Final Review: DSC 208 Data Management for Analytics

Review Questions

SQL Review Questions

Question 1

What is a logical query plan? How is it different from a physical query plan?

Answer:

A **Logical Query Plan (LQP)** is a Directed Acyclic Graph (DAG) whose vertices represent ”Logical Operators” from Extended Relational Algebra. It describes *what* is computed in a query.

A **Physical Query Plan (PQP)** is also a DAG whose vertices are ”Physical Operators.” It specifies the exact algorithm or code to run for each logical operation, including all parameters, thus detailing *how* the query is computed. A single LQP can have many possible PQPs. The key difference is that LQP focuses on the ”what,” while PQP focuses on the ”how” of query execution.

Question 2

Briefly explain 2 benefits of declarativity in data systems.

Answer:

Declarativity, often referred to as logical-physical separation or data independence, offers significant benefits in data systems:

1. **Increased User Productivity**: Users can focus on what data they want (the logical query) without needing to specify the exact execution steps, simplifying query writing.
2. **Automated Optimization and Scalability**: The RDBMS can automatically optimize the query code, leading to faster and more scalable performance without manual tuning by the user. Another benefit is **Application Portability**, meaning that changes to the internal system (physical implementation) do not require changes to the application code.

Question 3

Which index structure is useful only for equality predicates?

Answer:

A **Hash Index** structure is generally efficient and useful primarily for equality predicates (‘=‘). It is not efficient for range queries (e.g., ‘¿‘, ‘¡‘, ‘BETWEEN‘) or ‘LIKE‘ pattern matching with wildcards at the beginning.

Question 4

Suppose you have two predicates, P1 with selectivity 20% and P2 with selectivity 2%. Which is said to be more selective?

Answer:

P2 with 2% selectivity is said to be more selective. Selectivity refers to the fraction of rows that satisfy a predicate. A lower percentage indicates that the predicate filters out a larger proportion of data, leaving a smaller, more specific result set. More selective predicates help reduce the amount of data that needs to be processed, which is beneficial for query performance.

Question 5

Are all composite indexes unique indexes? Are all unique indexes primary indexes?

Answer:

- **Are all composite indexes unique indexes? No.** A composite index is an index on multiple columns. It does not inherently guarantee uniqueness across those columns unless explicitly defined as a 'UNIQUE' index.
- **Are all unique indexes primary indexes? No.** A 'UNIQUE' index ensures that all values in the indexed column(s) are distinct. While a 'PRIMARY KEY' constraint inherently creates a unique index, not all 'UNIQUE' indexes are primary keys. A table can have multiple 'UNIQUE' constraints, but only one 'PRIMARY KEY'. The primary key also cannot contain 'NULL' values, whereas a unique index might allow one 'NULL' value (depending on the DBMS).

Question 6

Given a relation with 3 attributes, how many different B+ tree indexes can be built on this relation?

Answer:

Given a relation with 3 attributes (let's call them A, B, C), the number of different B+ tree indexes that can be built is calculated by considering all possible subsets of attributes and their permutations (since the order of attributes matters in composite B+ tree indexes).

- **Single-attribute indexes (1 attribute):** (A), (B), (C) = 3 options
- **Two-attribute composite indexes (2 attributes, order matters):** (A, B), (B, A), (A, C), (C, A), (B, C), (C, B) = 6 options
- **Three-attribute composite indexes (3 attributes, order matters):** (A, B, C), (A, C, B), (B, A, C), (B, C, A), (C, A, B), (C, B, A) = 6 options

Total number of different B+ tree indexes = $3 + 6 + 6 = \mathbf{15}$ different B+ tree indexes.

Question 7

What are the 2 main parts of a typical RDBMS query optimizer?

Answer:

The two main parts of a typical RDBMS query optimizer are:

1. **Plan Enumerator**: This component explores and generates alternative Physical Query Plans (PQPs) for a given Logical Query Plan (LQP). It does this by applying algebraic rewrite rules to transform LQPs and by selecting different physical operators for logical operations.
2. **Plan Cost Estimator**: This component calculates an analytically determined runtime cost for each generated PQP. This estimation helps the optimizer choose the most efficient plan.

