ETL

Dr. Yashvardhan Sharma

Department of Computer Science & Information Systems

BITS, Pilani

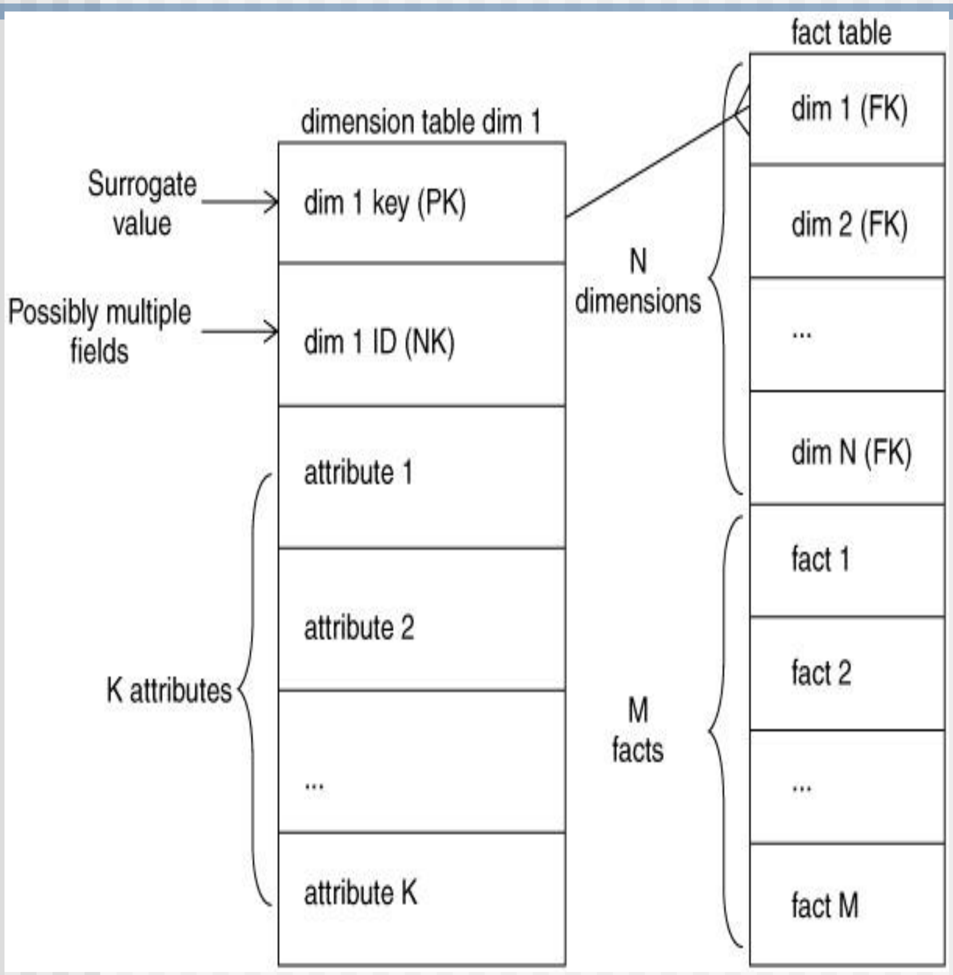
L9 (Lec 15)

5. Delivering Dimension Tables

Loading Dimensions

- Physically built to have the minimal sets of components
- The primary key is a single field containing meaningless unique integer – Surrogate Keys
- The DW owns these keys and never allows any other entity to assign them
- De-normalized flat tables – all attributes in a dimension must take on a single value in the presence of a dimension primary key.
- Should possess one or more other fields that compose the natural key of the dimension

The basic structure of a dimension



- Primary key (PK)
  - Mostly a surrogate key
  - Foreign Key to Fact Tables
- Natural key (NK)
  - Meaningful key extracted from source systems
  - 1-to-1 relationship to the PK for static dimensions
  - 1-to-many relationship to the PK for slowly changing dimensions, tracks history of changes to the dimension
- Descriptive Attributes
  - Primary textual but numbers legitimate but not numbers that are measured quantities
  - 100 such attributes normal
  - Static or slow changing only
  - Product price -- either fact or dimension attribute

Smart keys in place of Surrogate keys

- Smart Key
  - Natural Key concatenated with timestamp
- A wrong approach: Reasons
  - Keys assume responsibility
    - Changes may happen to natural key, causing the change of all fact table records referring it.
  - Poor performance
    - Key may take CHAR or VARCHAR type, comparison is slower.
    - Joins between two CHAR / VARCHAR is slower process
  - Heterogeneous source systems
    - Common dimensions are sourced by different sources.
    - Conflict arise when second source comes into the scene to source the key.

