Database Management

Foundational data modeling

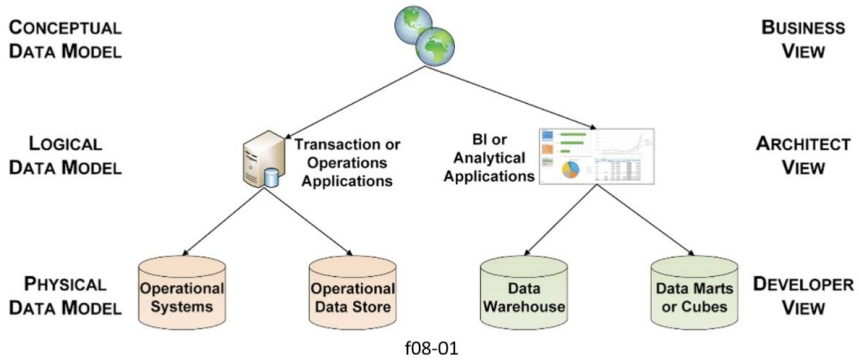
Data Models

- Definition:
  - Representation of the data structure based on business requirements.
  - Provides a template for building the database architecture.
- Types of Data Models:
  - Conceptual Model – high-level, focuses on business requirements and entities.
  - Logical Model – defines structure (tables, attributes, relationships) without focusing on technical details.
  - Physical Model – implements the database design with actual tables, columns, keys, and constraints.

Data Modeling

- Definition:
  A structured approach to designing data models.
- Purpose:
  - Build conceptual, logical, and physical models.
  - Gather and formalize business requirements.
  - Support application specifications, development, and deployment.
- Benefits:
  - Ensures alignment between business needs and database structure.
  - Improves communication between stakeholders (business + technical teams).
  - Reduces errors and redundancies in database design.

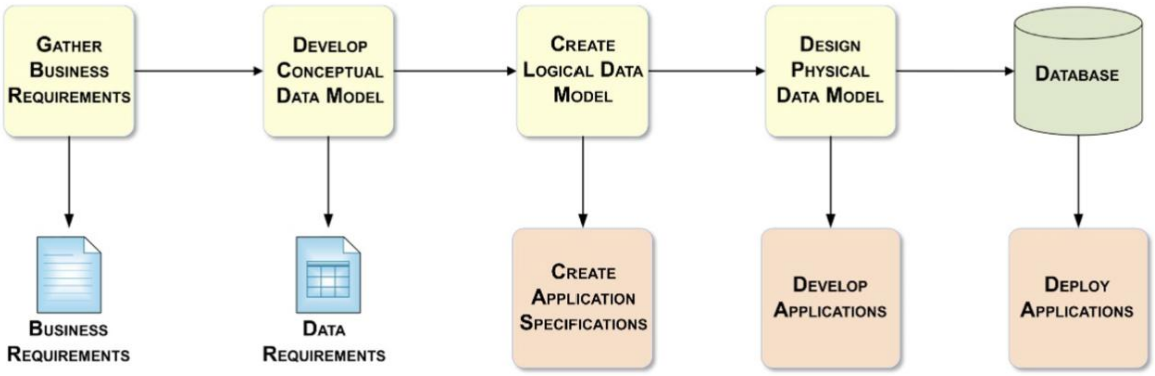
Three levels of data models



f08-01

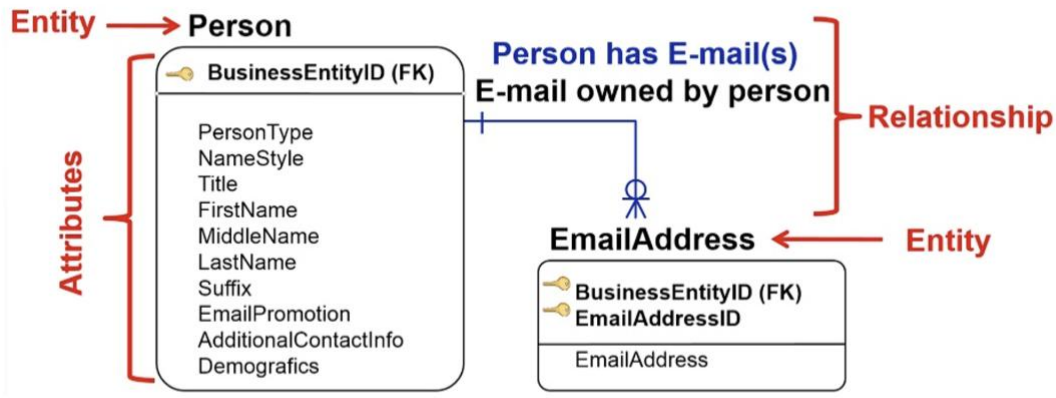
1. Conceptual Data Model
    - High-level view of data.
    - Focuses on business requirements and defining main entities.
    - Used to communicate effectively with business stakeholders.
    - Answers the question: *What data is important to the business?*
2. Logical Data Model
    - More detailed than conceptual, but independent of technology/DBMS.
    - Focus is on the design of BI or applications.
    - Defines:
        - Data objects and their attributes.
        - Relationships between entities.
    - Helps understand the structure of the data without actual implementation.
    - Answers the question: *How should the data be structured logically?*
3. Physical Data Model
    - Implementation level of the data model.
    - Translates the logical model into actual database structures:
        - Tables
        - Columns
        - Keys
        - Constraints
        - Indexes
    - Requires in-depth knowledge of the specific DBMS.
    - Developed mainly by database developers/engineers.
    - Answers the question: *How will the data be stored in the database?*

Data modeling workflow



f08-02

Er building blocks



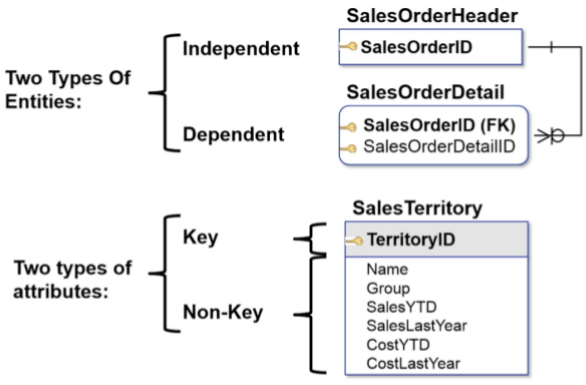
f08-04

Entities

- Independent entities: Do not need other entities to exist
- Dependent entities: Child records that need a parent to exist

Attributes

- Key (uniquely identify an attribute)
- Non-key (does not uniquely identify an attribute



f08-05

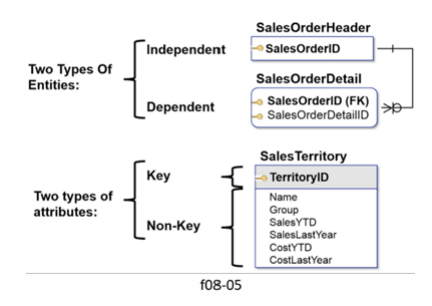
Independent Entities (Strong Entities)

- These can exist on their own, without needing another entity.
- They usually represent the main objects in a system.
- Example:
    - Customer (a customer exists even without placing an order).
    - Product (a product exists even if no one buys it).

