DSA 610 Redesign, Lecture 5 Outline

Lecture Outline: Physical Database Model, Data Storage Methods, and Comparing Databases vs. Spreadsheets
Duration: 50 minutes

---

1. Introduction to the Physical Database Model (10 minutes)

- **Objective:** Provide an understanding of the physical model in database design and its importance.
- **Content:**
  - **Overview of Database Models:**
    - **Conceptual Model:** High-level, abstract design (ER diagrams).
    - **Logical Model:** Detailed schema design, including tables, relationships, and keys.
    - **Physical Model:** The implementation level, detailing how data is stored on physical storage devices.
  - **Physical Model Definition:**
    - **Definition:** The physical model focuses on how data is stored, accessed, and managed on hardware.
    - **Components:** Includes storage formats, indexing, file structures, and data access paths.
  - **Importance of the Physical Model:**
    - **Performance:** Influences query speed, data retrieval, and overall system efficiency.
    - **Storage Efficiency:** Determines how well storage resources are utilized.
    - **Data Integrity:** Ensures reliable data storage and retrieval.
  - **Example:** Discuss a physical model for a relational database, showing how tables might be physically stored on disk with row-oriented storage.

---

2. Data Storage Methods (20 minutes)

- **Objective:** Introduce different data storage methods, types, and formats used in databases.
- **Content:**
  - **Storage Types:**
    - **File Storage:** Data stored in files on a disk.
      - **Flat Files:** Plain text or CSV files, often used for simple data storage.
      - **Binary Files:** More efficient, but less human-readable.
    - **Block Storage:**
      - **Definition:** Data is stored in fixed-size blocks on disk, typical for relational databases.
      - **Use Cases:** Used in transactional systems where quick read/write access is needed.
    - **Object Storage:**
      - **Definition:** Data stored as objects, typically used in NoSQL and cloud storage systems.
      - **Use Cases:** Ideal for storing unstructured data like images, videos, and documents.
    - **Columnar Storage:**
      - **Definition:** Data is stored by columns rather than rows.

        - **Use Cases:** Used in data warehousing and analytics, as it optimizes read operations on specific columns.
        - **Example:** Comparing how a columnar database (like Apache Parquet) differs from a row-oriented database (like MySQL).
  - **Data Formats:**
    - **Text-based Formats:**
      - **CSV:** Common for data exchange, but not efficient for large datasets.
      - **JSON and XML:** Structured text formats often used for web data exchange.
    - **Binary Formats:**
      - **Avro and Parquet:** Efficient for storage and transmission, with built-in schema definitions.
      - **BLOBs (Binary Large Objects):** Used to store large files, such as images and videos, in databases.
    - **Specialized Formats:**
      - **HDF5:** Used for storing large amounts of scientific data.
      - **SQL Databases:** Proprietary binary formats optimized for database operations.
    - **Example:** Show how data might be stored in different formats, like a CSV vs. a JSON vs. a Parquet file, and discuss the advantages and disadvantages of each.
  - **Data Access and Retrieval:**
    - **Indexes:**
      - **Definition:** Data structures that improve the speed of data retrieval.
      - **Types:** B-trees, hash indexes, and full-text indexes.
      - **Trade-offs:** Discuss the balance between faster reads and slower writes/updates due to indexing.
    - **Compression:**
      - **Purpose:** Reducing storage space and speeding up I/O operations.
      - **Techniques:** Lossless compression methods (e.g., gzip, zlib) and their impact on performance.
    - **Partitioning:**
      - **Definition:** Dividing a database into smaller, more manageable pieces.
      - **Types:** Horizontal (row-based) and vertical (column-based) partitioning.
      - **Example:** Show how partitioning can improve query performance in large databases.