The grain of a dimension

- Grain of the dimension means, the definition of the key of the dimension
- The grain of the customer dimension is *customer*
- Verify that a given source (file) implements the intended grain
  - Nothing should be returned by the following query from the source system/file for the fields A,B & C to form the natural key

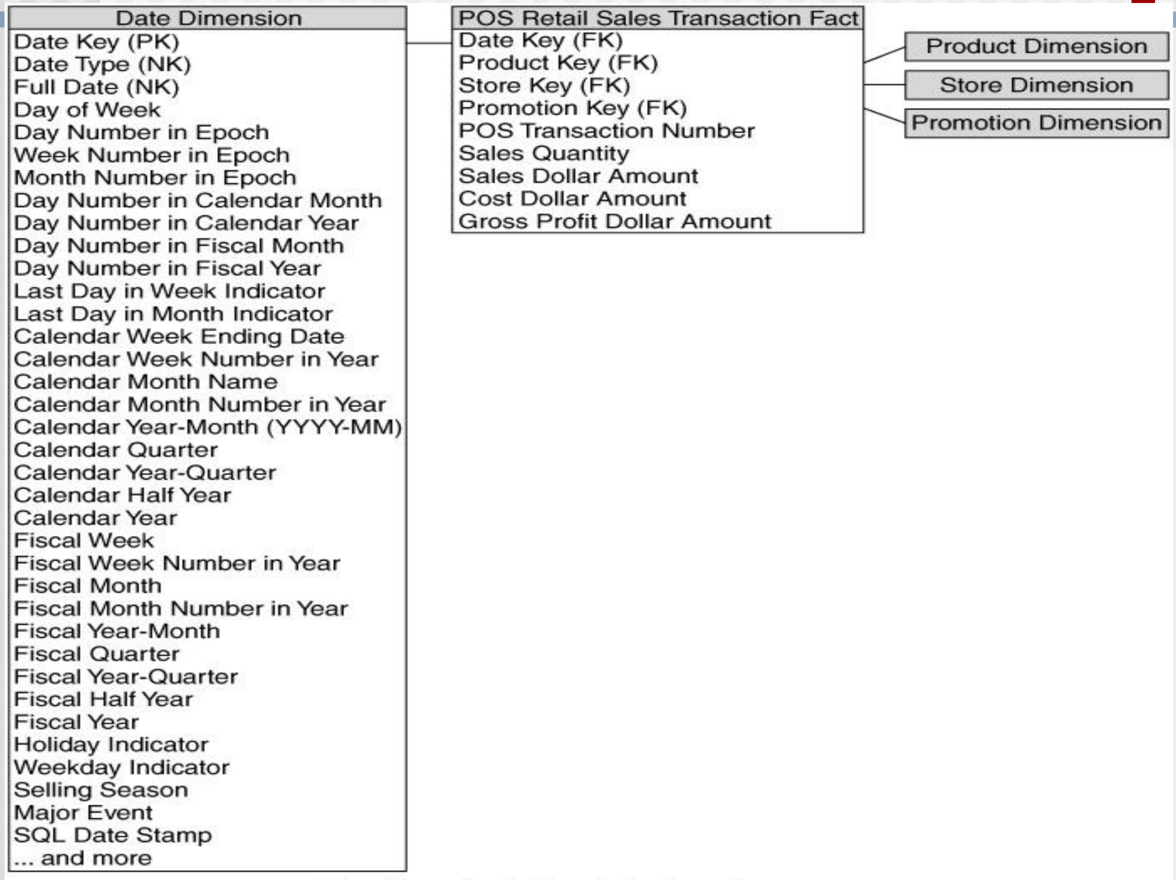
```
select A, B, C, count(*) from DimensionTableSource group by
 A, B, C having count(*) > 1
```

  - If something is returned by this, the fields A, B and C do not represent the grain of the dimension

The basic load plan for a dimension

- Data cleaning
  - Validate the data, apply business rules to make the data consistent, column validity enforcement, cross-column value checking, row de-duplication
- Data conforming
  - Align the content of some or all of the fields in the dimension with fields in similar or identical dimensions in other parts of the data warehouse
- Data Delivery
  - All the steps required to deal with slow-changing dimensions
  - Write the dimension to the physical table
  - Creating and assigning the surrogate key, making sure the natural key is correct, etc.

