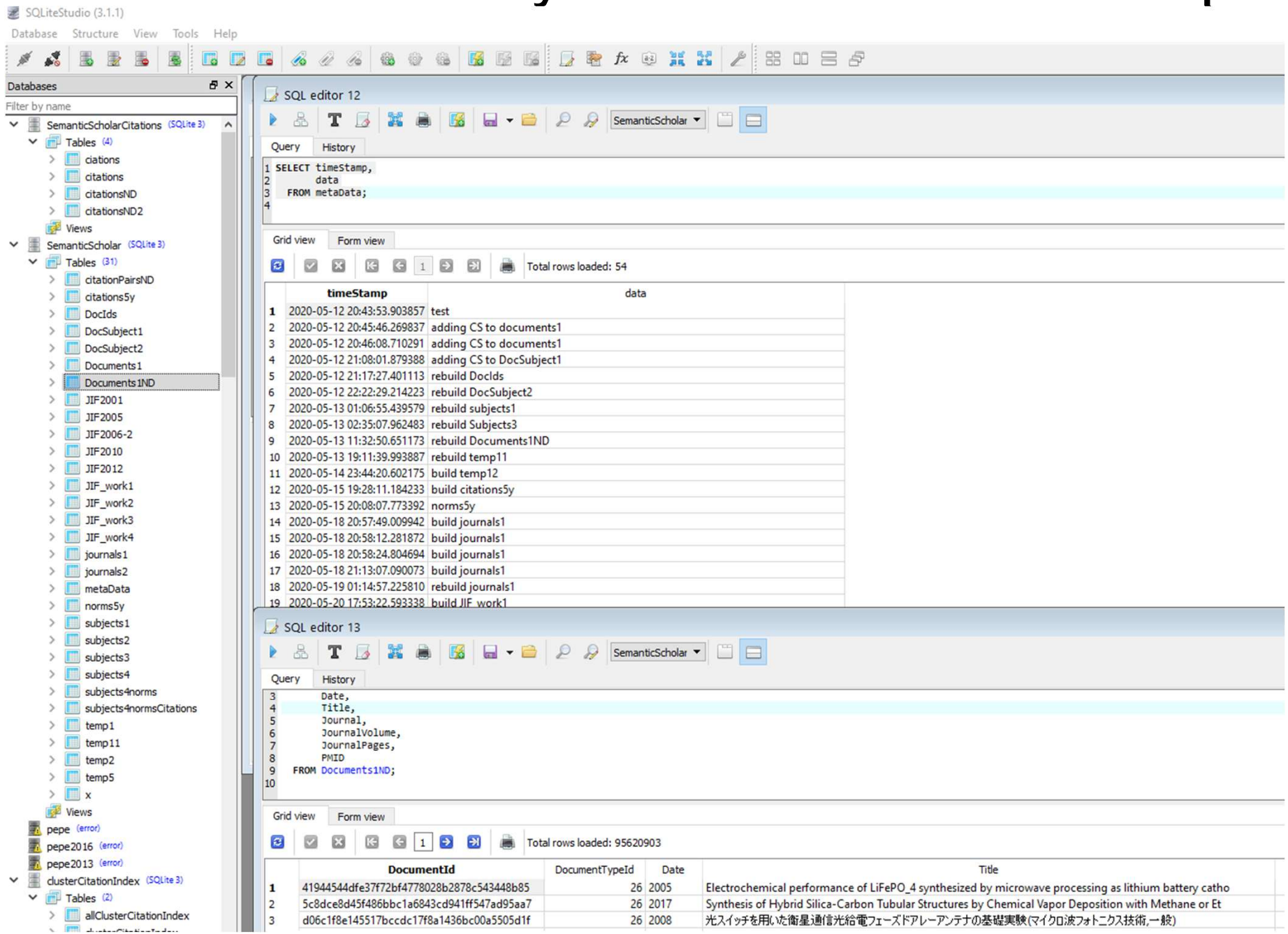
What does this really mean?

- There is no such thing as an ad-hoc query or ad-hoc fix in a Data Warehouse

- **What Is he talking about?**
  - It's tempting when you see a missing value in the products table or a typo to just fix it.
  - The problem with that is a month from now if a fire destroys the database server. We typically download a backup copy, but we may be unaware of any changes made between the last backup and the fire.
  - So even though it's painful it's better to write an update query to fix the typo and at minimum paste it into a Jupyter notebook or batch file or Luigi or Pentaho or Talend process.