---

3. Comparison of Databases vs. Spreadsheets (15 minutes)

- **Objective:** Compare and contrast databases and spreadsheets, highlighting their respective strengths and weaknesses.
- **Content:**
  - **Overview of Spreadsheets:**
    - **Definition:** A software application for data storage, organization, and analysis, typically using tables (e.g., Excel, Google Sheets).
    - **Use Cases:** Suitable for small datasets, personal finance, and simple data analysis tasks.
  - **Comparison with Databases:**
    - **Scalability:**

        - **Spreadsheets:** Limited by file size and performance issues with large datasets.
        - **Databases:** Designed to handle large datasets and concurrent access by multiple users.
    - **Data Integrity:**
        - **Spreadsheets:** Prone to errors, data corruption, and manual entry mistakes.
        - **Databases:** Enforce data integrity through constraints, transactions, and ACID properties.
    - **Data Relationships:**
        - **Spreadsheets:** Can handle simple relationships, but complex relationships are difficult to manage.
        - **Databases:** Built for managing complex relationships between data entities (e.g., foreign keys, joins).
    - **Collaboration and Multi-user Access:**
        - **Spreadsheets:** Limited multi-user collaboration, with potential for conflicts.
        - **Databases:** Support for concurrent access, with sophisticated mechanisms for handling conflicts and ensuring consistency.
    - **Data Analysis and Querying:**
        - **Spreadsheets:** Basic filtering, sorting, and formulas for analysis.
        - **Databases:** Powerful querying capabilities with SQL, allowing complex data analysis and aggregation.
    - **Automation and Integration:**
        - **Spreadsheets:** Limited automation through macros and integrations with other tools.
        - **Databases:** Extensive automation possibilities with stored procedures, triggers, and integration with other systems through APIs.
    - **Security:**
        - **Spreadsheets:** Basic password protection, but not suitable for sensitive data.
        - **Databases:** Robust security features including encryption, user roles, and access controls.
    - **Example:** Show a scenario where a spreadsheet might be sufficient (e.g., tracking personal expenses) and where a database is necessary (e.g., managing customer data for an e-commerce platform).

---

4. Practical Demonstration: Database vs. Spreadsheet (5 minutes)

- **Objective:** Provide a brief demonstration to reinforce the comparison between databases and spreadsheets.
- **Content:**
    - **Demonstration:** Show how a simple task, like aggregating data or filtering records, can be done in both a spreadsheet and a database.
    - **Discussion:** Highlight the ease or complexity, speed, and flexibility of each tool in handling the task.

---

5. Conclusion & Q&A (5 minutes)

- **Objective:** Summarize key concepts and address any remaining questions.

- **Content:**
    - Recap the importance of understanding the physical database model and its impact on performance and storage.
    - Highlight the differences between databases and spreadsheets, helping students choose the right tool for their needs.
    - Open the floor for questions and final discussions.

Key Takeaways

- The physical model of a database is crucial for optimizing data storage and access, directly impacting system performance.
- Different data storage methods and formats serve various needs, from simple text files to complex binary formats optimized for speed and efficiency.
- Databases and spreadsheets have distinct strengths and use cases; understanding these differences is key to making informed decisions in data management and analysis.

Resources:
PostgreSQL: https://www.postgresql.org/
MySQL: https://www.mysql.com/
Oracle Database: https://www.oracle.com/database/
Microsoft SQL Server: https://www.microsoft.com/en-us/sql-server
MongoDB: https://www.mongodb.com/
Apache Cassandra: https://cassandra.apache.org/_/index.html
Redis: https://redis.io/
Neo4j: https://neo4j.com/
Amazon Neptune: https://aws.amazon.com/neptune/
Amazon Redshift: https://aws.amazon.com/redshift/
Big Query: https://cloud.google.com/bigquery

Lecture Outline: Coded Values, Dealing with Missing Data, and Dealing with Outliers
Duration: 50 minutes