Date and Time Dimensions



Date dimension in the retail sales schema.

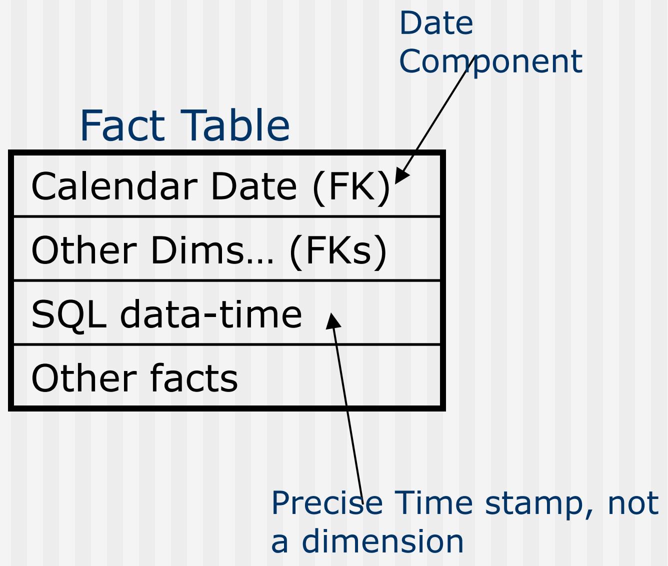
- Virtually everywhere: measurements are defined at specific times, repeated over time, etc.
- Most common: calendar-day dimension with the grain of a single day, many attributes
- Doesn't have a conventional source:
  - Built by hand, spreadsheet
  - Holidays, workdays, fiscal periods, week numbers, last day of month flags, must be entered manually
  - 10 years are about 4K rows

Date Dimension

- Note the Natural key: a day type and a full date
  - Day type: date and non-date types such as inapplicable date, corrupted date, hasn't happened yet date
  - fact tables must point to a valid date from the dimension, so we need special date types, at least one, the "N/A" date
- How to generate the primary key?
  - Surrogate Keys, whose zero value is assigned to the standard starting date of the history of records
  - Partition warehouse on date helps improve performance & allows work on the current partition such as, creating & dropping index.
  - Surrogate key 9999999 may be designed to mean N/A. This high value will keep it always in the current partition.

Other Time Dimensions

- If the grain is month, not day, then create a separate dimension table for month. Similarly, if the grain is week or some domain specific span.
- Creating month view on the date dimension is not desirable, since it would drag larger table in to the month based query. 120 records will be there for monthly grain dimension table for 10 yrs.
- When the grain is seconds, it is not advisable to create 375 million rows per year *(approx)* for every seconds instead use the design shown.



This design is preferred over creating a dimension table record for every minutes or seconds

Big and Small Dimensions

BIG

- Examples: Customer, Product, Location
- Millions or records with hundreds of fields (insurance customers)
- Or hundreds of millions of records with few fields (supermarket customers)
- Always derived by multiple sources
- These dimensions should be conformed

SMALL

- Examples: Transaction Type, Claim Status
- Tiny lookup tables with only a few records and one ore more columns
- Build by typing into a spreadsheet and loading the data into the DW
- These dimensions should NOT be conformed

JUNK dimension

- A tactical maneuver to reduce the number of FKs from a fact table by combining the low-cardinality values of small dimensions into a single junk dimension.
- Generate as you go, don't generate the cartesian product

Dimensional Roles

- The same dimension can be attached to a fact table multiple times
  - Sale has an order date, payment date, a shipping date, a return date
  - A claim transaction has multiple internal people, a claim intake, a medical nurse, a claim adjuster, a payer, etc
- End-user might be confused when they do drill-down into the dimension as to what the dimension represents, so it helps to have the dimension have different names