Dependent Entities (Weak Entities)

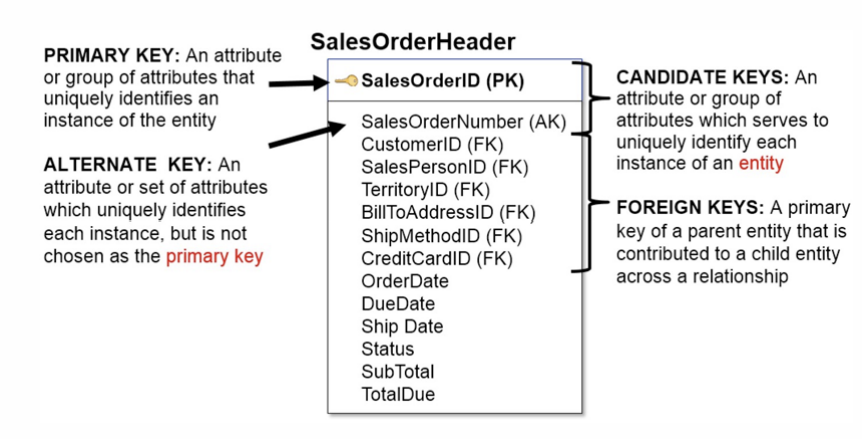
- These cannot exist on their own; they depend on another entity (a “parent”).
- They are sometimes called child entities.
- Example:

- Order Item depends on an Order (an order item doesn't exist without an order).
- Payment depends on a Rental (a payment needs a rental transaction to exist).



f08-05

Keys



Relationship Cardinality

One-to-One (1:1)

- Definition: One record in Entity A matches with exactly one record in Entity B.
- Example:
  - One employee has one employee ID card.
  - One person has one passport.
- 🔑 Usually represented by putting a unique foreign key in one of the tables.

One-to-Many (1:N)

- Definition: One record in Entity A can be related to many records in Entity B.
- Example:
  - One customer can have many sales orders.
  - One teacher can teach many students.
- 🔑 The "many" side gets a foreign key to link back to the "one."

Many-to-One (N:1)

- Definition: Many records in Entity A relate back to one record in Entity B.
- Example:
  - Many survey responses can belong to the same question.
  - Many employees can belong to the same department.
- 🔑 Technically, this is the reverse view of One-to-Many.

Many-to-Many (M:N)

- Definition: Many records in Entity A can be related to many records in Entity B.
- Example:
  - Many students can enroll in many courses.
  - Many diagnoses can require many procedures for treatment.
- 🔑 In databases, this is usually broken down with a junction (bridge) table.
  - Example: Student_Course table with Student_ID + Course_ID.

Identifying relationships

The parent entity's primary key helps identify rows in the child entity (i.e., the primary key of the parent entity must be a part of the primary key of the child entity).

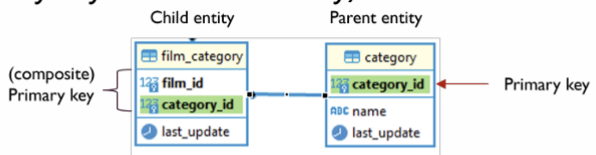


Figure: Identifying relationship (from dvdrental database)

- "category_id" in the "category" table can exist independently
- "category_id" in the "film_category" table helps identify rows in the "film_category" table.

Non-identifying relationships

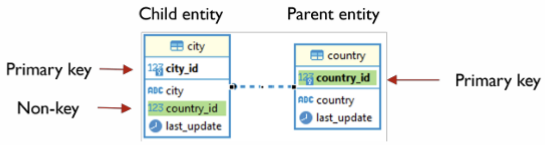


Figure: Non-identifying relationships (from dvdrental database)

- Non-identifying mandatory relationship: "category_id" cannot be NULL The parent entity primary key must occur in the child entity but it does not identify it uniquely
- Non-identifying optional relationship: "category_id" can be NULL The parent entity primary key can optionally occur in the child entity and it does not identify it uniquely

Many-to-many relationship

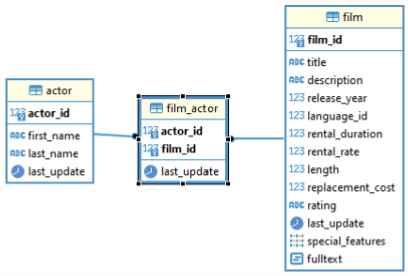


Figure: Many-to-many relationship (from dvdrental database)

- Most difficult to implement, and often handled with a "bridge" table
- One actor can be in many films, and one film can have many actors

Recursive Relationships

A recursive relationship is when an entity has a relationship with itself.

Example:

- Employee has a Manager who is also an Employee.
- The table will have EmployeeKey (unique ID) and ManagerKey (foreign key back to EmployeeKey).

This is usually non-identifying and optional (an employee *may or may not* have a manager).

Referential Integrity (RI)

RI = Rules that keep relationships valid and consistent in a database.

- Key point: You cannot insert, update, or delete data if it breaks the defined rules.

RI Rules include:

1. Cardinality rules (1:1, 1:N, M:N).
2. Relationship type rules (identifying, non-identifying, optional).

How RI is enforced:

1. Database constraints (e.g. FOREIGN KEY in SQL).
2. ETL code checks (best practice in BI, since ETL often loads data from different sources).

Normalization

Process of organizing data to reduce redundancy and ensure consistency.

Goal = transactional integrity.

Downside = complex models, performance issues.

- In operational systems → aim for 3NF.
- In BI/dimensional models → normalization is not the goal (denormalization is often preferred for query speed).

First Normal Form (1NF)

- Rule: No repeating groups of attributes.
- If an attribute can occur more than once → move it to a new entity.
- Example:
    - A Student table with multiple courses in one row → split into Student + StudentCourse entity with a composite key (StudentID, CourseID).

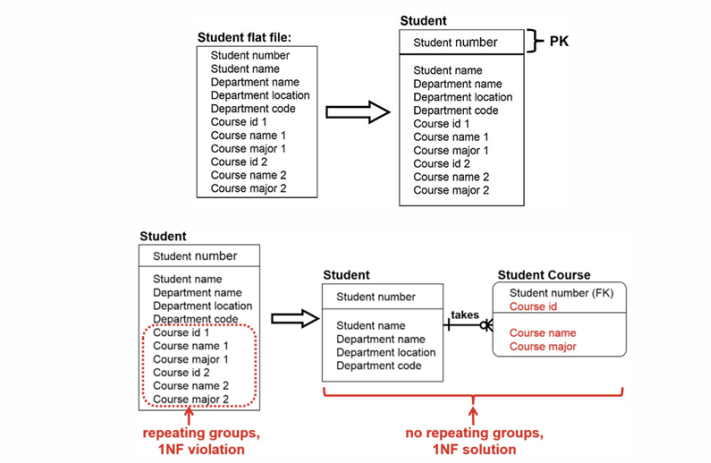
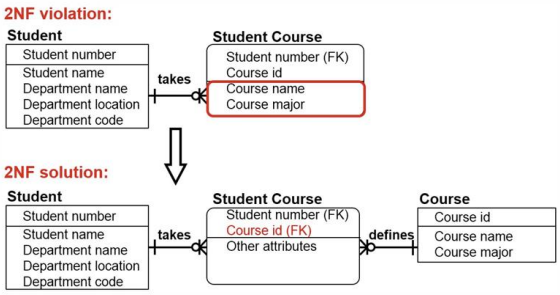