---

1. Introduction to Coded Values (10 minutes)

- **Objective:** Understand the concept of coded values in data analysis, why they are used, and how to work with them.
- **Content:**
    - **Definition and Purpose:**
        - **Coded Values:** Representation of categorical or qualitative data in numerical form.
        - **Purpose:** Simplify data entry, storage, and analysis by converting categories into numeric codes.
    - **Examples of Coded Values:**
        - **Binary Coding:** 0 for "No", 1 for "Yes".
        - **Ordinal Coding:** Ranking categories such as 1 for "Low", 2 for "Medium", 3 for "High".
        - **Nominal Coding:** Assigning arbitrary numbers to categories like 1 for "Apple", 2 for "Banana", 3 for "Cherry".
    - **Encoding Techniques:**
        - **One-Hot Encoding:** Create binary variables for each category in the dataset.

        - **Label Encoding:** Assign a unique integer to each category.
    - **Considerations:**
        - **Interpretation:** Importance of understanding the meaning behind the codes.
        - **Data Analysis Implications:** How coded values impact analysis, particularly in regression models where ordinal vs. nominal data must be treated differently.
    - **Example:** Demonstrating the use of coded values in a dataset, such as encoding gender as 0 and 1.
- **Special Codes in Data Analysis**
    - **Outliers**: Special codes can be assigned to outliers to indicate data points that fall outside the expected range. For example, in census data, outliers might be coded as -999 or 999999 to flag them for further review.
    - **Missing Values**: Missing values are often coded with specific numbers to indicate that data is not available. Common codes include -1, 99, or 999. These codes help analysts identify and handle missing data appropriately.
    - **Invalid Entries**: Codes can also be used to mark invalid or erroneous entries. For example, a code like 9999 might be used to indicate a data entry error that needs correction.
    - **Suppressed Data**: In some cases, data might be intentionally suppressed for privacy reasons. Special codes like 999 or *** can be used to indicate that the data is not disclosed.
- **Why Use Special Codes?**
    - **Data Integrity**: Helps maintain the integrity of the data by clearly marking anomalies.
    - **Error Detection**: Facilitates the detection of errors and anomalies in the data set.
    - **Consistency**: Ensures consistency in how missing or anomalous data is handled across different datasets.
- **Identifying Special Codes in Data Analysis**
    - **Data Dictionary**: A well-documented data dictionary often includes explanations of all special codes used within the dataset. Analysts should always refer to this document first.
    - **Summary Statistics**: Analysts can calculate summary statistics to identify unusual values that may represent special codes. For example, extremely high or low values might indicate outliers or missing values.
    - **Frequency Distribution**: Generating a frequency distribution of each variable can help identify unexpected values that occur more frequently, suggesting they might be special codes.
    - **Domain Knowledge**: Having domain knowledge or understanding the context of the data can help analysts recognize values that don't fit within the expected range or patterns, indicating possible special codes.
    - **Consistency Checks**: Performing consistency checks across related variables can reveal special codes. For instance, if age is recorded as -999 for missing values, other demographic information for that record might also have similar codes.
    - **Descriptive Analysis**: A thorough descriptive analysis, including visualizations like histograms or box plots, can help identify anomalies or special codes.

By combining these techniques, analysts can systematically identify and handle special codes, even when they are not explicitly marked.