Question 8

Why is a selection pushdown rewrite often faster?

Answer:

A selection pushdown rewrite is often faster because it performs the filtering (selection) operation *before* other more expensive operations, such as joins. By applying the selection predicate as early as possible (pushing it down the query plan tree), it significantly reduces the size of the intermediate data sets that need to be processed by subsequent operations (like joins). Smaller intermediate data sets lead to fewer I/O operations and less CPU processing, resulting in faster query execution.

Data and Task Parallelism Review Questions

Question 1

Briefly explain two advantages of large-scale data for analytics.

Answer: Large-scale data offers several advantages for analytics:

1. It allows for the study of granular phenomena in sciences and businesses that were previously impossible to observe or analyze.
2. It enables new applications and facilitates personalization or customization, such as tailored recommendations in e-commerce.
3. It supports more complex Machine Learning (ML) prediction targets and helps mitigate variance to achieve high accuracy.

Question 2

What are the "3 Vs" of so-called "Big Data"?

Answer: The "3 Vs" of "Big Data" are:

- **Volume**: Data that is larger than what can be processed or stored on a single node's DRAM.
- **Variety**: Data that comes in diverse forms, including relational data, documents, social media posts, and multimedia.
- **Velocity**: Data generated at a high rate, such as from sensors or surveillance systems.

Question 3

Why is "Big Data" a thing nowadays even though Petabyte-scale databases have been around since the 1990s?

Answer: "Big Data" has gained prominence now primarily due to two factors beyond just volume:

1. **Applications**: A new "data-driven mentality" has emerged across nearly all human endeavors, from web services like e-commerce and social media to scientific research, medicine, logistics, finance, and humanities. This widespread adoption drives the need to process and analyze massive, varied datasets.
2. **Storage and Compute Advancements**: While petabyte-scale databases existed, modern hardware advancements in storage capacity (HDDs, SSDs, NVM-NAND), compute capacity (multi-core CPUs, GPUs, TPUs), and DRAM capacity have made it feasible and cost-effective to process and manage these large datasets. Cloud computing has also democratized access to this powerful infrastructure.

Question 4

What is a dataflow graph? Name two data processing tools that deal with dataflow graphs:

Answer: A **dataflow graph** (also known as a Logical Query Plan in database systems or a Neural Computational Graph in ML systems) is a Directed Acyclic Graph (DAG) where vertices represent operators (e.g., relational algebra operators or tensor algebra operations), and edges represent the flow of data between these operators.

Two data processing tools that deal with dataflow graphs are:

1. **Relational Database Management Systems (RDBMSs)**: They internally convert SQL queries into Logical Query Plans, which are a form of dataflow graphs.
2. **Apache Spark**: While not explicitly named in the context of "dataflow graph" definition, Spark's RDDs and DataFrames often build execution plans that are essentially dataflow graphs. (Additionally, tools like **Dask** also manage computations as dataflow/task graphs).

Question 5

What is the basic difference between task-parallel systems and data-parallel systems?

Answer: The basic difference lies in how the workload is distributed and processed:

- **Task-parallel systems** (or functional parallelism) focus on dividing a larger problem into multiple independent tasks, where each task performs a different operation. These tasks can then be executed concurrently, often on different data or different parts of a larger computation. An example is a pipeline of different operations.
- **Data-parallel systems** (or data-level parallelism) involve applying the same operation or program to different partitions of a large dataset simultaneously. The data is sharded across multiple workers, and each worker performs the same computation on its local data subset.

  Essentially, task parallelism means "different things to different data," while data parallelism means "same thing to different data."

Question 6

How is data-parallelism different from task parallelism?

Answer: Data parallelism and task parallelism represent two distinct approaches to parallelizing computations:

- **Data Parallelism**: This involves partitioning a large dataset across multiple processors or nodes, with each processor executing the *same operation* or program on its own subset of the data simultaneously. The focus is on distributing the data. Examples include MapReduce and parallel RDBMS queries.
- **Task Parallelism**: This involves decomposing a problem into multiple distinct, independent tasks or functions. Each task can perform a *different operation*, and these tasks are then executed concurrently on different processors. The focus is on distributing the work (tasks). An example would be a processing pipeline where different stages run in parallel.