Figure: Student number and course id together forms the primary key (a composite primary key) in the Student Course entity

Second Normal Form (2NF)

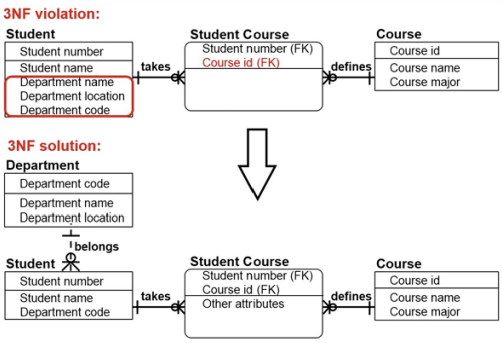
- Rule: No partial dependency of non-key attributes on part of a composite key.
- If found → move those attributes into a new entity.
- Example:
    - In StudentCourse(StudentID, CourseID, CourseName, CourseMajor)
    - CourseName and CourseMajor depend only on CourseID, not on the full key → violation.
    - Solution: Move them to a separate Course entity.



"Course id" is a part of the composite primary key. Both "Course name" and "Course major" depend on "Course id", and therefore partially depend on the primary key, which results in a 2NF violation

Third Normal Form (3NF)

- Rule: No transitive dependency (non-key attributes depending on other non-key attributes).
- If found → move them into a new entity.
- Example:
    - In Employee(EmployeeID, DepartmentName, DepartmentCode, DepartmentLocation)
    - Department attributes depend on each other, not directly on EmployeeID.
    - Solution: Move them to a new Department entity.



As "Department name", "Department location", and "Department code" depends on each other, they are split into their own entity

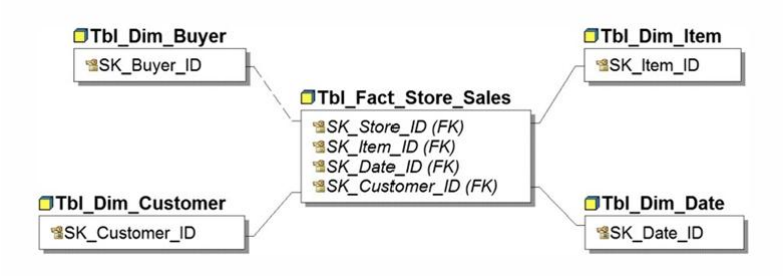
Quick Summary of NF Rules:

- 1NF: No repeating groups.
- 2NF: No partial dependency on a composite key.
- 3NF: No non-key depends on another non-key.

Dimensional modeling

Purpose of dimensional modeling: Enable BI reporting, analysis, and query

Key concepts: Facts and dimensions



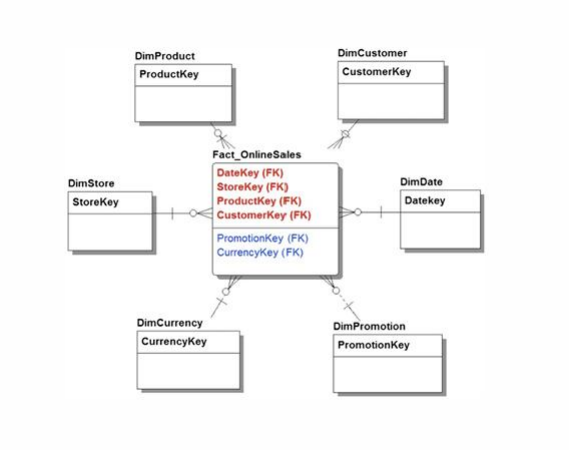
Facts

- Definition: Facts are measurements of business activity.
- Examples: Sales revenue, inventory levels, number of products sold.
- Characteristics:
    - Generally numeric.
    - Stored in fact tables, which can grow very large.
    - Measures must be clearly defined; different business units might define the same measure differently.

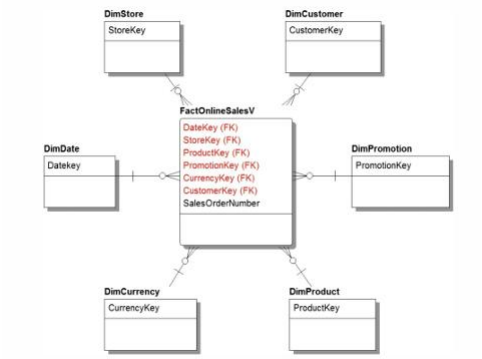
Fact table keys

- Columns
    - Fact tables have two types of columns:
        - Keys
            - Foreign keys pointing to dimension tables.
            - Should not contain null values.
            - Often a one-to-many relationship with dimensions.
            - Primary key of fact table = often a compound/composite key (combination of foreign keys).
        - Measures
            - Numeric values representing the actual facts of business events.
            - Always relate to a specific event (sale, invoice, shipment).
            - Grain of measurement: The level of detail (e.g., one row per product sold in a sale).
            - Best practice: store at lowest possible grain; avoid pre-aggregated data.

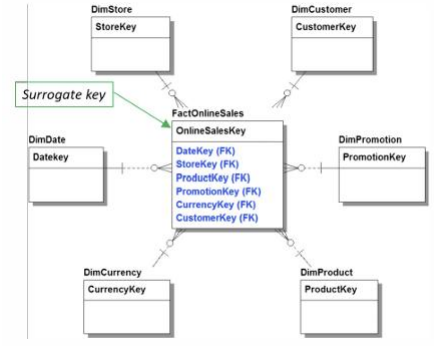
- Compound/Composite Key
    - A compound key is a combination of foreign keys used to uniquely identify each row.
    - Ensures that each fact record is unique based on the combination of its dimensional references.



1. Degenerative Dimension
    - When a row in a fact table cannot be uniquely identified by foreign keys:
    - Include a degenerative dimension, an attribute that does not have its own dimension table.
    - Often comes directly from the operational system (e.g., invoice number).



2. Surrogate Key
    - Optional alternative to using operational system keys.
    - Purpose: uniquely identify rows within the fact table.
    - Has no meaning outside the dimensional model.
    - Can be generated automatically using IDENTITY or similar mechanisms.



Types of Facts (Additivity)

1. Additive Facts
    - Can be summed across all dimensions.
    - Example: Number of products sold (sum over customers, stores, dates).
2. Semiadditive Facts
    - Can be summed across some dimensions, but not all.
    - Example: Inventory level (cannot sum meaningfully over time).
3. Nonadditive Facts
    - Cannot be summed across any dimensions.
    - Example: Temperatures, percentages.
4. Transaction Fact Tables
    - One record per business event.
    - Example: Each individual sale or invoice.
5. Periodic Snapshot Fact Tables
    - One record per specific point in time.
    - Example: Account balance at the end of the month.
6. Accumulating Snapshot Fact Tables
    - One record per process or lifecycle, with multiple date columns for events.
    - Example: Order lifecycle table with order date, payment date, shipment date, delivery date.

Dimension

Definition: Define the context of facts (who, what, where, when, why).

Purpose: Store descriptive information to avoid repeating it in fact tables.

