1.2 Dimensional Modeling (Kimball)

1.2 – Q1) Design a Star Schema for Sales Analytics

Two word logic: Question first

Utility analogy: Like arranging gauges around a dashboard so every dial answers a different “why” or “where.”

Core Idea

A star schema organizes data into a central fact table (measures) surrounded by dimensions (context), enabling fast, intuitive analytics for revenue, customers, products, and channels.

Architecture (Two Word Logic per Step)

Step	Component	Two-Words Logic	Description
1	BI Requirements	Question mining	Identify key questions: revenue, margin, customer behavior.
2	Grain Definition	Event clarity	Define atomic grain: one row per order line.
3	Fact Table	Metric center	Create fact_sales_line with measures and foreign keys.
4	Dimension Design	Context framing	Build dimensions: date, customer, product, channel, and region.
5	Surrogate Keys	Stable identity	Use surrogate keys for slowly changing attributes.
6	SCD Strategy	History handling	Choose SCD Type 1/2 for dimensions.
7	Aggregations	Query speed	Pre-aggregate for common rollups.
8	Naming Standards	Cognitive ease	Use business friendly names.
9	Documentation	Analyst	Publish schema diagrams and

		enablement	**metric definitions.**

High Level Flow (Analogy)

Questions gathered → Grain chosen → Fact table built → Dimensions added → Keys assigned → History rules applied → Aggregations created → Analysts enabled.

Python-Style Pseudocode

```
def build_fact_sales(order_line):
    return {
        "date_key": to_key(order_line["order_date"]),
        "customer_key": lookup_customer(order_line["customer_id"]),
        "product_key": lookup_product(order_line["sku"]),
        "channel_key": lookup_channel(order_line["channel"]),
        "quantity_sold": order_line["qty"],
        "gross_revenue": order_line["qty"] * order_line["price"]
    }
```

Design Principles (Two Word Logic)

- **Grain clarity**
- **Star simplicity**
- **Metric governance**
- **Query performance**
- **Business naming**

Interview Ready Summary

“A star schema starts with defining the grain and business questions. I design a central fact table for sales measures and surround it with conformed dimensions like date, customer, and product. Surrogate keys and SCD strategies preserve history, while clear naming and documentation enable self-service analytics.”

1.2 – Q2. Select an SCD Strategy (Type 1/2/3/6)

Two word logic: History strategy **Utility analogy:** Like choosing how to track changes in a customer's address — overwrite, keep history, or store both old and new.

Core Idea

Slowly Changing Dimensions (SCDs) define how attribute changes (e.g., customer segment, product category) are tracked over time. The strategy depends on business need for history.

Architecture (Two Word Logic per Step)

Step	Component	Two Word Logic	Description
1	Attribute Review	Change analysis	Identify which attributes change and how often.
2	Business Need	History requirement	Determine if history must be preserved.
3	SCD Selection	Strategy choice	Choose Type 1, 2, 3, or 6.
4	Key Assignment	Identity stability	Use surrogate keys for SCD2.
5	Effective Dating	Time bounding	Add start/end dates for SCD2.
6	Version Columns	Change tracking	Add version numbers or flags.
7	ETL Logic	Update rules	Implement insert/update logic.
8	Testing	Behavior validation	Validate history correctness.
9	Documentation	Rule clarity	Document SCD rules for analysts.

High Level Flow (Analogy)

Identify changing fields → Decide if history matters → Pick strategy → Add keys/dates → Implement ETL → Validate → Document.

Python Style Pseudocode

```
def apply_scd2(existing, incoming):
    if existing["attribute"] != incoming["attribute"]:
        expire(existing)
        return create_new_version(incoming)
    return existing
```

Design Principles (Two Word Logic)

- **History preservation**
- **Business alignment**
- **Stable keys**
- **Clear rules**
- **Predictable behavior**

Interview Ready Summary

“I choose SCD strategies based on business need for history. Type 1 overwrites, Type 2 preserves history with new rows, Type 3 stores limited prior values, and Type 6 blends multiple techniques. Surrogate keys, effective dates, and clear ETL logic ensure consistent behavior.”