Question 7

What are the three main paradigms of multi-node parallelism in cluster computing?

Answer: The three main paradigms of multi-node parallelism in cluster computing are:

1. **Shared-Nothing Parallelism**: Each CPU (worker) has its own dedicated memory and disk, with communication occurring explicitly over a network. Data is typically partitioned (sharded) across these independent nodes.
2. **Shared-Disk Parallelism**: Multiple CPUs share access to the same set of disks via an interconnect, but each CPU has its own private memory.
3. **Shared-Memory Parallelism**: Multiple CPUs share access to a common memory pool and often a common set of disks, usually within a single machine due to memory coherence and bandwidth limitations.

Question 8

What are the three main data partitioning strategies in data-parallelism? How do they relate to data layout strategies?

Answer: The three main data partitioning strategies for row-wise (horizontal) partitioning in data-parallelism across k nodes are:

1. **Round-robin**: Tuple i is assigned to node $i \pmod{k}$.
2. **Hashing-based**: Requires one or more attributes to be designated as hash partitioning attributes. This is very common in practice for RA and SQL workloads.
3. **Range-based**: Requires one or more ordinal partitioning attributes and is often good for range predicates in RA/SQL.

These strategies relate to data layout by physically distributing data blocks or tuples across different nodes or disks in a cluster. Beyond row-wise partitioning, data can also be laid out using **columnar partitioning** (partitioning data by columns across nodes) or **hybrid/tiled partitioning** (a combination of row-wise and columnar approaches). These strategies determine how the data is physically stored and accessed in a distributed environment to optimize for different types of queries or workloads.

Question 9

What is the main difference of peer-to-peer cluster architecture vs. manager-worker architecture? Name three example systems of each.

Answer: The main difference between peer-to-peer and manager-worker cluster architectures lies in their centralization of control:

- **Manager-Worker Architecture**: This is a centralized architecture where one or a few special nodes (Managers/Masters) dictate what the other worker nodes should do and when they should communicate. The manager coordinates tasks, distributes work, and collects results. **Examples**: Dask, Apache Spark, and parallel RDBMSs.
- **Peer-to-Peer Architecture**: This is a decentralized architecture where there is no single manager node. Workers communicate directly with each other to coordinate tasks and exchange data. **Example**: Horovod. (Note: The provided materials explicitly name only one example for peer-to-peer. Other systems like BitTorrent, or certain blockchain networks, also exhibit peer-to-peer characteristics, but are not mentioned in the provided context.)

Question 10

Briefly explain 2 reasons why BSP may not yield linear speedup.

Answer: Bulk Synchronous Parallelism (BSP) often does not yield linear speedup due to several overhead factors:

1. **Manager Overhead**: The manager node incurs overhead in dividing up work among workers and collecting or unifying partial results from them. This centralized coordination can become a bottleneck as the number of workers increases.
2. **Stragglers**: BSP involves barrier synchronization, meaning the manager must wait until all workers have completed their current task before proceeding to the next piece of work. If even one worker is significantly slower (a "straggler"), it can delay the entire computation, limiting the overall speedup.
3. **Communication Costs**: The time and resources spent on communication between the manager and workers, and potentially among workers themselves, add to the total execution time and reduce the efficiency of parallelization.
4. **Per-Worker Overhead**: Each worker incurs startup and tear-down costs, which can become significant in fine-grained parallel tasks.

Question 11

Briefly explain 1 pro and 1 con of NFS vs. HDFS.

Answer: Based on the provided materials, information on NFS is not explicitly detailed for a direct comparison. However, the characteristics of HDFS are discussed. **Hadoop Distributed File System (HDFS):**

- **Pro**: HDFS is highly scalable, designed to scale to thousands of nodes and petabytes of data, and is fault-tolerant through data replication.
- **Con**: HDFS primarily supports read-only access and batch-append operations. It does not offer efficient fine-grained updates or writes to existing data blocks, making it less suitable for workloads requiring frequent modifications to existing files.