Benefits:

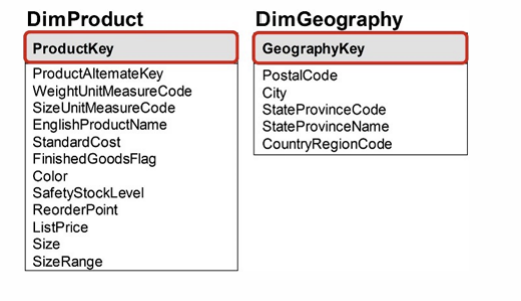
- Alter dimensional attributes without changing facts.
- Easily filter, group, and analyze data.

Characteristics of Good Dimensions

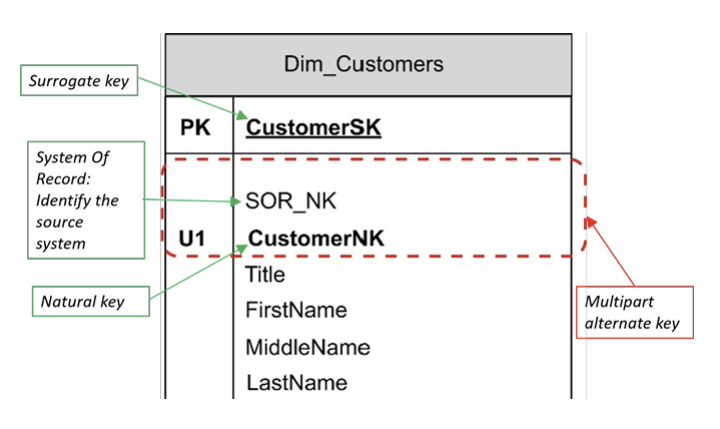
- Descriptive – easy to understand.
- Complete – no missing values.
- Unique – values can identify rows.
- Hierarchies – one-to-many relationships for drill-down/up analysis (e.g., Country → State → City).

Dimension Keys

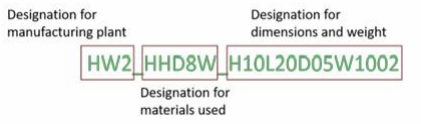
- **Primary Key:** Single column uniquely identifying each row.



- **Surrogate Primary Keys:**
  - Best practice to use instead of operational system keys.
  - Only purpose: identify rows in the dimension.
  - Source system key is stored as an alternate key.
  - If multiple source systems exist, add a column for the source system.



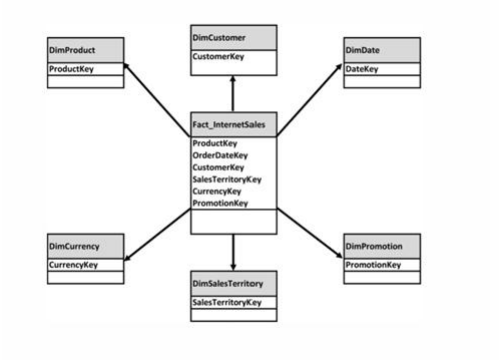
- **Smart Keys:**
  - Alphanumeric keys with embedded meaning.
  - Can lead to inconsistencies → **surrogate keys preferred**.



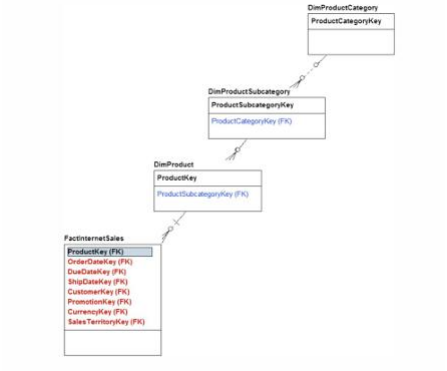
- **Handling Nulls:**
  - Foreign keys in fact tables should **never be null**.
  - Solution: create special “unknown/missing/invalid” rows in the dimension (e.g., key = -888 for missing).

Schemas

- Star Schema
    - Fact table normalized; dimension tables denormalized.
    - Easy to query, good performance.



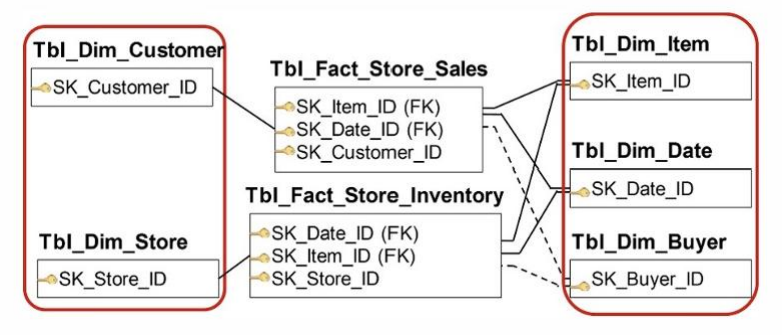
- Snowflake Schema
    - Hierarchies within dimensions are normalized.
    - Fact table points to lowest level of hierarchy.
    - Harder for business users → generally discouraged.



- Multidimensional Schema (OLAP Cube)
    - A database structure where facts (measures) are stored in cells, and dimensions define the context (e.g., product, date, geography).
    - Characteristics:
        - Each cell in the cube represents a measure.
        - Stored in multidimensional databases (OLAP cubes).
        - Structure is often proprietary and can vary widely.
        - Different from relational databases which rely on SQL and standardized tables.
    - Capabilities:
        - Drill-down / drill-up: Navigate through hierarchies (e.g., Country → State → City).
        - Summarization / aggregation: Summarize data along dimensions (e.g., total sales by region).
        - All facts, dimensions, and hierarchies are stored in a metadata repository.
        - Can be visualized as a cube where each axis is a dimension and each cell contains a fact.

Multifact Star Models

- Definition: Fact tables with multiple measures or business events.
- Requirements:
    - Conformed Dimensions: Dimensions shared by multiple fact tables.
        - Example: A Date dimension shared between Sales and Inventory fact tables.
        - Ensures consistency across different fact tables.
- Enterprises often need multiple fact tables to model different business processes.
- Using conformed dimensions makes reporting and analysis more consistent and reliable.

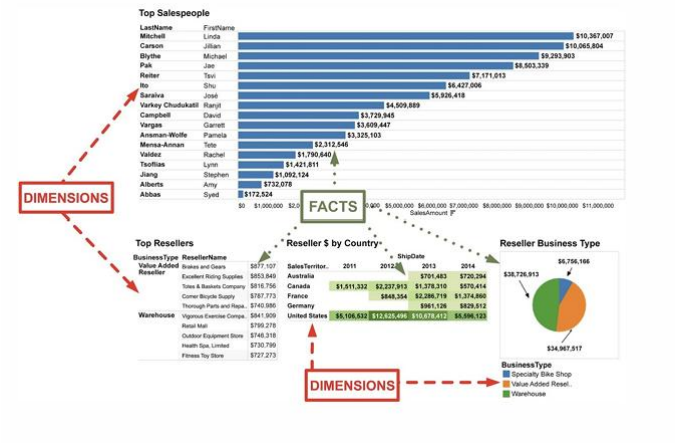


ER (Entity-Relationship) vs. Dimensional Modeling

Aspect	Dimensional Modeling (BI)	ER Modeling (Operational)
Purpose	Standard for BI, reporting, and analysis	Standard for operational systems, transactional processing
Storage	Higher storage requirements (redundancy allowed)	Efficient storage, minimal redundancy
Query Performance	Generally higher for analytical queries	Optimized for transactions, not analytical queries
Data Consolidation	Can consolidate inconsistent data from multiple sources	Eliminates inconsistent data at source
Maintenance	More maintenance due to redundancy and multiple fact/dimension tables	Less maintenance
Best Use	Generating information **important for business decisions**	Handling day-to-day operations

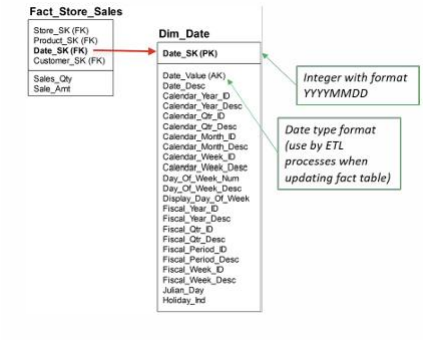
Dimensional Modeling

Well suited to the generating the information important for the business people



Date Dimension

- Almost all facts are linked to a date.
- Provides filtering and grouping without extra calculations.
- Can be used as a conformed dimension for consistency across multiple fact tables.