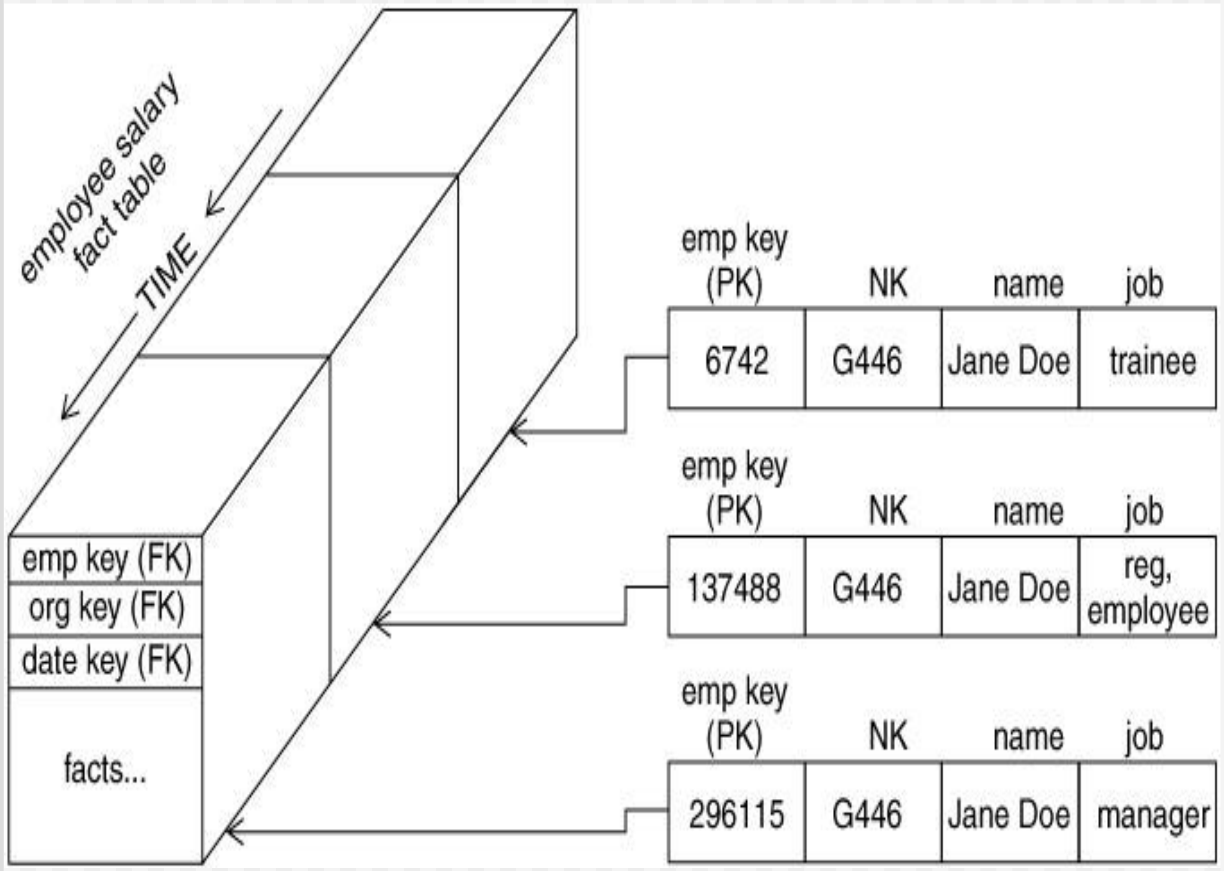
- The data loading module consists of all the steps required to administer slowly changing dimensions (SCD) and write the dimension to disk as a physical table in the proper dimensional format with correct primary keys, correct natural keys, and final descriptive attributes.
- Creating and assigning the surrogate keys occur in this module.
- The table is definitely staged, since it is the object to be loaded into the presentation system of the data warehouse.

Type-2 Slowly Changing Dimension

- When a record changes, instead of overwriting
  - create a new dimension record
  - with a new surrogate key
  - add the new record into the dimension table
  - use this record going forward in all fact tables
  - no fact tables need to change
  - no aggregates need to be re-computed

- Perfectly partitions history because at each detailed version of the dimension is correctly connected to the span of fact tables for which that version is correct

Type-2 Slowly Changing Dimensions



emp key (PK)	NK	name	job
6742	G446	Jane Doe	trainee

emp key (PK)	NK	name	job
137488	G446	Jane Doe	reg, employee

emp key (PK)	NK	name	job
296115	G446	Jane Doe	manager

- Type-2 do not change the natural key (the natural key should never change)

Type-2 SCD Precise Time Stamping

- With a Type-2 change, you might want to include the following additional attributes in the dimension
  - Date of change
  - Exact time of change
  - Reason for change
  - Current Flag (current/expired)

Type-2 SCD using CRC

- For small tables, use a brute force approach of comparing every incoming field with every field in the DW to see if anything changes
- For larger tables, use the CRC, or cyclic-redundancy checksum, a number of about 20 digits, like this:
  - Treat the entire incoming record as a string
  - Compute the CRC value of the string (some numeric value)
  - Compare the CRC value of today's record with the CRC value of yesterday's record
  - If the CRC values match, today's record is identical to yesterday's record
  - If the CRC values do not match, do a field by field comparison to see what has changed
  - Depending on whether the changed field is a Type-1, Type-2 or Type-3 change, do the necessary updates
  - Most ETL packages and the Internet include CRC computation and comparison code

