# Part 1 (Theoretical questions)

## Question 1

Dimensional design is process, after which *dimension* and *fact* entities can be formed. The purpose is to optimize data warehouse for fast retrieval of the data. Model created after design should describe Why, When/Where/Who, How Much and What business process consist of. According to Kimball, there are four steps in dimensional modeling:

1. *Business Process*

It should answer the business question from requirements.

1. *Grain*

Choose atomic/most detailed information collected. This information cannot be more divided.

1. *Dimensions*

Based on grains, dimensions can be formed. If dimension violates the grain by forcing additional fact row to be generated, then the previous step should be re-done.

1. *Facts*

The last step is determining which fact should be used in Fact Table. Grain can be a big help in this part of process.

## Question 2

There are multiple ways how to achieve security of Data Warehouse. We can consider *Internal* and *External* threats. To make Data Warehouse more secure, Confidentiality, Integrity and Availability should be applied.

*Confidentiality* prevents unauthorized user from manipulation or seeing data. It protects personal privacy and sensitive information.

*Integrity* preserves the authenticity of data, makes sure data is not modified or deleted.

*Availability* ensures fast and reliable access and use of information.

Another rules Kimball says that there should be only one, single database with personal information, not scattered around whole organization. Security roles should be enforced. Server containing personal information should be physically isolated on its own segment and LAN. A database which tracks all the changes and requests on a database with sensitive information should be implemented, this database must notify every individual user which application wants to access its personal data and what is the application. There are many more rules that can be applied on a database or data warehouse.

## Question 3

Factless fact table is a Fact Table with no facts in it. It connects dimensions by holding their primary keys and foreign keys. It does not contain any calculation, just information about the event which happened over the time. For example, if we have Fact connecting an Employee, Date, and Absence dimensions, Fact table can tell us how many times specific employee has taken an absence just by counting entries in the fact table.

## Question 4

Degenerate dimension is a dimension attribute in fact table that does not have corresponding dimension table. That means degenerate dimension uses its field as a primary key, name, and attribute. Example would be extracting *line\_total* from Assignment 2 and 3 to separate Dimension. Then, we could see that this Line Total Dimension would be growing proportionally with Fact Sales entity. This is an indicator, that Degenerate Dimension should be used. Thus, it is better approach to include this attribute as a field in Fact Table.

## Question 5

Bus architecture says that each dimension table is designed and maintained in only one location. Process like load and update the data is executed only on one specific table. This way fact tables can share dimensions and there is no issue with data integrity. Facts use the same data, no matter from where this data come from. Kimball advices to use this type of architecture as it enforces creating dimension prior to facts.

## Question 6

Principles of handling changes in data warehouse context are known under name Slowly Changing Dimensions Techniques (SCD). There are 8 types or/and 8 ways how to handle changed data in data warehouse. Starting with Type 0, ending with Type 7.

Within curriculum of this subject only three of them were discussed which are:

* *Type 1*: Overwrite the value

Old value is overwritten with new value; thus, attributes always reflect the newest value. This technique destroys history. It is easy to implement and does not create additional dimension rows.

* *Type 2*: Add New Row

Every dimension with type 2 changes should contain minimum *valid from* stamp, *valid to* stamp and current row indicator. Whenever the dimension is changed or deleted, *valid to* is set to past date and a new row with new surrogate key and updated value is created.

* *Type 3*: Add New Attribute

To follow this type of changes, a new attribute must be created. This attribute is often related to as an alternate reality. When the change is detected, old value is placed to alternate reality attribute and new value overrides previous one (as in Type 1 changes).

## Question 7

ETL (Extract, Transform and Load) process for initial load of Data Warehouse consists of multiple parts:

1. *Insert data into the Stage Dimensions entities*

Select data relevant data from Relational Database and insert it into Stage Dimensions

1. *Correct NULL values*

Lookup NULL values in Dimensions and replace them with “N/A” or relevant message

1. *Insert corrected data into the Data Warehouse Dimension entities*

When data is inserted in Data Warehouse Dimensions, surrogate keys are autogenerated

1. *Insert data into the Stage Fact entity*

Select relevant data from Relational Database and insert it into Stage Fact entity.

1. *Lookup surrogate keys in Data Warehouse Fact entity based on the business keys*

Use Data Warehouse to make lookup because it already contains autogenerated surrogate keys.