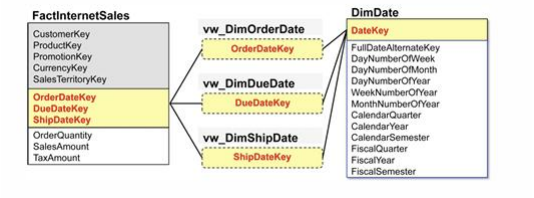


Time Dimension

- Time-of-day is usually not necessary, but if needed:
    1. Store time-of-day as a DATETIME in the fact table.
    2. Create a time-of-day dimension with descriptive periods (e.g., morning, lunch).
- If no time dimension exists, a value-band dimension can be created for time ranges.

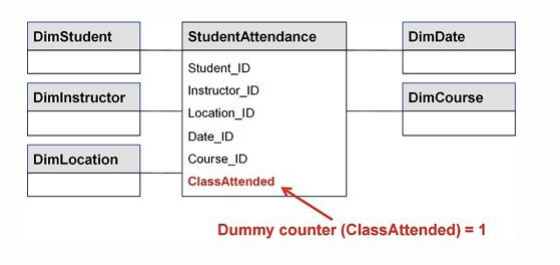
Role-Playing Dimensions

- Same dimension used in multiple contexts (e.g., OrderDate and ShipDate referencing the same Date dimension).
- Fact table will have multiple foreign keys to the same dimension.
- Best practice: create views for each role to simplify queries and join



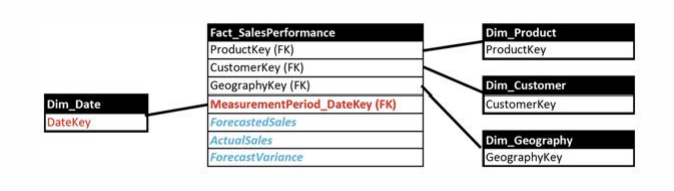
Event Tables / Factless Facts

- Definition: Fact tables without numeric measures.
- Purpose: capture events or conditions (e.g., attendance, registrations).
- Can include a dummy counter (BIT or 0/1) for counting purposes.



Consolidated Fact Tables

- Combine multiple fact tables for easier reporting.
- Requirements:
    1. Conformed dimensions shared across fact tables.
    2. Same grain (level of detail), often by aggregating underlying fact tables.
- Best practice: maintain separate fact tables in the DW; create consolidated tables in BI layer for reporting.



Hierachies

Types of Hierarchies

Hierarchy Type	Depth/Level Consistency	Key Feature & Analogy	Solutions for Data Gaps
Balanced (Fixed)	**Consistent Depth:** All branches have the **same number of levels**.	A perfectly straight tree; every path from root to leaf has the same length.	No gaps to address. (e.g., Year → Month → Day)
Ragged	**Consistent Depth** (Structure), but **Inconsistent Data** (Gaps).	A tree where some leaves are missing a branch segment. **Levels are skipped** for some records.	1. Use a **Default Value Indicator** (e.g., "Not applicable"). 2. **Repeat the value** from the level above the gap.
Unbalanced	**Inconsistent Depth:** Branches have a **varying number of levels**.	A real-world organization chart where department structures differ.	None needed, as the **parent-child relationship** is consistent even with varying length. (e.g., Manager → Employee org chart)

Variable-Depth Hierarchy Definition

- Definition: A blanket term referring to any hierarchy that is not Balanced (fixed). It includes both:
    - Ragged Hierarchies: Consistent depth, but with gaps in the data.
    - Unbalanced Hierarchies: Inconsistent depth (varying number of levels).

II. Approaches to Handling Variable-Depth

There are two primary methods for modeling these complex relationships:

Approach	Description	Technical Implementation	Drawbacks/Limitations
1. Recursive Pointer (or Adjacency List)	Embeds the **parent-child relationship** directly within the **dimension table itself**.	The table has an attribute (column) that is a **Foreign Key** pointing back to the **Primary Key** of the same table.	**Inefficient for Analysis:** Standard **SQL** and many **BI Tools** cannot easily navigate this structure from top to bottom (root to leaf). This is especially true if the hierarchy is both ragged and unbalanced.
2. Bridge Table (or Path Enumeration)	(The slides mention the bridge table, but don't detail it yet.) This method uses a separate **many-to-many junction table** to map every possible hierarchical path (ancestor to descendant) and the distance between them.	(Awaiting further slides for technical details.)	(Generally considered more complex to load/maintain, but far superior for querying.)

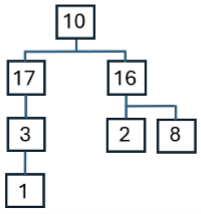
Recursive Pointer

Concept	Definition & Role	Example/Implementation	Drawback Reminder
Recursive Pointer	A modeling technique where an entity (row) is related to **another instance of the same entity** (another row in the same table).	**Employee → Manager Relationship:** An employee entity is the "child" to a manager entity, which is also an employee entity and the "parent."	Difficult for standard SQL/BI tools to traverse fully, especially with deep or complex (unbalanced/ragged) structures.
Relationship Type	**Non-Identifying, Non-Mandatory:**	The foreign key (e.g., `Manager Key`) **does not** form part of the primary key, and the relationship is **optional** (e.g., the top-level entity, like the CEO, has a `NULL` foreign key).	
Data Record	The manager's ID is recorded in **two columns**:	1. In the **Employee Key** column (as the employee's ID). 2. In the **Manager Key** column (as the parent/manager of another employee).	