- Best practice is not only to fix the typo with an update query but also insert a new record into the metadata table that says on 2/26/2025 we fixed a typo from Lay's Potatoe Chips to Lay's Potato Chips

Non-Grocery Store Real Life Example



This is a semantic scholar database with 95 million papers and 54 lines of metadata describing how it was built

Non-Grocery Store Real Life Example

SQL editor 12

SemanticScholar

Query | History

```
SELECT timeStamp,
       data
  FROM metaData;
```

Grid view | Form view

Total rows loaded: 54

	timeStamp	data
19	2020-05-20 17:53:22.593338	build JIF_work1
20	2020-05-20 18:39:32.580851	build JIF_work2
21	2020-05-20 19:31:53.761702	build JIF_work3
22	2020-05-20 19:32:16.687594	build JIF_work3
23	2020-05-20 19:41:18.167551	build JIF_work4
24	2020-05-20 20:45:15.669731	build JIF_work3
25	2020-05-20 20:54:35.682908	build JIF_work3
26	2020-05-20 21:27:31.062411	build JIF2012
27	2020-06-15 19:12:36.377290	renamed temp12 to citationPairsND
28	2020-06-23 00:35:15.850041	build subjects4 (exactly the same as subjects3 except it has a pa
29	2020-06-23 00:35:35.537682	build subjects4 (exactly the same as subjects3 except it has a pa
30	2020-06-23 02:30:01.878812	build subjects4norms
31	2020-06-23 03:05:40.064533	build subjects4normsCitations
32	2020-06-23 03:09:54.782923	build subjects4normsCitations
33	2020-08-10 18:45:11.091847	build JIF_work1 for 1 year JIF (2001)
34	2020-08-10 18:49:24.750282	build JIF_work2
35	2020-08-10 19:37:15.026281	build JIF_work4
36	2020-08-10 19:37:16.430265	build JIF_work3
37	2020-08-10 19:37:58.560390	build JIF2001
38	2020-08-17 17:21:26.750404	build JIF_work1 for 5 year JIF (2005)
39	2020-08-17 17:51:45.811692	build JIF_work2
40	2020-08-17 18:31:56.504771	build JIF_work4
41	2020-08-17 18:32:02.677467	build JIF_work3
42	2020-08-17 18:33:06.792118	build 5-year JIF2005
43	2020-08-17 18:49:28.479532	build 5-year JIF2005
44	2020-08-24 16:22:04.752261	build JIF_work1 for 2 year JIF (2006)
45	2020-08-24 16:34:54.951252	build JIF_work2
46	2020-08-24 17:18:42.213884	build JIF_work4
47	2020-08-24 17:18:47.114274	build JIF_work3
48	2020-08-24 17:23:57.484154	build 2-year JIF2006-2
49	2020-08-24 17:47:18.990516	build 2-year JIF2006-2
50	2020-08-24 20:29:14.431695	build JIF_work1 for 5 year JIF (2010)
51	2020-08-24 20:39:31.017629	build JIF_work2
52	2020-08-24 21:29:57.586105	build JIF_work4
53	2020-08-24 21:30:13.537669	build JIF_work3
54	2020-08-24 21:37:33.715273	build 2-year JIF2010

- You might say that the comments in the meta table are so cryptic that they aren't useful.
- The key is they exist and they are date stamped.
- I can search for any of these items in my Python script to see exactly what was done

```
('7337f695dda255baafd36c1b995f6e469a56a698', '2016', 'xxxMedicine')
('9d278aec52ec7199b6782b960b66e3e9f2e895d4', '2011', 'xxxMaterialScience')
('949cd269e9f3ebad47e97cddbd843b8c21c43e9b', '2003', 'xxxMaterialScience')
```

```
In [37]: total_rows(cur,'journals1',False)
Out[37]: 95620903

In [23]: insertMeta(con,cur,"build JIF_work1 for 5 year JIF (2010)")
         cur.execute('drop table if exists JIF_work1')
         s = 'create table JIF_work1 as SELECT distinct cited, citedDate, citing, citingDate '
         s = s + 'FROM citationPairsND where
         ((citedDate > 2004 and citedDate < 2010) and citingDate = 2010);'
         cur.execute(s)
Out[23]: <sqlite3.Cursor at 0x20af62082d0>

In [24]: total_rows(cur,'JIF_work1',True)
         selectNrows(cur,'JIF_work1',25)

         Total rows: 7978002
         ('d2f4ec6272ff209ff8c94812480decce262d9214', '2007', '671d486c59214d5cbfed3593890a98acb
         ('72bf5c28fb4ca18e5ee6409811bbfb890eb37d8d', '2009', '336dbce77e35fcf8322319fdb91b44cc9
         ('5e3ab31a7a10cc4d78faa7ffead5e11de664e241', '2007', 'c62cde4f56691091f3ef127b7710e139e
         ('6e8bfc7eba3bef85031819018feabb86a32a5117', '2008', '0057e9645a79bf2aa38b4057d3d2c2564
```