---

2. Dealing with Missing Data (20 minutes)

- **Objective:** Learn methods for identifying, understanding, and handling missing data in datasets.
- **Content:**
    - **Types of Missing Data:**
        - **MCAR (Missing Completely at Random):** Data is missing independently of any observed or unobserved data.
        - **MAR (Missing at Random):** Missingness depends on observed data but not on the missing data itself.
        - **MNAR (Missing Not at Random):** Missingness depends on the unobserved data, making it more complex to handle.
    - **Identifying Missing Data:**
        - **Using Python Libraries:** Tools like pandas to check for missing data (isnull(), notnull(), missingno library for visualization).
        - **Example:** Visualizing missing data patterns in a dataset using missingno.
    - **Strategies for Handling Missing Data:**
        - **Deletion:**
            - **Listwise Deletion:** Removing entire rows with missing data.
            - **Pairwise Deletion:** Only removing the missing values in the context of a specific analysis.
            - **Considerations:** Impact on sample size and potential bias.
        - **Imputation:**
            - **Mean/Median/Mode Imputation:** Replacing missing values with the mean, median, or mode of the column.
            - **Regression Imputation:** Predicting missing values using a regression model based on other variables.
            - **Multiple Imputation:** Using a probabilistic model to generate multiple imputations and averaging results.
            - **Example:** Imputing missing values in a dataset using pandas or scikit-learn.
        - **Advanced Techniques:**
            - **K-Nearest Neighbors (KNN) Imputation:** Filling missing values by averaging the nearest neighbors' values.
            - **Machine Learning Models:** Using algorithms like Random Forests to predict and impute missing values.
        - **Considerations and Trade-offs:**
            - **Bias Introduction:** Potential bias when imputing data.
            - **Data Integrity:** Balancing the need for complete data with maintaining data integrity.
    - **Example:** Demonstrating imputation techniques on a dataset with missing values.

---

3. Dealing with Outliers (15 minutes)

- **Objective:** Understand how to identify and handle outliers in datasets to improve data quality and analysis accuracy.
- **Content:**
    - **Definition and Impact:**
        - **Outliers:** Data points that significantly deviate from the rest of the data.
        - **Impact on Analysis:** Outliers can distort statistical measures like mean, variance, and regressions, leading to inaccurate results.
    - **Identifying Outliers:**

        - **Visualization Techniques:**
            - **Box Plots:** Visualize data spread and identify potential outliers.
            - **Scatter Plots:** Identify outliers in bivariate data.
            - **Histogram/Distribution Plots:** Detect outliers in the distribution of a single variable.
        - **Statistical Methods:**
            - **Z-Scores:** Data points with a Z-score above a certain threshold (e.g., 3) are considered outliers.
            - **IQR (Interquartile Range):** Data points outside 1.5 times the IQR above the third quartile or below the first quartile are considered outliers.
        - **Example:** Visualizing and identifying outliers in a dataset using Python (matplotlib, seaborn).
    - **Strategies for Handling Outliers:**
        - **Exclusion:**
            - **Removing Outliers:** When outliers are errors or are irrelevant to the analysis.
            - **Considerations:** Potential loss of important data and the need for justification.
        - **Transformation:**
            - **Log Transformation:** Reducing the effect of outliers by transforming the data.
            - **Winsorizing:** Replacing extreme values with the nearest acceptable value.
        - **Capping/Flooring:** Setting upper and lower limits to keep outliers within a reasonable range.
        - **Model-Based Approaches:**
            - **Robust Regression:** Using models less sensitive to outliers (e.g., Huber Regression).
            - **Machine Learning Models:** Algorithms like Decision Trees are less affected by outliers.
        - **Example:** Applying different methods to handle outliers in a dataset and comparing results.
    - **Considerations:**
        - **When to Keep Outliers:** Understanding when outliers represent true variation in the data.
        - **Impact on Analysis:** Assessing how outlier handling methods affect the results and conclusions of data analysis.

---

4. Practical Implementation & Q&A (5 minutes)

- **Objective:** Apply the learned concepts through a hands-on example and address any questions.
- **Content:**
    - **Example:** A brief walkthrough where students identify and handle missing data and outliers in a sample dataset using Python.
    - **Q&A:** Open the floor for questions, encouraging students to discuss challenges they've faced with missing data or outliers in their own work.

Key Takeaways

- **Coded Values:** Understanding how and why categorical data is coded and the implications for data analysis.
- **Missing Data:** Familiarity with various types of missing data and effective strategies to handle them, ensuring data quality.
- **Outliers:** Identifying outliers and using appropriate methods to address them, balancing data integrity with accurate analysis.

