DWH

- businesses use data to
    1. keep business running
    2. understand current situation, answer business questions and make better decisions
- operational data - generated from business operations
    - need OLTP
        - mostly process one record per time
        - create, update, delete operations
        - historical context not required
- analytical data - used for querying and analysis
    - need OLAP
        - process many records per time
        - query operations with fast query performance
        - historical context needed
- DWH helps address analytical data needs i.e. it is a database used to store analytical data and is optimized for reporting and analysis
- ETL or data warehousing is process of bringing data stored in different sources (each source may have different data format) to a centralized storage
- ETL stands for Extract, Transform, Load
- DWH is created for business intelligence (BI)
- In BI, data is turned into meaningful insights to help make better decisions
- both Data lake (DL) and DWH are centralized storage but
    - DWH stores processed data, DL stores raw data
    - DWH stores structured data, DL can store unstructured data e.g. files, image, videos
    - data in DWH is ready to be used

  - data in DL might need processing before use
  - data in DL is used by data scientists, the data quality is not ensured
- ETL
  - Extract - data is extracted(copied) into staging area
  - Transform - transformations are done to data (if a lot of transformations need to be done, cleansing area might help to store data temporarily)
  - Load - transformed data is loaded to DWH (single point of truth)
- data marts have tables taken from DWH for specific use case which
  - increases user-friendliness (not overwhelming with large no. of tables)
  - helps with query performance because not everyone is querying from same storage
  - allows using different storage technologies, modeling techniques that are optimized for BI tools e.g. making query performance fast
- data mart is not always necessary
- why we need staging area
  - to not slow down sources which store operational data i.e. to not query from sources for long time (help with the performance)
  - to transform data
- temporary staging area is cleaned up after ETL for future use as opposed to persistent staging area
- types of extraction methods
  - full extraction - all the data currently available is extracted from the source, no need to keep track of changes to data since last successful extraction
  - incremental extraction - only data that has changed since a point in time will be extracted, some of the techniques are
    - change data capture (CDC) - changes i.e. insert, update, delete operations done to source are recorded and reflected in DWH near real-time
    - last modified timestamp - records which were updated later than most recently updated record from last execution are extracted

            - can use a control table to store the max last modified timestamp of the extracted records
        - partition source by last modified date - makes extraction faster with less rows to scan through
        - triggers - add triggers to source for insert, update, delete operations to reflect data changes in DWH
        - merge - extract all available data from source system, compare and merge the changes to previously extracted data (least preferable option & not suitable for large no. of records)
- relational database (RDB)
    - data is stored as tables(relations) consisting of rows and columns
    - each table has primary key (PK) which is unique & non null for each row
    - a table can optionally have foreign keys (FK) to reference PK of other tables
    - SQL is used to query data
- in-memory database
    - stores data in memory which give high query performance, suitable for analytics (usually used for data marts)
    - independent of how data is structured
    - data is not durable, durability is added by taking snapshots and saving them to disk
    - high cost
- OLAP cubes
    - non-relational (use arrays instead of tables), can help increase query performance
    - data in cube are pre-calculated (aggregated), reduce query time
    - should be built for specific use case, building as separate cubes makes them more efficient and less complex
    - optional after
        - star schema is built in RDB
        - hardware advancement

- Operational Data Storage (ODS)
    - storage of data integrated from different operational systems
    - can use ETL to integrate data from sources into ODS
    - data in ODS is used for operational decision making which
        - not need for long historical data
        - need very current & real-time data
- an organization might have
    - parallel ETLs
        - sources - ETL - DWH (for analytical decisions)
        - sources - ETL - ODS (for operational decisions)
    - sequential ETLs
        - sources - ETL - ODS - ETL - DWH (more common than parallel ETLs)
        - ODS is acting like staging layer for DWH
        - saves a lot of work compared to parallel ETLs
- ODS is not always necessary (nowadays sources are fast, source performance is not affected by data extraction)
- dimensional modeling - methods of organizing data (typically in DWH)
    - data is organized into fact tables and dimension tables
        - fact tables are in the middle and surrounded by dimension tables
            - fact table contains - PK, FKs & facts (aggregates, measurements)
            - dimension table contains - PK, FK (optional) & dimensions (context to fact)
            - FK in fact table is linked to PK of dimension table
            - facts (foundation of DWH) are aggregated and analyzed by dimensions
    - makes data retrieval faster for reporting, OLAP
    - usually the difference between fact and dimension are that
        - fact is aggregatable as opposed to dimension