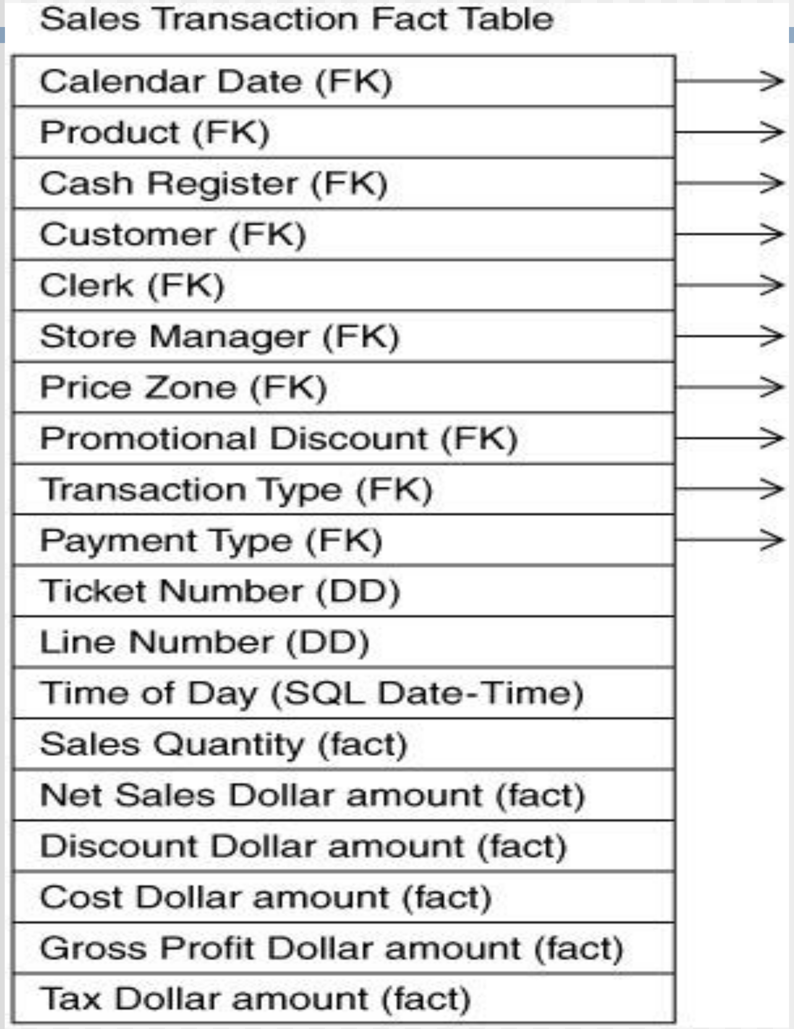
5. Delivering Fact Tables

Loading facts

- Facts

  Fact tables hold the measurements of an enterprise. The relationship between fact tables and measurements is extremely simple. If a measurement exists, it can be modeled as a fact table row. If a fact table row exists, it is a measurement