Recursive pointer

Column name	Type
Employee_key	Primary key
Manager_key	Foreign key
First_name	

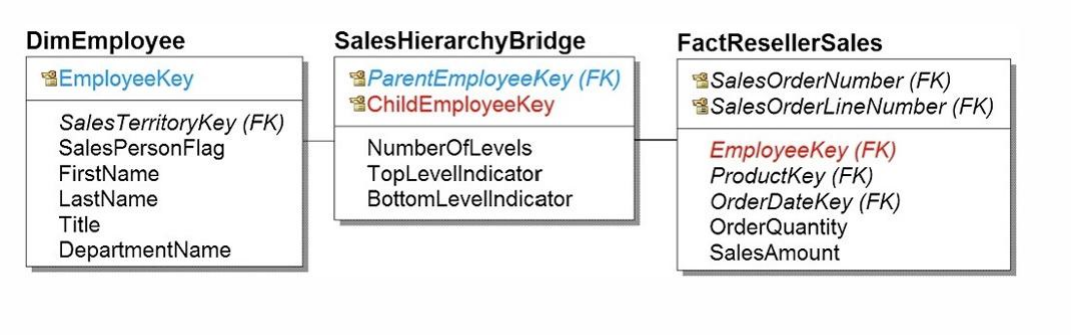
Hierarchy by Employee_key



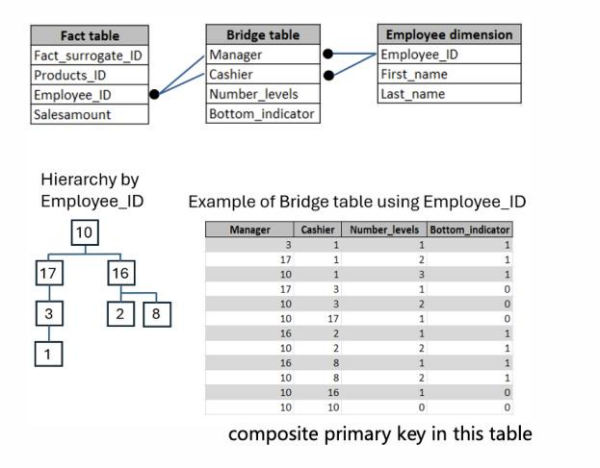
Employee_key	Manager_key	First_name
1	3	Joe
3	17	Alice
17	10	Bob
2	16	Jane
8	16	Michael
16	10	Emma
10	NULL	David

Variable-depth hierarchies: Bridge table

- The relation between ParentEmployeeKey and ChildEmployeeKey gives the hierarchical relation

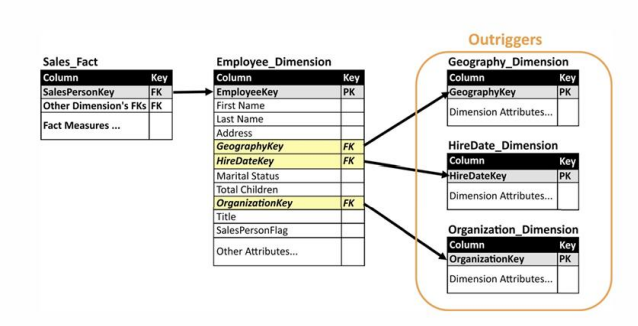


- Hierarchical relationships within a dimension (e.g., manager and subordinate roles) can be modeled using a bridge table, which links different levels of the hierarchy.



Outrigger tables

- Attributes related to each other are typically placed in one dimension
- Outriggers can be constructed due to e.g. different data sources or times of update
- Outriggers also eliminate redundant data and the amount of data that needs to be changed upon updates



Slowly Changing Dimensions (SCDs)

1. The Core Problem

Operational systems (like the ones used for daily business) only care about being up-to-date ("as is"). When a customer's address changes, they simply overwrite the old address, and the history is lost.

However, for management analysis (in a data warehouse), we need to track backwards in time to see historical sales based on the customer's status "as was" at that time. SCDs solve this by ensuring historical data is always compatible with the correct context.

2. The Solution: SCD Types

SCDs are a set of seven different techniques (SCD Type 0 to SCD Type 7) used to manage how changes to descriptive data (like names, addresses, or product categories) are recorded over time.

The choice of technique is driven entirely by business analytical need. Key considerations include:

- Which dimensions (e.g., Customer, Product) need tracking?
- Which attributes within those dimensions need change history?
- What level of detail is needed for analysis?

3. SCD Type 0: Keep Original

- Action: The original value is kept forever, and any changes to the attribute are completely ignored.
- Result: Historical facts will always be tied to the original, static value.
- Use: Reserved for attributes that are truly static (like a birth date) or where history is completely irrelevant (it's the least likely type to use for changing data).

SCD type 1

Overwrite existing data

- Overwrites historical values with most current values (thereby disregarding old values)

Initial Row

Employee Key	First Name	Last Name	Title	HireDate	Marital Status	Base Rate	Row Created	Row Modified
12345	Guy	Gilbert	Prod Technician	08/01/07	S	$12.45	08/01/07	

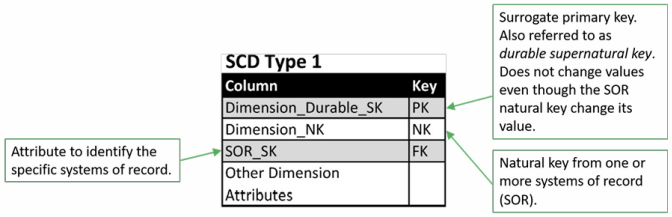
Current Row

Employee Key	First Name	Last Name	Title	HireDate	Marital Status	Base Rate	Row Created	Row Modified
12345	Guy	Gilbert	Prod Supervisor	08/01/07	S	$25.00	08/01/07	01/02/14

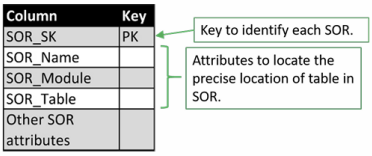
- Shortcomings
  - Cannot track back in time
  - Needs to reaggregate data in summary tables or OLAP cubes

SCD type 1 schema

- We can support our dimension by linking to the systems of record (SOR)



- A SOR dimension enables us to identify the system of record



SCD type 2:

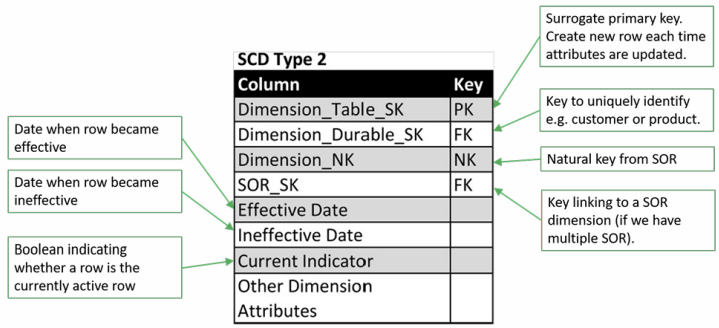
Create new row

- Create a new row when attribute change
- Enables to track records
- Whether we should perform ”as is” or ”as was” analysis is a business decision

Durable_ID	ID	Last Name	Title	Hire Date	Marital Status	Base Rate	Effective Date	Ineffective Date	Current
1	1	Gilbert	Prod Technician	2017-08-01	S	$12.45	2017-08-01	2018-09-02	0
1	2	Gilbert	Prod Technician	2017-08-01	M	$12.45	2018-09-02	2019-03-12	0
1	3	Gilbert	Prod Technician	2017-08-01	M	$17.50	2019-03-12	2019-06-20	0
2	4	Jane	Prod Technician	2019-04-01	S	$12.45	2019-04-01	9999-01-01	1
1	5	Gilbert	Prod Supervisor	2017-08-01	M	$25.00	2019-06-20	9999-01-01	1