Resources:
The Prevention and handling of missing data: https://pmc.ncbi.nlm.nih.gov/articles/PMC3668100/
How to deal with missing data: https://www.mastersindatascience.org/learning/how-to-deal-with-missing-data/
Strategies for dealing with missing data: https://www.dasca.org/world-of-data-science/article/strategies-for-handling-missing-values-in-data-analysis
Working with missing data in Pandas: https://www.geeksforgeeks.org/working-with-missing-data-in-pandas/
Imputation of missing values: https://scikit-learn.org/stable/modules/impute.html
Missing value imputation in Python: https://medium.com/@hassankhan2608/missing-value-imputation-methods-using-python-f1b8796901ba
Dealing with Outliers: https://thedocs.worldbank.org/en/doc/20f02031de132cc3d76b91b5ed8737d0-0050012017/related/lecture-12-1.pdf
How to find and deal with outliers in your data: https://cxl.com/blog/outliers/
How to handle outliers in Python? https://www.projectpro.io/recipes/deal-with-outliers-in-python
Detecting and Treating Outliers: https://www.analyticsvidhya.com/blog/2021/05/detecting-and-treating-outliers-treating-the-odd-one-out/
Census code lists: https://www.census.gov/programs-surveys/acs/technical-documentation/code-lists.html, https://www.census.gov/programs-surveys/acs/technical-documentation.html

Lecture Outline: Time Series, Datetimes, and Basic Data Cleaning in Python
Duration: 50 minutes

---

1. Introduction to Time Series Data (10 minutes)

- **Objective:** Understand what time series data is, its key characteristics, and common use cases.
- **Content:**
    - **Definition:**
        - **Time Series Data:** Data points indexed in time order, typically consisting of sequences of observations at specific time intervals.
    - **Characteristics:**
        - **Temporal Dependency:** Observations depend on previous time points.
        - **Trend:** Long-term increase or decrease in the data.
        - **Seasonality:** Repeated patterns at regular intervals (e.g., daily, monthly).
        - **Cyclicity:** Long-term cycles not fixed in time.
        - **Stationarity:** Statistical properties (mean, variance) are constant over time.
    - **Examples of Time Series:**
        - **Finance:** Stock prices, exchange rates.
        - **Economics:** GDP, inflation rates.
        - **Weather:** Temperature, rainfall over time.
        - **Health:** Daily step counts, heart rate monitoring.
    - **Use Cases:**

        - **Forecasting:** Predicting future values based on past data.
        - **Anomaly Detection:** Identifying outliers or unusual patterns.
        - **Signal Processing:** Filtering and analyzing time-dependent signals.

---

2. Working with Datetimes in Python (15 minutes)

- **Objective:** Learn how to handle and manipulate datetime data in Python using the datetime module and pandas.
- **Content:**
    - **Python datetime Module:**
        - **Basic Operations:**
            - **Creating Datetime Objects:** datetime.datetime(), datetime.date(), datetime.time().
            - **Formatting and Parsing:** Converting strings to datetime and vice versa using strptime() and strftime().
            - **Arithmetic Operations:** Adding and subtracting dates/times (timedelta objects).
        - **Example:** Converting a list of date strings into datetime objects and performing arithmetic operations.
    - **Working with Datetimes in pandas:**
        - **Datetime Index:**
            - **Setting Datetime as Index:** Using pd.to_datetime() to convert and set a datetime index for time series data.
            - **Example:** Loading a CSV file with date information and setting the datetime column as the index.
        - **Datetime Operations:**
            - **Resampling:** Aggregating data by different time intervals (e.g., daily to monthly).
            - **Shifting:** Moving data points forward or backward in time (e.g., creating lag features).
            - **Handling Missing Dates:** Filling in missing dates and times using resample().ffill() or bfill().
        - **Time Zone Handling:**
            - **Localizing Time Series:** Setting a time zone using tz_localize().
            - **Converting Time Zones:** Using tz_convert() to change time zones.
        - **Example:** Time zone conversion and resampling in a time series dataset using pandas.