The basic structure of a fact table



Sales Transaction Fact Table

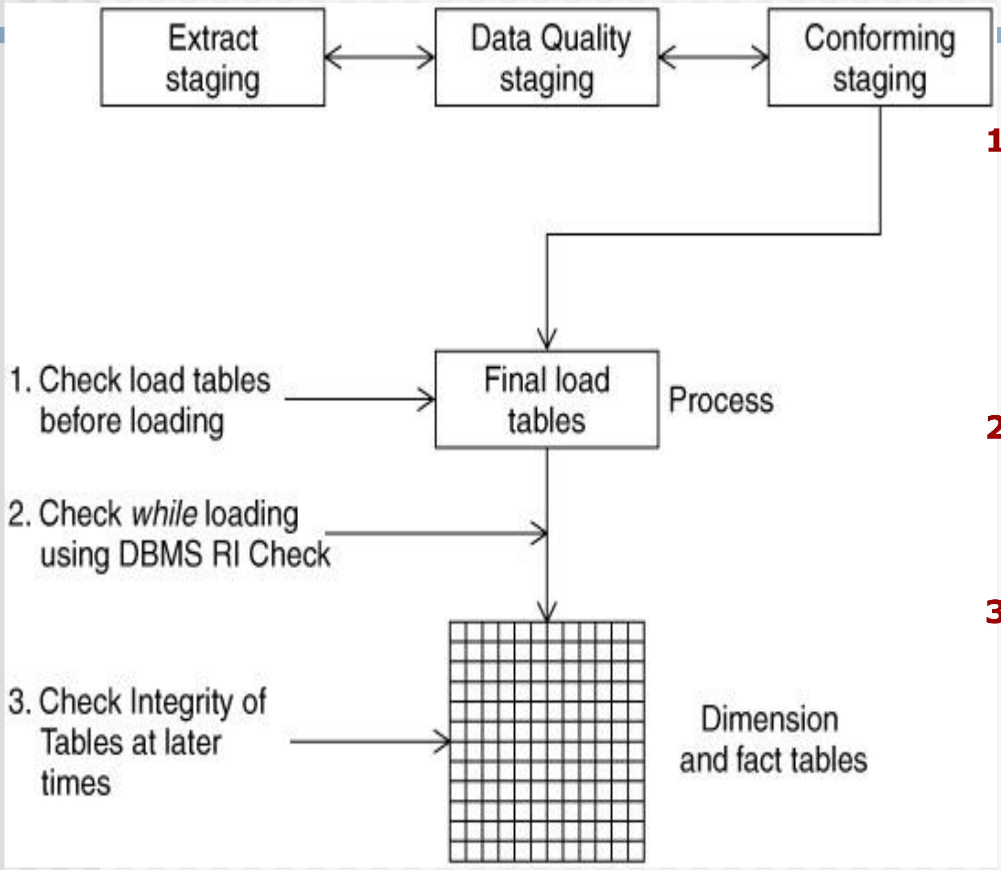
Calendar Date (FK)
Product (FK)
Cash Register (FK)
Customer (FK)
Clerk (FK)
Store Manager (FK)
Price Zone (FK)
Promotional Discount (FK)
Transaction Type (FK)
Payment Type (FK)
Ticket Number (DD)
Line Number (DD)
Time of Day (SQL Date-Time)
Sales Quantity (fact)
Net Sales Dollar amount (fact)
Discount Dollar amount (fact)
Cost Dollar amount (fact)
Gross Profit Dollar amount (fact)
Tax Dollar amount (fact)

- Every table defined by its grain
  - in business terms
  - in terms of the dimension foreign keys and other fields
- A set of foreign keys (FK)
  - context for the fact
  - Join to Dimension Tables
- Degenerate Dimensions
  - Part of the key
  - Not a foreign key to a Dimension table
- Primary Key
  - a subset of the FKs
  - must be defined in the table
- Surrogate keys
  - Has no business intelligence
  - Should not be delivered to the users

Key Building Process - Facts

- When building a fact table, the final ETL step is converting the natural keys in the new input records into the correct, contemporary surrogate keys
- ETL maintains a special surrogate key lookup table for each dimension. This table is updated whenever a new dimension entity is created and whenever a Type 2 change occurs on an existing dimension entity
- All of the required lookup tables should be pinned in memory so that they can be randomly accessed as each incoming fact record presents its natural keys. This is one of the reasons for making the lookup tables separate from the original data warehouse dimension tables.

Guaranteeing Referential Integrity



Extract staging
Data Quality staging
Conforming staging
Final load tables
Process
1. Check load tables before loading
2. Check *while* loading using DBMS RI Check
3. Check Integrity of Tables at later times
Dimension and fact tables

"EVERY FACT TABLE ROW MUST BE FILLED WITH LEGITIMATE FOREIGN KEYS"

1. **Check Before Loading**
   - Check before you add fact records
   - Check before you delete dimension records
   - Best approach
2. **Check While Loading**
   - Serious performance issue, while loading thousands of data.
3. **Check After Loading**
   - Theoretically ok, to find all violations
   - Running query over the entire fact table cause serious performance issues.
   - *If the checking is restricted to data loaded only today, we make an assumption that the date foreign key is reliable*