Non-Grocery Store Real Life Example

- Here's the insertMeta code. It looks to see if a meta-data table exists and if it doesn't it creates it. It then inserts a date stamped message

```
#insert data into metadata table
def insertMeta(con,cur,message):
  cur.execute("SELECT count(*) FROM sqlite_master WHERE type='table' AND name='metaData';")
  info = cur.fetchall()
  if (info[0][0]==0):
    cur.execute('CREATE TABLE metaData(timeStamp Varhchar(100), data TEXT)')
  try:
    cur.execute("INSERT INTO metaData VALUES (?,?)",(str(datetime.datetime.now()),message))
  except lite.OperationalError as err:
    print("insert error: %s", err)
  con.commit()
```

Why you need a metadata table in your life

- It's pretty obvious if you are building a 40 million dollar data warehouse that you should have a metadata table, but I keep them for all my projects.

- I don't have a $40 million dollar data warehouse, but I do have a data warehouse of patents granted in 1969 through last Tuesday.

- Every Tuesday new patents are granted and the Data Warehouse is updated.
- Every month citation tables and other metrics from the patent system are updated.
- I have several clients that I generate automatic reports for every month.

- If I get hit by a truck my co-workers need to know how all of this stuff works

- How about for short term projects like the Semantic-Scholar database?
- That one was related to matching patent references to underlying science.
- That one lasted 4-6 months but it's still useful to treat it like a larger enterprise datawarehouse

Why you need a metadata table in your life

- Learn from my youthful indiscretions!

- When I was a young consultant, I used to do projects that included building software or a database, generating a report or dashboard and then moving on to the next project.

- Invariably when clients are paying real money for a project, they have the right to call you 6 months later and ask questions about the underlying data.
- If it's not well-documented you will spend a day trying to figure out your logic from 6 months ago.

- So now I have at minimum a batch-file or jupyter notebook for every table and query I build and a date stamped record of it in a metadata table.
  - Even if the metadata is a cryptic message, it is linked to a jupyter notebook that exists for that project and can be searched so that we can see exactly what query is associated with the metadata record.
  - It's essentially self-documenting. We don't spend a lot of time on it because it is unlikely to be needed in a short-term project but if it is needed it will save hours.
  - On a $40 million data warehouse we might have a more exotic metadata repository.

Why you need a metadata table in your life

- One last note and then I'm done evangelizing on metadata
- You're building a query that depends on table1 and table2. You know that you recently updated table0 which will affect table 2.
- You updated table0 before going on vacation and can't remember if you reran the query to update table2
- Just to be safe do you update table2 even though it takes an hour?
- No, because you can see in your metadata table that it was rebuilt right after table0 was updated.
- Better yet create a trigger that automatically updates table2 any time table 0 is updated.
- Look at HTML file of another database that shows metadata table updates