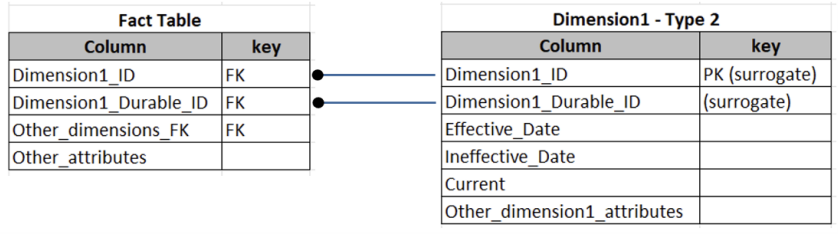
Ineffective Date = Date when the data was updated or until when the old dat was valid

SCD type 2 schema



Dimension Table SK	Dimension Durable SK	Dimension NK	SOR SK	Effective Date	Ineffective Date	Current Indicator	Dimension Attributes
20123	12345	ABC123	2	1/2/2007	2/3/2009	N	
21357	12345	ABC123	2	2/4/2009	6/22/2012	N	
23488	12345	ABC123	2	6/23/2012	1/1/2014	N	
24719	12345	ABC123	2	1/2/2014	12/31/9999	Y	

- SCD Type 2 also allow for including "what will be" values (e.g. predicting future sales)
- Fact table is linked to dimension with two foreign keys



SCD type 3:

Track changes using separate columns but overwrite rest

- Has extra column(s) for the attribute(s) that is important to track
- Loss of intermediary information (only have original and current information)

Initial Row

Employee Key	First Name	Last Name	Title	Original Title	HireDate	Marital Status	Base Rate	Effective Date	Ineffective Date
12345	Guy	Gilbert	Prod Technician	Prod Technician	08/01/07	S	$12.45	08/01/07	12/31/99

Current Row

Employee Key	First Name	Last Name	Title	Original Title	HireDate	Marital Status	Base Rate	Effective Date	Ineffective Date
12345	Guy	Gilbert	Prod Supervisor	Prod Technician	08/01/07	S	$25.00	01/02/14	12/31/99

Indicates when the (new) title is effective from

SCD type 3 schema

- Durable supernatural key: "Durable" because values does not change even if SOR values change, and "Supernatural" to distinguish it from the SOR natural key

SCD Type 3

Column	Key
Dimension_Durable_SK	PK
Dimension_NK	NK
SOR_SK	FK
Current_Value	
Original_Value or Previous_Value	
Other Dimension Attributes	

Surrogate key/durable supernatural key

SOR natural key

Key linking to a SOR dimension (used if multiple SOR is used)

Current value of attribute using SCD type 3 technique

Original or previous value of attribute using SCD type 3 technique

- Only need a single foreign key from fact table

Fact Table

Column	Key
Dimension_Durable_SK	FK
Other Dimension's FKs	FK
Other Attributes...	

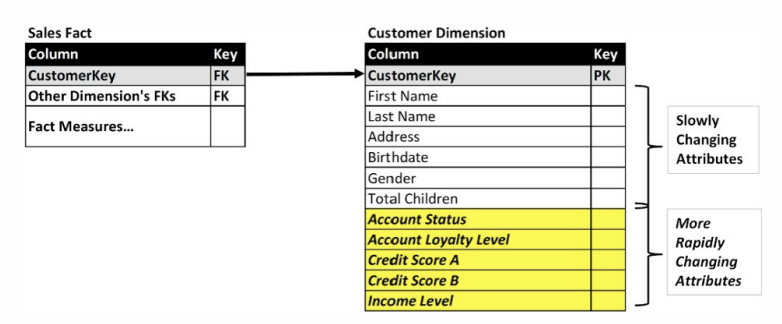
Dimension - Type 3

Column	Key
Dimension_Durable_SK	PK
Dimension_NK	NK
SOR_SK	FK
Current_Value	
Original_Value or Previous_Value	
Other Dimension Attributes	

SCD type 4

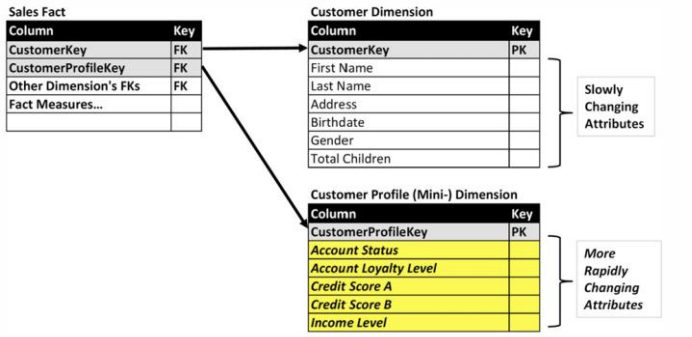
Add mini-dimension

- Can be used when some attributes changes slowly over time while other attributes change rapidly
- Having rapidly and slowly changing attributes in same table can affect both loading data and BI query performance
- A solution can be to add mini-dimension (for the rapidly changing attributes)



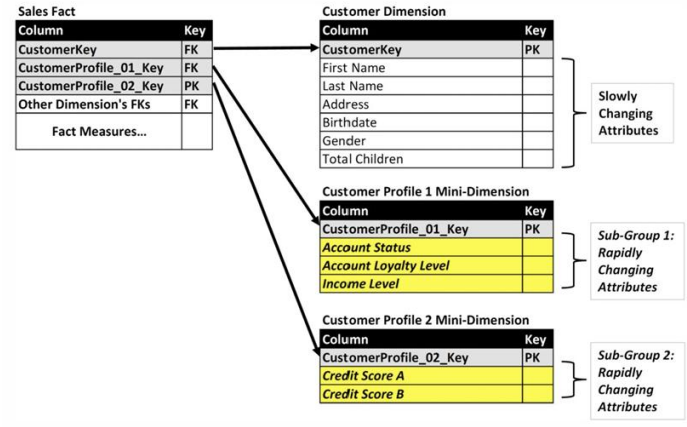
Single mini Dimension

- Split approach- single mini-dimension



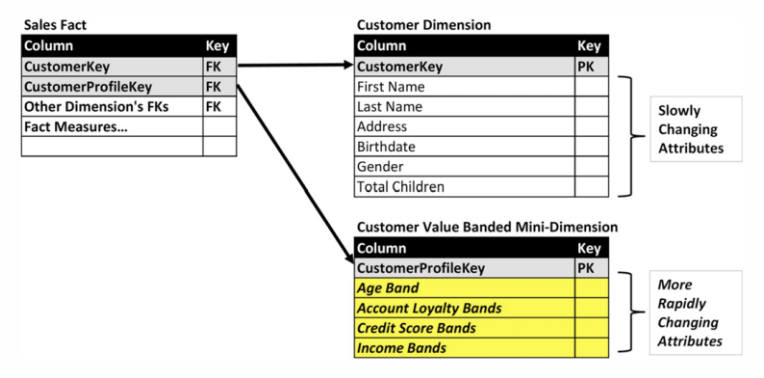
Multiple mini-dimension

- Make multiple mini-dimensions, where attributes are grouped based on different time/rate of updates