Options for loading the Surrogate Keys of Dimensions

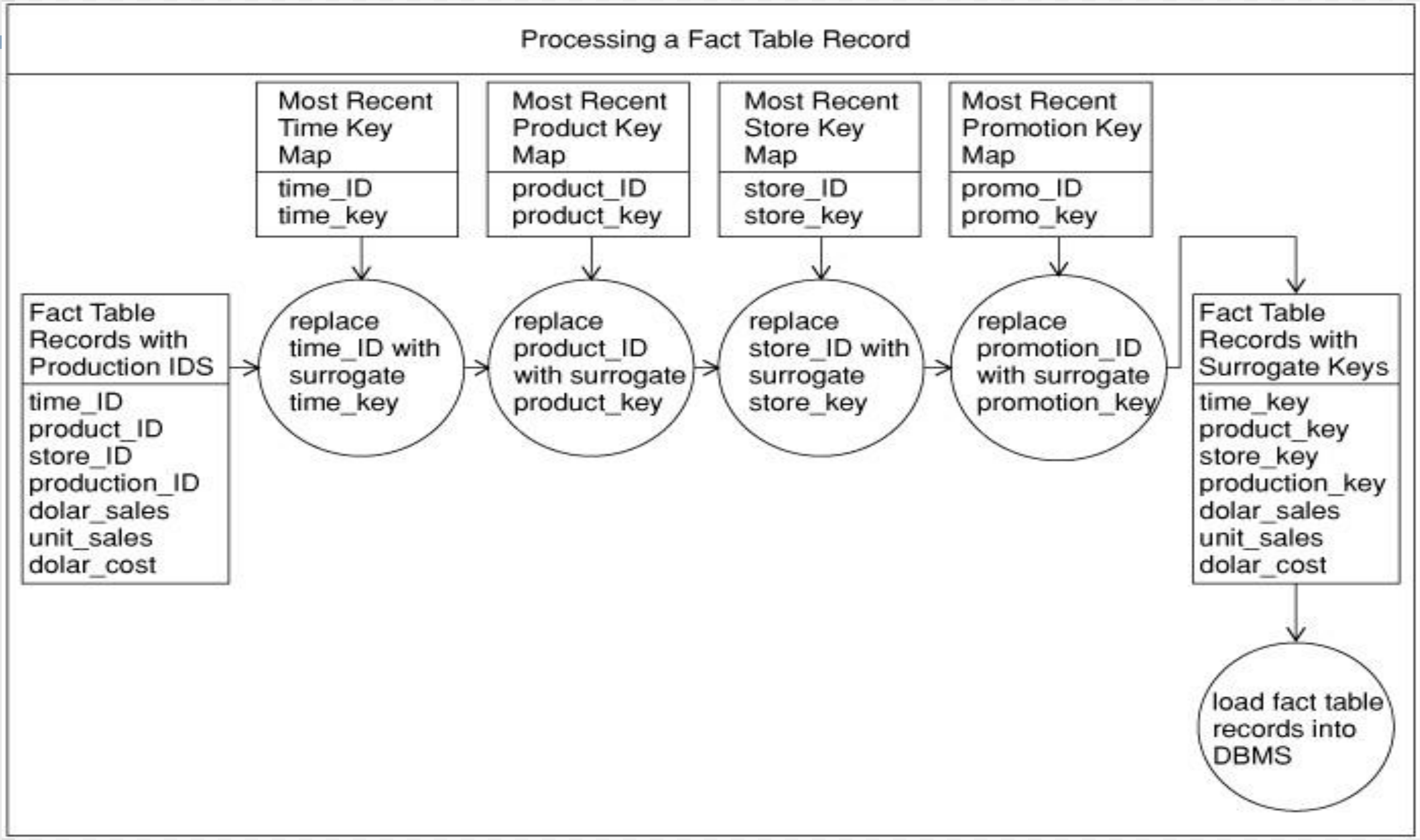
1. Look up the current surrogate key in each dimension, fetch the record with the most current surrogate key for the natural key and use that surrogate key. Good option but very slow.

2. Maintain a surrogate key lookup table for each dimension. This table is updated whenever a new record is added or when a Type-2 update occurs in an existing dimensional entity.
   - The dimensions must be updated with Type-2 updates before any facts are loaded into the Data Warehouse, to guarantee referential integrity

Surrogate Key Pipeline

- Assume that all records to be added to the fact table are current, i.e.,
  - in the incoming fact records, the value for the natural key of each dimension is the most current value known to the DW
- When loading a fact table, the final ETL step converts the natural keys of the new input records into the correct surrogate key of the dimensions using the key mapping tables

Surrogate Key Pipeline (contd...)