(For a full comparison with NFS, additional information on NFS's characteristics, such as its POSIX compliance and fine-grained write capabilities, would be needed, which is beyond the scope of the provided documents.)

Question 12

Does data-parallelism apply to dataflow graphs or task graphs?

Answer: Data parallelism primarily applies to **dataflow graphs**. In dataflow graphs, the operations are applied to streams or sets of data, which aligns perfectly with the concept of applying the same operation to different partitions of data. The MapReduce model, for instance, is a data-parallel dataflow framework. While data-parallel computations can be represented within broader task graphs (which can be more coarse-grained and show dependencies between larger computational tasks), the core concept of data parallelism itself maps directly to the data transformation nature of dataflow graphs.

MapReduce and Spark Review Questions

Question 1

Briefly explain 2 reasons why parallel RDBMSs did not suffice for parallel processing of "Big Data."

a) **Developability:** Custom data models and computations were hard to program on SQL/RDBMSs, leading to a need for simpler APIs.

b) **Fault Tolerance:** RDBMSs were not designed for the graceful handling of worker failures at the scale of thousands of machines, which is common in "Big Data" environments. Other reasons include elasticity and cost.

Question 2

What is MapReduce? What is Hadoop? How are they different?

a) **MapReduce:** A programming model and an associated implementation for processing and generating large datasets with a parallel, distributed algorithm on a cluster. It defines two main functions: Map and Reduce.

b) **Hadoop:** A Free and Open-Source Software (FOSS) system that implements the MapReduce programming model and includes HDFS (Hadoop Distributed File System) as its underlying filesystem.

c) **Difference:** MapReduce is a programming model/paradigm, while Hadoop is a specific software framework that provides an implementation of the MapReduce model along with a distributed file system. Hadoop handles aspects like input splits, data distribution, shuffling, and fault tolerance automatically for MapReduce jobs.

Question 3

Can the data types of the inputs to Map and Reduce differ? Yes, the data types of the inputs to Map and Reduce functions can differ. The Map function takes an input (e.g., a document name and text) and emits key-value pairs (e.g., (word, 1)). The Reduce function then takes a key and an iterator of values emitted by the Map phase and produces an output (e.g., (word, sum of counts)).

Question 4

Which extended relational algebra operator does MapReduce resemble the most? MapReduce most closely resembles the **Group By and Aggregate** operator from extended relational algebra. The Map phase often processes records and emits keys for grouping, while the Reduce phase aggregates values within each group.

Question 5

How does Hadoop/MapReduce achieve fault tolerance? Hadoop/MapReduce achieves fault tolerance by automatically handling worker failures. If a worker fails during a Map or Reduce task, the system detects the failure and re-executes the failed task on another available worker. This is facilitated by replicating data blocks in HDFS and by the idempotent nature of many MapReduce tasks.

Question 6

What are Map-only jobs in MapReduce? Why do they arise?

a) **Map-only jobs:** These are MapReduce jobs where no Reduce phase is necessary. The entire computation can be completed by the Map tasks alone, often because no aggregation or grouping across different keys is required.

b) **Why they arise:** They typically arise when the task involves a row-wise or element-wise transformation or filtering operation on the data, where each input record can be processed independently without needing to combine results based on a key (e.g., projection, selection, or per-record feature extraction like Bag of Words).

Question 7

Suppose you are given a large relation stored as an HDFS file. You are asked to compute 3 AVG values on 3 different attributes. Write a single MapReduce job to compute the above. To compute 3 AVG values on 3 different attributes (say, $attr1$, $attr2$, $attr3$) in a single MapReduce job:

Map Function: For each tuple/record in the HDFS file:

Emit a global dummy key (e.g., "GLOBAL_AVG") and a value containing a tuple of:

$$(attr1_{value}, attr2_{value}, attr3_{value}, 1)$$

The 1 acts as a count for that record.