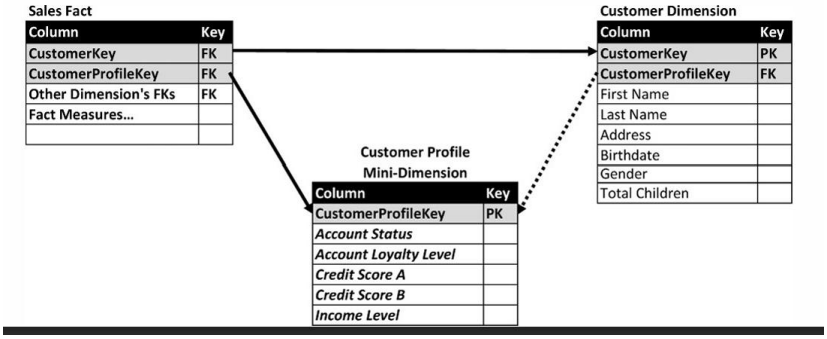
Band-valued mini dimension

- Make rapidly changing mini-dimensions contain value ranges this requires fewer updates
- Downside: A loss of information



Mini-dimensions need outrigger

- A direct connection between mini-dimension and the dimension itself facilitate
- Faster updating: Instead of adding a new row to the customer dimension when e.g. Account Status change, we can add a new row only to the mini-dimension. This requires updating the CustomerProfileKey (FK) in the customer dimension, and we could keep the old row in the mini-dimension to track historical sales.
- Less redundant information in join queries



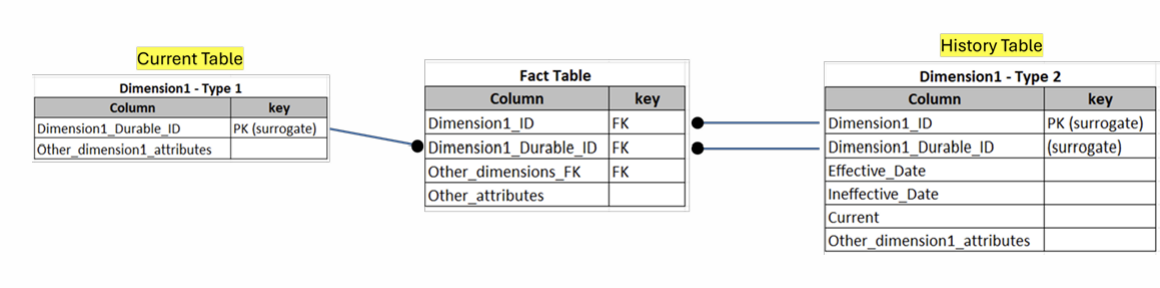
SCD type 5, 6

- SCD type 5 combines Types 4 and 1
  - Adds a new mini-dimension to SCD type 4 (so we have two minidimensions) that stores the most current information
  - Have attributes in the primary dimension that store current values of rapidly changing attributes but keep also the mini-dimension for tracking change over time
- SCD type 6 combines type 1 (overwrite data), type 2 (add rows upon updates), and type 3 (add columns for the attribute that we wish to track). We will not go deeper into this type.

SCD type 7

Use both type 1 and type 2

- Create a type 1 version of the dimension (only current values) together with a type 2 version of the dimension (create new rows)
- Enables tracking of historical data (type 2) and fast query and easy overview (type 1)
- Easy access to ”as is” analysis but still a possibility to track data over time



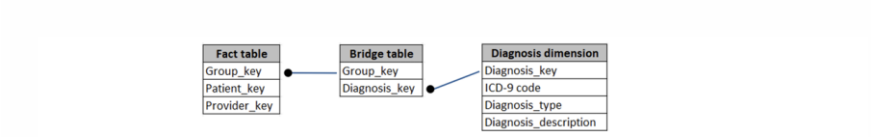
Causal and multivalued dimensions

Causal Dimension

- A Causal Dimension is simply a dimension table that stores the reason *why* a specific fact (like a sale or an event) happened. Instead of just recording *what* was bought or *when*, it records the cause (e.g., "was this sale due to an email promotion, a price match, or a newspaper ad?"). This is extremely useful for analysis because it helps management understand the drivers behind their metrics.

Multivalued Dimensions (Many-to-Many Problem)

- The Multivalued Dimension problem occurs when a single event (a fact) can be linked to multiple members of a dimension, and vice-versa.
- For example, a single Patient Visit (Fact) might result in several Diagnoses (Dimension). In a standard data warehouse, a Fact Table can only point to one dimension member.
- The slides offered three ways to deal with this complex relationship:
    - Pick one value: e.g. only pick one diagnosis for a patient (downside: loss of information)
    - Extend the attributes: Have e.g. 10 attributes that can store patient diagnosis (downside: repeating groups, upper limit on number of diagnosis and hard to find total number different diagnosis)
    - Bridge table: Accommodate the many-to-many relationship
        - Note: The bridge table holds a composite primary key, consisting of the columns ”Group key” and ”Diagnosis key”



Group_key	Patient_key	Provider_key
10	1	23
2	2	12
10	3	71
2	4	23
15	5	12

Group_key	Diagnosis_key
10	4
10	1
2	2
15	4
15	5

Diagnosis_key	ICD-9 code	Diagnosis_type	Diagnosis_description
1	dm-342	Chronic	Diabetes mellitus
2	es-101	Chronic	Essential hypertension
3	gr-003	Acute	Gastroesophageal reflux
4	au-001	Chronic	Asthma, unspecified type
5	os-001	Chronic	Osteoarthritis

Junk dimensions

- Used to handle many different low-cardinality system codes, identifiers etc.
- I.e. many different types of codes that each does not have many distinct values. Instead of putting them into their own small seperate dimensions, they are combined in one junk dimension

Promotion status codes

Order status codes

Price type codes

Junk_ID	Group_Name	Code	Code_Description	SOR_Table_Name	SOR_Field_Name
201	Promotion Status	A	Approved	imm_promo_header	status
202	Promotion Status	C	Cancelled	imm_promo_header	status
203	Promotion Status	D	Deleted	imm_promo_header	status
204	Promotion Status	F	Pending Promo	imm_promo_header	status
205	Promotion Status	N	Running	imm_promo_header	status
206	Promotion Status	R	Released	imm_promo_header	status
207	Promotion Status	X	Expired	imm_promo_header	status
208	Promotion Status	Z	Frozen	imm_promo_header	status
401	Order Status	A	Tech-Approved	imm_item_order_wkbk	order_status_code
402	Order Status	E	Estimated	imm_item_order_wkbk	order_status_code
403	Order Status	H	On Hold	imm_item_order_wkbk	order_status_code
404	Order Status	N	Not Reviewed	imm_item_order_wkbk	order_status_code
405	Order Status	P	Pending	imm_item_order_wkbk	order_status_code
406	Order Status	R	Released	imm_item_order_wkbk	order_status_code
407	Order Status	S	Skipped	imm_item_order_wkbk	order_status_code
501	Price Type	C	Cost Base	bn_item	price_type_code
502	Price Type	F	Freight Pass Thru	bn_item	price_type_code
503	Price Type	N	Net Price	bn_item	price_type_code
504	Price Type	R	Retail	bn_item	price_type_code

Other types of dimensions

- Value band reporting: Create a table with value bands
- Alternate dimensions: how to handle that different business functions have different definitions/usages of dimensions
    - Hot swappable dimension: Create alternative dimensions for each business function or process (e.g. 3 alternate product dimensions)
    - Custom dimension groups: Create an outrigger from the dimension that identifies each grouping. The outrigger will have a column for each group. A foreign key in the dimension can then link the product to the outrigger, and the various groupings can be identified from the outrigger columns