1. *Insert values from Stage Fact to Data Warehouse Fact entity*

Lastly, insert all data to Data Warehouse

A detailed Activity Diagram can be found in *Appendix D – Activity Diagram Initial Load.*

# Part 2 (Give - examples questions)

## Question 1

Data redundancy means that some data is unnecessarily reused or copied throughout the data set. In our case, redundancy occurs while copying data to Data Warehouse Sale Fact Table. To populate Fact Table, a join to multiple tables in relational database must be made to acquire needed data. In other words, we are de-normalizing our database to have data at one place. That is what is needed for data warehouse. Example of such a join can be seen in *Appendix A – Incremental Load Source Code* in Initial Load of Stage Fact Table.

## Question 2

Grain is the smallest, the atomic value of our business requirement. It can not be more divided. This grain is telling is what the data warehouse is going to collect, what data. For example, in out Hand Ins some of atomic values were *quantity* and *total* *price*. This was given to us from customer. *“Senior management would like to be able to track sales by customer, and product, with the goal of establishing which products are the top sellers, which customers are the top buyers, and sales per month for top 3 selling products”.* This was our grain. When the grain is acquired, Dimensions are formed from them, therefore if they are wrongly analyzed, it will be reflected on Dimensions and grains would have to be re-done. If we should design a table with non-atomic data, it would have to be part of grain (business requirement). For example, it could be, *“Which customer made the most orders”*. It that case, Dimension Customer would need only customer Id, to distinguish different customers from each other, but no other atomic information (grain) is needed. In that case, table would have no atomic value.

## Question 3

There are 8 different designs/architectures for incremental load. During our studies we have deeply focused only on Type 2. These types are considered Slowly Changing Dimensions Techniques (SCD). For example, Type 1 just replaces previous value and no history is kept. Apart from this, Type 3 uses so called alternate reality, which is an attribute in the table, which holds previous value. Whenever there is an update, old value is placed to alternate reality column and new value replaces old one.

For our assignments we have used Type 2 architecture – we created attribute for *valid\_from* and *valid\_to* data, and initialized it based on situation. In Initial Load *valid\_from* was looked up in relational database and *valid\_to* was set to *‘9999-12-31’*. While checking for Incremental Load, there were 3 different cases:

* *New value added:*

It is found by comparing two sets of data. These values are the ones which are in relational database but are not in data warehouse. New row is created for every new value in dimensional table.

* *Value deleted:*

It is found by comparing two sets of data. Business keys from relational database are subtracted from all business keys in data warehouse. If data warehouse still contains keys, *valid\_to* attribute on these rows is set to previous day, thus marking these items no longer valid. We NEVER delete though!

* *Value changed:*

It is found by comparing two sets – Relational Database and Data Warehouse. Firstly, we find all new rows. Secondly, we find all changed rows and subtract the new ones. After these operations we will have a set of only changed items which are saved in temporary table. Subsequently we lookup all changed *IDs* in data warehouse and set *valid\_to* attribute to yesterday’s date. Lastly, we update *valid\_from* to todays date and *valid\_to* to end of the world date and insert data into data warehouse.

Decision to follow Type 2 SCD architecture was made based on business requirements and on the fact that this type was the on tough in class.

Methods of Change Data Capture are:

1. *Date Modified* – This CDC assumes that every table contains an attribute with date modified value. When Incremental Load, only rows with modified date higher then last update are considered. This method fails when it comes to detecting deleted rows, unless business logic is implemented which preserves deleted rows in relational database and sets flags instead.
2. *Difference* – This method compares current states of Data Warehouse and RDB. Read more above in Type 2 architecture, which is this method following.
3. *Triggers* – Whenever there is a change of data, trigger is executed, and inserts changed tuple to shadow table. This method might slow down transaction by up to 100%.
4. *Log-Based* – All transactions are saved in transaction log, so they can be later recovered in case of database crash. This CDC uses transactional database to read the changed from the log.

## Question 4

Querying dimensional model/data warehouse is done with external tools like Microsoft PowerBI or it can be done by forming a SQL query. This query is done by joining needed dimensions with fact table. Such an example can be found in Question 3 and example Power BI report can be found in *Appendix C – Power BI Road Bottle Cage Buyers*. Power BI can be used to analyze two types of data. One type is to pull data from database and process it locally, but there are some limitations. The other type is direct query, which queries database directly and changes are seen on graphs few seconds after database updates.