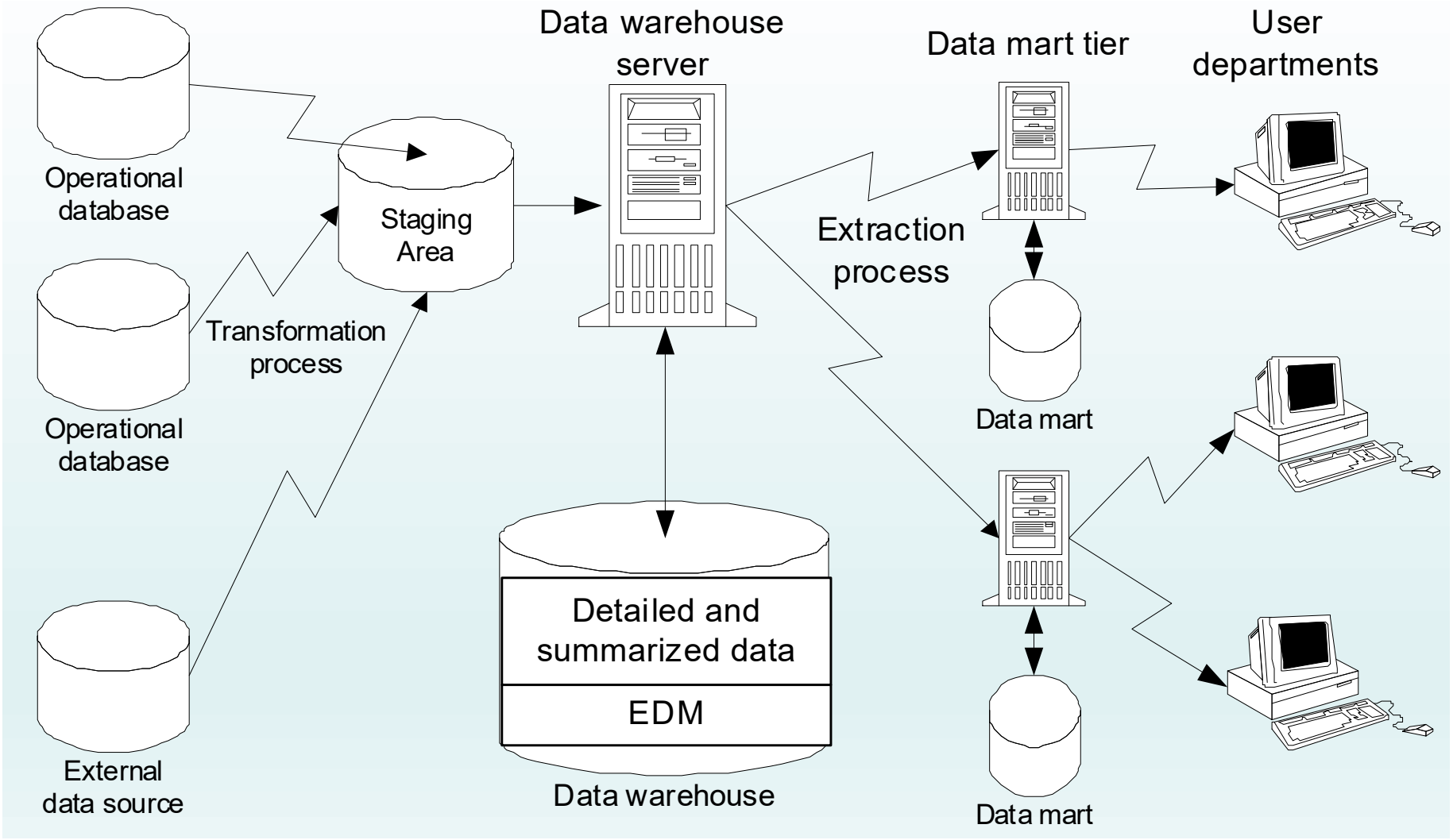
Tips/Tricks (2)

- Keep track of sources of tables
  - You don't have to go to the detail of deliverable 2b (that's an exercise to show how the world works)
  - But we should know the source of the transactions (e.g. JoeGroceryV3-6.py) Because although it's fresh in your mind now, which version of the program is the latest and greatest, that might not be the case a month from now.