Processing a Fact Table Record
Most Recent Time Key Map
time_ID
time_key
Most Recent Product Key Map
product_ID
product_key
Most Recent Store Key Map
store_ID
store_key
Most Recent Promotion Key Map
promo_ID
promo_key
Fact Table Records with Production IDS
time_ID
product_ID
store_ID
production_ID
dolar_sales
unit_sales
dolar_cost
replace time_ID with surrogate time_key
replace product_ID with surrogate product_key
replace store_ID with surrogate store_key
replace promotion_ID with surrogate promotion_key
Fact Table Records with Surrogate Keys
time_key
product_key
store_key
production_key
dolar_sales
unit_sales
dolar_cost
load fact table records into DBMS

Grains of Fact Tables

- Every fact table should belong to exactly one of the following grains
  - Transaction grain
  - Periodic snapshot grain
  - Accumulating snapshot grain

Transaction Grain Fact Tables

- The grain represents an instantaneous measurement at a specific point in space and time
  - retail sales transaction
- More detailed fact table
- Generally consists of more number of dimensions
- Each record correspond to the instant of measurement.
- Transactions are time stamped to the nearest second/minutes
- Fact table records arrive in batches at regular intervals

Periodic Snapshot Fact Tables

- The grain represents a span of time periodically repeated
  - A periodic snapshot for a checking account in a bank, reported every month
    - Beginning balance
    - Ending balance
    - Number of deposits
    - Total for deposits
    - Number of withdrawals
    - Total for withdrawals
- Attributed by more number of facts
- Predictable sparseness
  - one checking account record per account per month

Accumulating Snapshot Fact Tables

- The grain represents finite processes that have a definite beginning and an end
  - Order fulfillment, claim processing, “small workflows”
  - But not large complicated looping workflows

- Example: shipment invoice line item
  - Order date
  - Requested ship date
  - Actual ship date
  - Delivery date
  - Last payment date
  - Return date

- Characteristics: large number of date dimensions, data are created and overwritten multiple times as events unfold

Loading a Table

- Separate inserts from updates
  - Identify update records from the bulk data to be loaded and process it first
  - Perform the insertion of the rest of the records using a bulk loader.
- Use a bulk loader
  - To improve performance of the inserts & decrease database overhead
  - With many bulk loaders, updating is not possible
- Load in parallel
  - ETL tool supports breaking data in logical segments, say one per year & load the data in parallel
  - Some ETL tools offers dynamic partition to help bulk loading
- Minimize physical updates
  - To decrease database overhead with writing the rollback logs, delete the records requires update & load it all afresh using bulk loader

Updating and Correcting Facts

- Negate the fact
  - Create an exact duplicate of the fact where all the measurements are negated (-minus), so the measures “cancel” each other in summaries
  - Do it for audit reasons and/or if capturing/measuring erroneous entries is significant to the business
- Update the fact
- Delete and reload the fact
  - Drawback: current versions of previously released reports no longer valid
  - Physical deletes → the record is deleted
  - Logical deletes → the record is tagged “deleted”

Graceful Modifications

- Modifications to the dimensional schema, without affecting end user applications

- Modifications includes
  - Adding a fact to the existing fact table at the same grain
  - Adding a dimension to the existing fact table at the same grain
  - Adding an attribute to an existing dimension
  - Increasing granularity of existing fact & dimension table
    - Complicated Job!
    - Changing weekly grain to daily grain ?
      - Constraints on weekly aggregation only work with the historical data.
      - Further additions can be at daily grain

Aggregate tables

- Aggregate navigator sits between user query & DBMS, intercepting queries and transforms base level queries into aggregate aware queries wherever possible

- Good aggregate program
  - Provides performance enhancement
  - Add only reasonable amount of storage
  - Transparent to the end user
    - Users are not aware of it.
    - No user program can refer to aggregation tables directly!
  - Affect the cost of extract system least possible
    - Each extract leads to the updation of aggregates!
  - Not adds much burden to the DBA
    - Generates metedata automatically

Aggregate tables- Design Requirements

1. Aggregates must be stored in their own fact tables, separate from base-level data.
2. Dimension tables attached to the aggregate tables must be shrunken version of the corresponding dimension

   The most shrunken version of the dimension is the dimension removed altogether
3. The base table & all its aggregate tables must be related together as a *family of schemas*, so that aggregate navigator knows which of the tables are related together
4. All queries from the users must be forced to refer the base table only.

Review Mid Sem

- Data Warehousing-Introduction
  - Definition
  - Characteristics
  - OLTP vs OLAP
  - Top-down vs. Bottom up approaches to Data Warehousing

- Data Warehousing Architecture
  - Architectural Components of a Data Warehouse
  - Data Marts
  - Data Staging
  - Meta Data

- Dimensional Modeling
  - Dimensional Modelling Vocabulary
  - ER Modelling vs. Dimensional Modelling
  - Retail Store Dimensional Model
  - Slowly Changing Dimensions, Surrogate keys
  - Factless Fact Tables