---

3. Basic Data Cleaning Procedures in Python (20 minutes)

- **Objective:** Equip students with basic data cleaning techniques for time series and datetime data in Python.
- **Content:**
    - **Loading and Inspecting Data:**
        - **Reading Data:** Loading data from CSV, Excel, or other sources using pandas.
        - **Initial Inspection:** Checking for missing values, data types, and summary statistics using .info(), .describe().
        - **Example:** Load a dataset and perform initial data inspection to identify potential issues.
    - **Handling Missing Data in Time Series:**

        - **Detecting Missing Values:**
            - **Identifying Gaps:** Visualizing and checking for missing dates in a time series.
        - **Filling Missing Data:**
            - **Forward/Backward Filling:** Filling missing values with previous/next valid data points.
            - **Interpolation:** Estimating missing values using linear or polynomial interpolation.
            - **Example:** Filling missing dates and interpolating missing values in a time series dataset.
    - **Removing Duplicates and Inconsistent Data:**
        - **Identifying Duplicates:** Using .duplicated() and .drop_duplicates() in pandas.
        - **Handling Inconsistent Data:**
            - **Correcting Errors:** Fixing typos or inconsistent data formats (e.g., inconsistent date formats).
            - **Example:** Removing duplicates and correcting inconsistent date formats in a dataset.
    - **Outlier Detection and Handling:**
        - **Detecting Outliers:**
            - **Visual Inspection:** Using plots like box plots and time series plots to identify outliers.
            - **Statistical Methods:** Using Z-scores or IQR to identify outliers.
        - **Handling Outliers:**
            - **Removal:** Dropping outliers if they are errors or irrelevant.
            - **Transformation:** Applying transformations to reduce the impact of outliers (e.g., log transformation).
            - **Example:** Detecting and handling outliers in a time series dataset using pandas and visualization libraries.
    - **Standardizing and Scaling Time Series Data:**
        - **Importance of Scaling:**
            - **Consistency:** Ensuring that all features are on the same scale for analysis.
            - **Scaling Techniques:** Using StandardScaler or MinMaxScaler from scikit-learn.
            - **Example:** Standardizing a time series dataset using scikit-learn.

---

4. Practical Implementation & Q&A (5 minutes)

- **Objective:** Apply the learned concepts through a hands-on example and address any questions.
- **Content:**
    - **Example:** A brief walkthrough where students load a time series dataset, perform datetime operations, and clean the data using the techniques discussed.
    - **Q&A:** Open the floor for questions, encouraging students to discuss challenges they've faced with time series data in their own work.

Key Takeaways

- **Time Series:** Understanding the unique characteristics of time series data and its applications in analysis.

- **Datetimes in Python:** Gaining proficiency in handling and manipulating datetime data using Python's datetime module and pandas.
- **Data Cleaning:** Learning essential data cleaning procedures for time series, including handling missing data, outliers, and inconsistent data.

Resources:
Time Series Analysis in Python: https://builtin.com/data-science/time-series-python, https://www.geeksforgeeks.org/time-series-data-visualization-in-python/
Line Graphs: https://www.datacamp.com/tutorial/matplotlib-time-series-line-plot
Time Series Forecasting: https://builtin.com/data-science/time-series-forecasting-python
Differencing: https://machinelearningmastery.com/difference-time-series-dataset-python/
ACF, PACF: https://www.kaggle.com/code/iamleonie/time-series-interpreting-acf-and-pacf
Datetimes: https://realpython.com/python-datetime/
Outliers with Python: https://medium.com/@saraswatp/exploring-data-anomalies-rejecting-outliers-with-python-660b1ed6bca6
Missing Data in Python: https://www.kaggle.com/code/parulpandey/a-guide-to-handling-missing-values-in-python
Rescaling Data: https://codefellows.github.io/sea-python-401d5/lectures/rescaling_data.html