## Question 5

There are three different types of fact tables – Transaction, Periodic Snapshot and Accumulating fact table.

*Transaction Fact Tables* – these tables hold a foreign key to multiple dimensions, and optionally to time stamp of entry. Rows exist only if new sales are created.

*Periodic Snapshot Fact Tables* – in this case the grain is period, not individual transactions. The fact tables are usually dense, because entry is made even if no activity took a place and values are replaced with NULLs or zeroes.

*Accumulating Snapshot Fact Tables* – entry in fact table is created initially with null values and as workflow continues, tuple is re-visited and updated as needed. There is a mandatory date foreign key in this table an optionally contain degenerate dimensions.

Example can be seen *Appendix A – Incremental Load Source Code* in Fact Sales entity. This entity contains foreign keys to other dimensions and some degenerate dimensions. Thus, this table belongs to *Transaction Fact Tables* group.

There are three types of Facts

*Additive* – these facts can be summed up for all dimensions in the fact table

*Semi-Additive* – these facts can be summed up for some of the dimensions in the fact table

*Non-Additive* – They can not be summed for any dimension on the fact table

## Question 6

ETL Process for initial load is done in multiple phases. Activity Diagram can be seen in *Appendix D – Activity Diagram Initial Load*.

* Firstly, two scripts are executed. The first one selects StartDate as minimum date and EndDate as maximum date from relational database. These values form an interval which shows the first and last order. Subsequently, script populates *D\_Date* dimension with relevant data. Secondly, script for creating Price Range is executed. This code creates entries for price ranges, every 200 until it reaches maximum price, for example: 0 – 199.99, 200 – 399,99 … and so on.
* Secondly Customer Stage Dimension is populated from relational database (RDB) by joining Customer and Person table in RDB. After data is inserted, NULL values are set to “N/A”, *valid\_from* is updated to current date and *valid\_to* to highest possible date. The same is done with Product Stage Dimension. This corrected data is inserted into the star schema dimensions.
* Thirdly RDB is queried by joining *SalesOrderHeader*, *SalesOrderDetail*, *Customer* and *Product* tables. *CustomerId*, *ProductId*, *OrderDate*, *OrderQty* and *LineTotal* attributes are selected and inserted into the Sale Stage Fact entity.
* Subsequently code for surrogate keys lookup is executed. This part of Initial Load looks up surrogate key of Product (*dimension\_product\_id*) in Data Warehouse *D\_Product* table based on product business key and assign value to *Stage\_F\_Sales* entity. This principle is repeated for every dimension.
* Lastly, all surrogate keys, total price and quantity from staging table are inserted into the actual *F\_Sales* table in data warehouse. Business keys are now existing only in Dimensions, not the Fact.

# Part 3 (DIY – exercise)

## Question 1

If Bob wants to see updates three times per day in data warehouse, Job Agent must be used. Details of design can be seen in Figure 1. At the bottom of the figure data warehouse – Product Dimension is updated. Implementation of this incremental load responsible for handling these changes can be found in *Appendix A – Incremental Load Source Code*.