1.2 – Q3. Define the Grain of a Fact Table

Two word logic: Event clarity **Utility analogy:** Like deciding whether a restaurant log tracks each meal, each dish, or each ingredient.

Core Idea

The grain defines the atomic level of detail in a fact table. All measures and dimensions must align to this grain to avoid ambiguity and double counting.

Architecture (Two Word Logic per Step)

Step	Component	Two Word Logic	Description
1	Business Events	Event inventory	Identify events: order, order line, click, and shipment.
2	Grain Options	Level selection	Compare possible grains.
3	Atomic Choice	Detail commitment	Choose the lowest meaningful level.
4	Measure Alignment	Metric consistency	Ensure measures match the grain.
5	Dimension Fit	Context alignment	Ensure dimensions join cleanly at the grain.
6	ETL Design	Load discipline	Enforce grain during ingestion.
7	Aggregation Rules	Rollup clarity	Define how to aggregate measures.
8	Validation	Double check	Ensure no double counting.
9	Documentation	Grain statement	Publish a clear grain definition.

High Level Flow (Analogy)

List events → Compare levels → Choose atomic grain → Align measures → Align dimensions → Validate → Document.

Python Style Pseudocode

```
def enforce_grain(order_line):
    return {
        "grain": "order_line",
        "order_id": order_line["order_id"],
        "line_number": order_line["line_no"]
    }
```

Design Principles (Two Word Logic)

- **Atomic detail**
- **Consistent measures**
- **Dimension alignment**
- **No ambiguity**
- **Clear documentation**

Interview Ready Summary

“I define the grain before designing any fact table. The grain determines what one row represents—order line, click, shipment, etc. All measures and dimensions must align to this grain to avoid ambiguity and double counting.”

1.2 – Q4. Design Conformed Dimensions across Domains

Two word logic: Shared meaning **Utility analogy:** Like using one universal map so every department navigates the same city the same way.

Core Idea

Conformed dimensions (Customer, Product, Date, Region) are shared across multiple fact tables and domains, ensuring consistent reporting and cross functional analytics.

Architecture (Two Word Logic per Step)

Step	Component	Two Word Logic	Description
1	Domain Inventory	Scope mapping	Identify domains: Sales, Marketing, Finance, Support.
2	Dimension Selection	Shared candidates	Choose dimensions that span domains.
3	Attribute Standard	Field harmonization	Standardize attributes across systems.
4	Key Strategy	Identity consistency	Use surrogate keys for stability.
5	SCD Rules	History alignment	Apply consistent SCD strategies.
6	Hierarchy Design	Rollup consistency	Define hierarchies (e.g., region → country → market).
7	Integration Layer	Cross-domain linkage	Ensure dimensions join cleanly across facts.
8	Testing	Consistency checks	Validate conformance across domains.
9	Documentation	Enterprise dictionary	Publish conformed definitions.

High Level Flow (Analogy)

Identify shared entities → Standardize attributes → Assign stable keys → Align history rules → Validate → Publish.

Python Style Pseudocode

```
def conform_customer(src):
    return {
        "customer_key": lookup_surrogate(src["customer_id"]),
        "segment": standardize_segment(src["segment"]),
        "region": normalize_region(src["region"])
    }
```

Design Principles (Two Word Logic)

- **Shared semantics**
- **Stable identity**
- **Cross domain reuse**
- **Consistent history**
- **Enterprise alignment**

Interview Ready Summary

“Conformed dimensions like Customer, Product, and Date ensure consistent reporting across Sales, Marketing, and Finance. I standardize attributes, apply stable surrogate keys, align SCD rules, and publish enterprise definitions to maintain cross domain consistency.”