        - fact is measurable but dimension is descriptive
        - often fact is event or transactional data
        - dimensions are usually used for filtering, grouping, labeling
- ways of dimensional modeling
    - start schema
        - dimension tables don't have FKs, not very normalized (denormalized, have data redundancy)
        - give better query performance because of denormalization (require only few join operations)
        - can have multiple fact tables that are usually not joined with each other
    - snowflake schema
        - dimensions can have FKs, more normalized than star schema (lower data redundancy, less storage used)
        - give better performance for insert, update, delete because of no repeated data (but require more join operations)
- star schema is more preferred in DWH and data marts
- often facts are numbers which can be additive
- there are 3 types of facts by their additivity
    - additive - can be added across all dimensions (most useful) e.g. order quantity, total price
    - semi-additive - can added across some dimensions e.g. balance (cannot be added across time but can be averaged)
    - non-additive - cannot be added across any dimension (limited analytical value) e.g. unit price (need to consider quantity too), percentage, ratio (store underlying value e.g. numerator and denominator separately instead)
- nulls in fact table
    - mostly null facts are ignored in aggregation
    - null facts can be replaced with meaningful values
    - should not have nulls in FK columns

        - if FK contains nulls, add new row to the dimension table for null category (dummy values)
        - fill the nulls in FK column with id of the null category row
- should aggregate to date values in BI tools and store underlying values in DWH
- types of fact tables
    - transactional fact table
        - a row include measurement of an event or a transaction
        - facts are typically additive, have a lot of dimensions associated
        - can be enormous in size and grow very fast
        - grain = transaction / event
    - periodic snapshot fact table
        - a row summarize measurement of many events or transactions over period of time e.g. 1 day, 1 week etc.
        - a row can optionally be summary over a combination of dimensions that includes period of time
        - tend to have a lot of aggregated measures, not so many dimensions
        - grain = shortest period
        - not so enormous in size, table growth is not rapid
    - accumulated snapshot fact table
        - a row include lifespan of a process e.g. order fulfillment, complaint resolution
        - usually used for tracking something in steps from beginning to end
        - can be used for workflow/process analysis
        - has many date dimensions associated with each row
    - faceless fact table
        - fact table that includes no facts (only PK and FKs)
        - existence of a row in fact table indicates that an event has occurred e.g. employee registration fact table, a row indicates a registration
- steps for creating a fact table

  - identify what we want to analyze e.g. sales, order processing
  - define the grain (level of detail) i.e. what is one row referring to e.g. transaction/event, aggregated value, business process
  - identify dimensions that are relevant
  - identify facts using the grain defined in 2.
- the higher level of detail (1 row = 1 unaggregated transaction/event) is preferred in fact tables for unlimitting the ways of analysis
- natural key - PK that come out from source system, can be alphanumeric value (string)
- surrogate key or artificial key
  - integer PK which was created during ETL
  - stored as additional column (foreign tables will reference the surrogate key)
  - can improve join performance if natural key is string
  - easier to administrate/update
  - help avoid duplicate PK values when integrating data from multiple source systems
- dimension table usually contain fewer rows than fact table
- date dimension table
  - contains date and the related dimensions e.g. year, month, quarter, date, day, flags(for weekend, holidays etc.)
  - has meaningful surrogate key - YYYYMMDD (integer)
  - can also include combination of features e.g. year-quarter, year-month
  - time is usually stored in different dimension table
- data from sources are often normalized (dimensions have hierarchy), but in DWH
  - snowflake schema or putting too many (irrelevant) FKs in fact table should be avoided
  - dimension tables should be flattened/denormalized to contain all relevant data e.g. product table - id, product name, product category, brand name etc.

- conformed dimension
  - is shared by multiple fact tables (PK of conformed dimension act as FK in multiple fact tables)
  - is used to compare (drill across) facts in different fact tables that can be of different grains (need to aggregate finer grained fact table)
  - e.g. date, time
- degenerate dimension
  - the only dimension in dimension table
  - is stored in fact table instead of dimension table
  - mostly found in transactional fact table e.g. invoice no., order no.
- junk dimension or transactional indicator dimension
  - are flags in fact table that does not fit into any dimension tables
  - can be stored in a dedicated dimension table (if there are multiple flags)
    - to save space, store only available combinations of flags in dimension table
    - use multiple junk dimension tables to reduce no. of rows in each
- role-playing dimension
  - is referenced by many columns in fact table
  - e.g. date dimension can be referenced in order fulfillment fact table as order date, shipping date, delivery date etc.
- dimensions might change, need strategy to handle each changing dimension
- strategy for handling changing dimension
  - Type 0 : retain original, when no need to make changes
  - Type 1 : overwrite, when changes need to be reflected (the history is lost and need to update the breaking queries)
  - Type 2 : new row, when changes need to be reflected and past records should not be affected
    - only surrogate key is not enough, natural key is needed here to find related past records