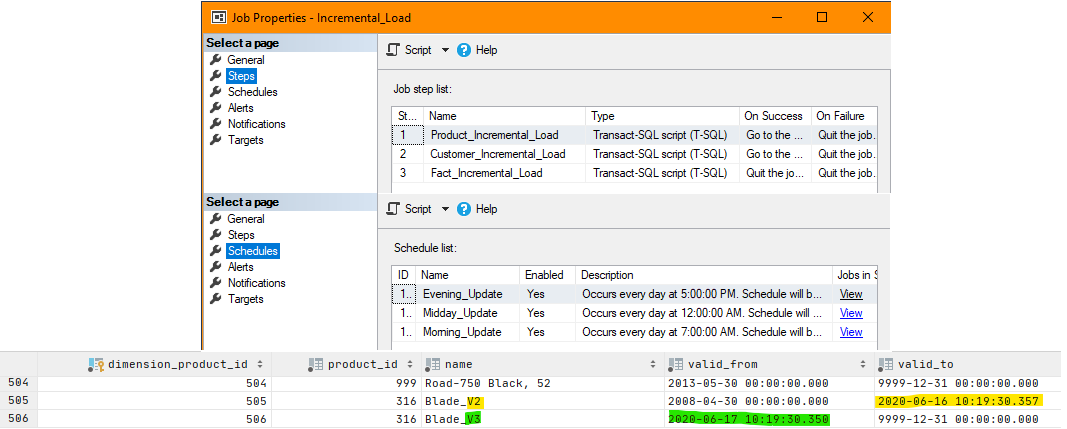


Figure 1 Job agent and result

## Question 2

To see a demand based on small changes in a price, a bridge table containing price range should be implemented. Reason is to avoid overloading data warehouse with big amount of data. Product Dimension will hold reference to Price Range bridge entity (Figure 2), and whenever the new price is out of this range, Dimension is updated accordingly (*valid\_to* set to yesterday’s date, new entry created with new reference). There is a possibility of creating new dimension, connection to current fact table. I did not choose this option because the purpose of Fact Sale entity is to give the user an overview about the whole sale, not a single product.

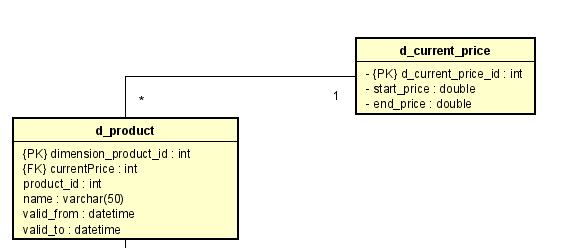


Figure 2 Bridge table Current Price

The table will be pre-populated with data, containing various ranges between minimal and maximum price in relational database. Later, relevant tuple will be looked up based on *currentPrice* and foreign key assigned. This price range could be easily changed. If the changes in price will not be that big, it can still belong to the same price range. Therefore, Price Range table can be easily modified for smaller ranges by changing *@Increment* value to smaller number. In the future, when the product price is changed, it might no longer fit to previous price range, thus data warehouse will be updated.

CREATE TABLE StagingDatabase.staging.d\_current\_price  
(  
 d\_current\_price\_id INT IDENTITY,  
 start\_price decimal(10,2),  
 end\_price decimal(10,2),  
 PRIMARY KEY (d\_current\_price\_id)  
);  
DECLARE @StartPrice decimal(10,2) = 0.00;  
DECLARE @EndPrice decimal(10,2) = 199.99;  
DECLARE @Increment decimal(10,2) = 200.00;  
  
DECLARE @MaxPrice decimal(10,2) = (SELECT *MAX*(ListPrice)  
 FROM AdventureWorks2017.Production.Product);  
  
WHILE @EndPrice <= @MaxPrice  
 BEGIN  
 INSERT INTO StagingDatabase.staging.d\_current\_price (start\_price, end\_price) VALUES (  
 @StartPrice,  
 @EndPrice  
 );  
 SET @StartPrice = @StartPrice + @Increment;  
 SET @EndPrice = @EndPrice + @Increment;  
 END

Figure 3 displays Power BI Report for Data Analyst. One can choose the price range in a Slider and a Product he is observing. Graph will display how many items were sold for specified Price Range. Source code for whole data warehouse process can be found in *Appendix B – Product Analysis Source Code*. Notice Difference Calculations Product when handling Updated Products. Multiple sub-selects must be made to compare price range.