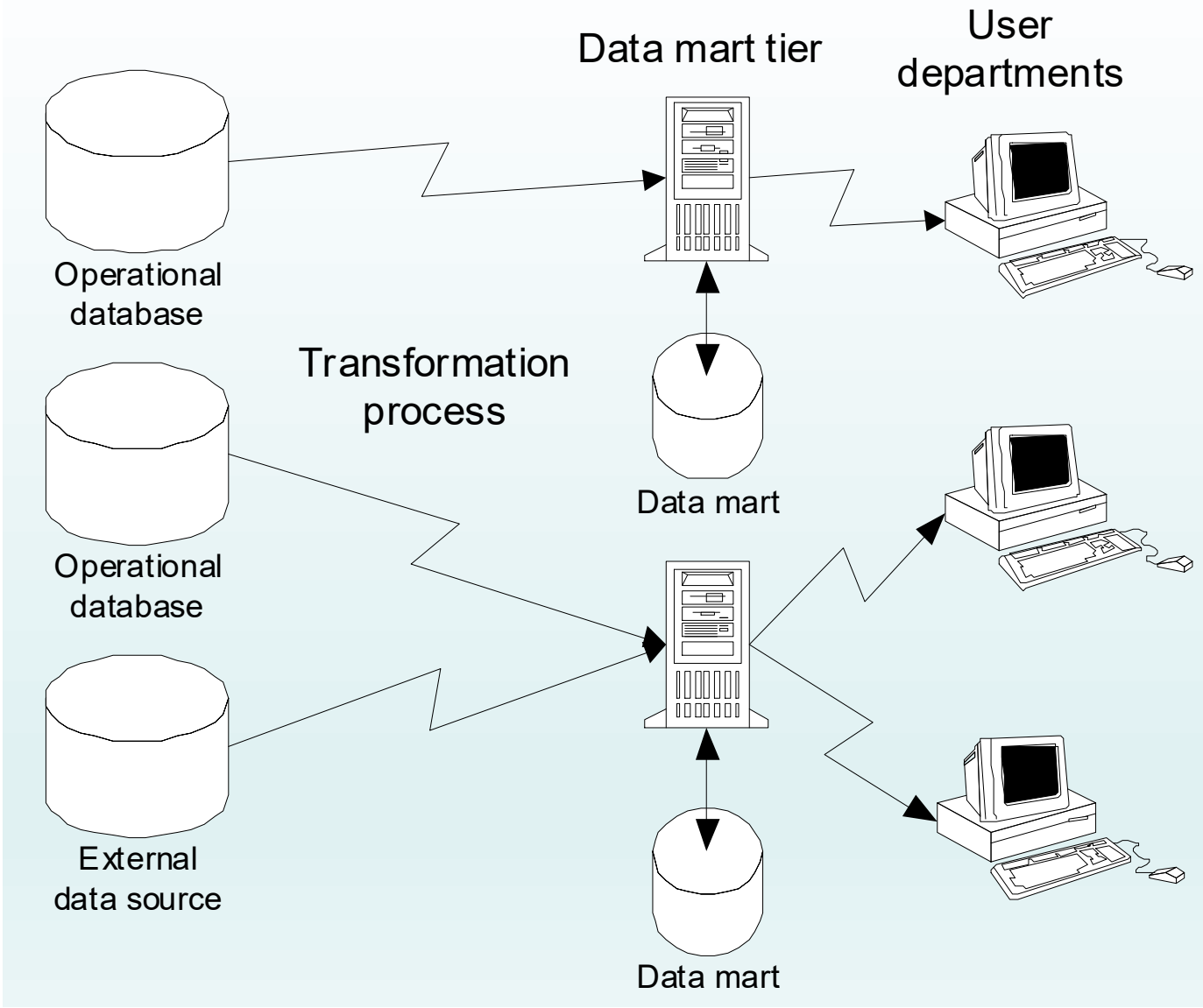
- Each team is responsible for its regional datamart of 3 to 4 stores, but it might be worth talking to the other teams about their approach as you progress. Remember we are eventually rolling these things up to a whole grocery chain, so the better they are integrated now, the easier it will be to build the final warehouse

- This was kind of a disaster the first time, but it went pretty well last time and I would expect even better this time.

This is Not what we're doing
Top Down Architecture (U. Colorado Bus. School)



This is what we're doing
Bottom Up Architecture (U. Colorado Bus. School)



- No centralized warehouse. Inmon argues this is a bad idea. Kimball argues this is a good intermediate step to building a warehouse

One More Thing… (maybe more than one)

- I know that many of you hate group work...
- Too bad!

- There is no such thing as an ETL Lone Wolf. It's called an ETL team. You may find yourself on one some day.

- If you are feeling overwhelmed at this point, relax. It's a lot of steps, but I deliberately put it into bite size pieces. None of the steps are particularly difficult (tedious perhaps but not difficult).

- Put a detail-oriented person on the product mapping. This is always an interesting exercise and I always find some goofy things that I get to mock you about in a couple of weeks.

- Have a little bit of fun with this part. ETL doesn't have to be the nightmare that it seems…