        - only natural key is not enough, surrogate key is needed here to differentiate between past and current records
        - can optionally add affective date and expiry date columns in dimension table to help look up current record
    - Type 3 : additional attribute, overwrite dimension but keep the previous version in separate column (not suitable for frequent changes)
- ETL tools has built-in tools for each phase of ETL
- with ETL tools we built workflows for each phase of ETL, the workflows are then scheduled as jobs running on specific time/frequency
- transient (temporary) staging layer, is the most common type of staging layer as opposed to permanent (persistent) staging layer
- ETL process
    - extract
        - first time (initial load) - use full extraction
            - ask business users - what data is needed
            - ask db admins - how data is structured in source system, when is good time to extract data e.g. nights, weekends
            - run small extractions to estimate time needed for full extraction
        - subsequent times (delta load) - use incremental extraction periodically (batch processing)
            - ask business users - how often delta load should be done to reflect changes from sources in DWH
                - settle conflicts e.g. business users want to delta load every 30 mins but ETL takes 1 hour
            - filter rows that are new or have been updated since last ETL and extract them
    - transform
        - design transformation steps
        - transform data extracted (reformat, restructure data into star schema)
        - basic transformation - deduplication, filtering out irrelevant rows, removing unwanted columns, converting column values to same format,

replacing nulls, surrogate key generation (can set the key column as auto increment in sql databases)

    - advanced transformation - joining to get surrogate PK values in FK column or to denormalize data, splitting string into columns, aggregating values, calculating derived values

  - load
    - initial load - insert all rows
    - delta load
      - insert new rows, update existing rows (if any updates)
      - if row was deleted in source system, don't delete the row in DWH (can mark the row as deleted from source using a flag column)

- types of ETL tool
  - enterprise (preferred)
    - commercial
    - have customer support
  - open-source
    - often free
    - lack of customer support
    - sometimes not enough documentation, hard to use
  - cloud-native
    - appropriate if data is already in cloud
    - sometimes need to work with data stored with other cloud provider (need multi-cloud flexibility)
  - custom
    - can be customized as tool users want
    - need to handle development, maintenance, training (need resources)
- choosing the right ETL tool
  - define requirements
  - evaluate alternatives

        - factors to consider
            - cost
            - data source connectors
            - capabilities
            - ease of use
            - reviews
            - support/extras
    - test or ask for demo/trial
    - make decision
- ELT
    - Extract - data is copied from source
    - Load
        - data is loaded immediately to DWH (no need to aggregate/transform data before loading)
        - allows data to be loaded into DWH near real-time by streaming
    - Transform
        - transformations are applied to data on the go (require very high performance DB)
        - data can be transformed in more than one way (flexible transformation)
- ELT is suitable with insensitive data because data is stored as it is in target DB
- ETL is suitable for reporting, ELT is suitable for data science/ML/big data
- DWH use case - reporting, data analysis, predictive analytics, big data
- in databases, normally data is not stored in systematic order and when querying e.g. filtering, entire table is scanned (full table scan) which is read-inefficient, to make read queries faster data can be stored in specific order using indexes
- indexing makes read queries faster but slows down the write queries (because index mapping also needs to be updated), indexing also requires additional storage to store mappings
- B-tree index

    - has multi-level tree structure
    - is default indexing method with an index already set on PK
    - should be used for columns with many unique values (high cardinality) e.g. surrogate keys, names
    - costly in terms of storage
- Bitmap index
    - data is stored in bits
    - good for large amount of data with low cardinality
    - storage efficient
- guidelines for indexes
    - should not create index on every column in table, create only for columns that are used for filtering often
    - indexing don’t help with read query performance when no. of rows in table is few
- types of DWH
    - on-premise DWH
        - DWH on own hardware (full control) but need to manage (require budget, workforce), not easy to scale down
    - cloud DWH
        - managed service, cost efficient (cheap), scalable, high availability
- massive parallel processing (MPP) - task is split into independent parallel sub-tasks, each with own computing resource
    - shared disk architecture - disk is shared among the parallel sub-tasks
    - shared nothing architecture - separated disk for each sub-task
- MPP helps when we have large amount of data or when multiple queries are run at the same time
- columnar storage
    - a column is stored in a block of data instead of a row
    - read efficient when querying all rows

- less storage space is used because each block stores only one type of data