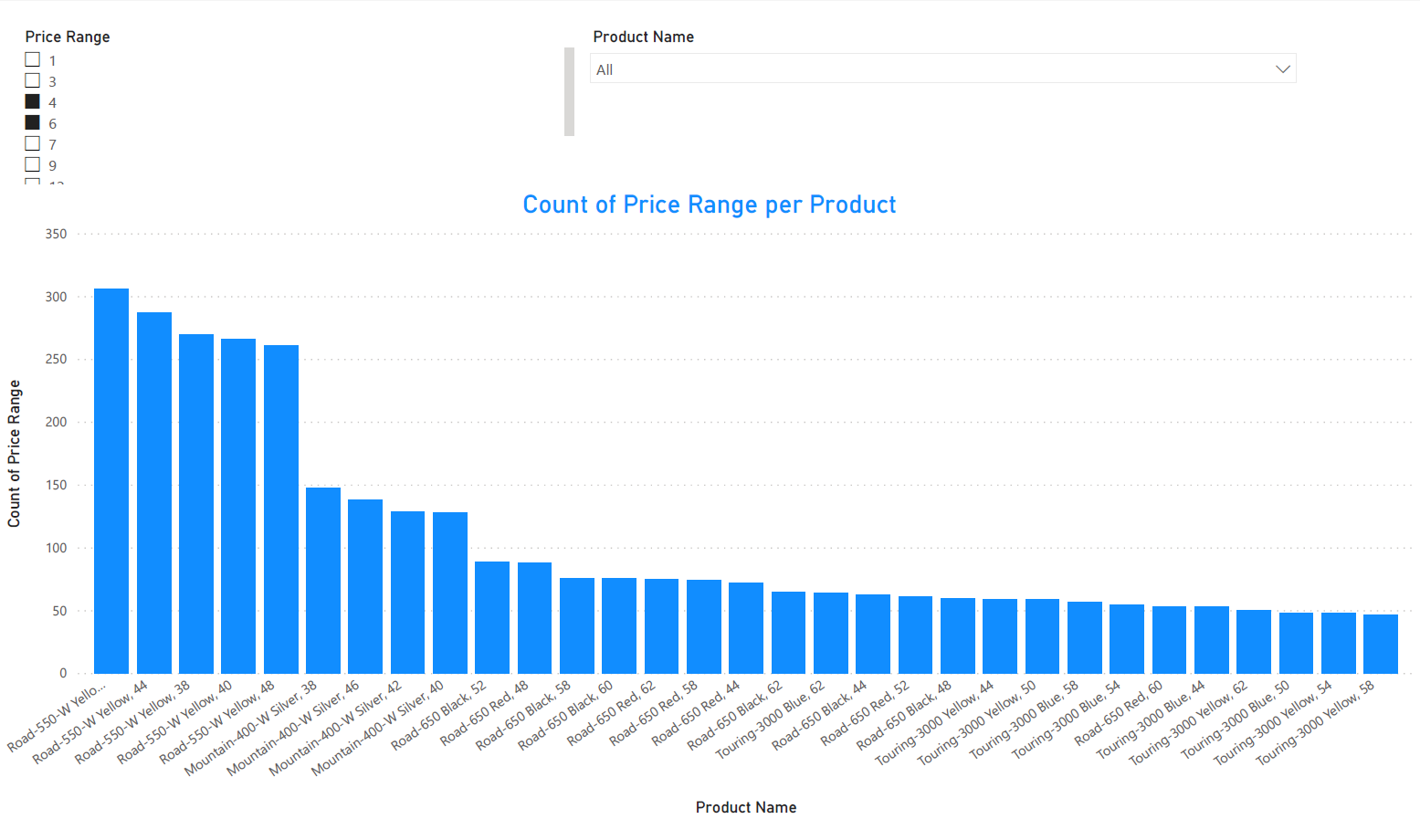


Figure 3 Power BI Report

## Question 3

To display desired information, SQL Query or Power BI can be used. Example of Power BI can be found in *Appendix C – Power BI Road Bottle Cage Buyers.* To use SQL, multiple joins must be made. Firstly, Product Dimension is joined to Sale Fact Entity, then Date Dimension and lastly with Customer Dimension. From this joint table only tuples with associated year 2014 and product name ‘Road Bottle Cage’ are chosen. Data warehouse code can be found in *Appendix B – Product Analysis Source Code.*

SELECT title, first\_name, last\_name  
FROM AdventureWorks\_DW.star\_schema.f\_sales  
JOIN AdventureWorks\_DW.star\_schema.d\_product ON f\_sales.product\_id = d\_product.dimension\_product\_id  
JOIN AdventureWorks\_DW.star\_schema.d\_date ON f\_sales.date\_id = d\_date.dimension\_date\_id  
JOIN AdventureWorks\_DW.star\_schema.d\_customer ON f\_sales.customer\_id = d\_customer.dimension\_customer\_id  
WHERE d\_date.date BETWEEN '2014-01-01' AND '2014-12-31'  
AND d\_product.name = 'Road Bottle Cage'

# Appendices

#### Appendix A – Incremental Load Source Code

#### Appendix B – Product Analysis Source Code

#### Appendix C – Power BI Road Bottle Cage Buyers

#### Appendix D – Activity Diagram Initial Load