Reduce Function: For the global dummy key, the reducer receives an iterator of all emitted $(attr1_{value}, attr2_{value}, attr3_{value}, 1)$ tuples. It sums up all $attr1_{value}$, $attr2_{value}$, $attr3_{value}$ and counts the total number of records (sum of '1's). Then, it computes the average for each attribute and emits the final result.

Question 8

Briefly explain 2 reasons why the model building stage of the data science lifecycle can become challenging.

a) **Iterative Nature and Hyper-parameter Tuning:** Finding the optimal model configuration often requires extensive iterative experimentation, involving selecting algorithms, engineering features, and tuning numerous hyper-parameters. This trial-and-error process can be time-consuming and computationally expensive.

b) **Feature Engineering Complexity:** Transforming raw data into effective features can be difficult, time-consuming, and requires significant domain expertise. Poor features can severely limit model performance regardless of the chosen algorithm.

Question 9

What are the 3 decisions inherent to the model selection process? The three decisions inherent to the model selection process are:

a) **Feature Engineering (FE):** Deciding which features to use, create, or transform from the raw data.

b) **Algorithm Selection (AS):** Choosing the appropriate machine learning algorithm(s) for the task (e.g., Logistic Regression, XGBoost, Neural Networks).

c) **Hyper-Parameter Tuning (HT):** Selecting the optimal values for the model's hyper-parameters (parameters that are set before training begins).

Question 10

Give an example of a feature engineering step that can be scaled using a Map-only job. An example of a feature engineering step that can be scaled using a Map-only job is **Bag of Words (BoW)** or **N-gram extraction** from a text corpus. Each document can be processed independently by a Mapper to count word frequencies or N-gram occurrences without needing to communicate with other Mappers or a Reduce step.

Question 11

Why is hyper-parameter tuning needed? Name 2 tuning methods.

a) **Why needed:** Hyper-parameter tuning is needed because hyper-parameters act as "knobs" for an ML model or its training algorithm, controlling the bias-variance trade-off in a dataset-specific manner. Optimizing them is crucial for a model to learn effectively and achieve good performance on unseen data.

b) **Two tuning methods:**

- **Grid Search:** Systematically evaluates all possible combinations of hyper-parameter values from a predefined discrete set.
- **Random Search:** Samples hyper-parameter values randomly from defined distributions or ranges, often more efficient than grid search in high-dimensional spaces.

Question 12

Briefly explain 1 pro and 1 con of AutoML tools.

a) **Pro:** AutoML tools can significantly **reduce the manual effort and time** required for tasks like feature engineering, algorithm selection, and hyper-parameter tuning, making ML more accessible to non-experts and accelerating the development process.

b) **Con:** AutoML tools can be **computationally intensive** due to the extensive search space they explore. They might also lead to less interpretability or understanding of the underlying model choices compared to manual approaches.

Question 13

textbfSuppose you decide to try logistic regression and XGBoost for a tabular data classification task. You try 4 learning rates for each model, as well as 2 regularizer values for the logistic model and 3 values of number of trees for XGBoost. What is the total number of models trained in this model selection process? Let's calculate the number of models for each:

- **Logistic Regression:**
  - Learning rates: 4
  - Regularizer values: 2
  - Total Logistic Regression models $= 4 \times 2 = 8$

- **XGBoost:**
  - Learning rates: 4
  - Number of trees: 3
  - Total XGBoost models = $4 \times 3 = 12$

Total number of models trained = Total Logistic Regression models + Total XGBoost models = $8 + 12 = \mathbf{20}$ models.

Question 14

Briefly explain 2 differences between Hadoop and Spark.

a) **In-memory Processing vs. Disk-based:** Spark primarily exploits distributed memory to cache data between computations, leading to significantly faster iterative algorithms and interactive queries. Hadoop MapReduce, by contrast, heavily relies on disk I/O between stages.

b) **Fault Tolerance Mechanism:** Spark's fault tolerance is based on "lineage," where it records the transformations applied to data (DAG of operations). If a partition is lost, Spark can recompute it from its lineage. Hadoop's fault tolerance primarily relies on data replication (HDFS) and re-executing failed tasks.

Question 15

What is the relationship between Spark's RDD API, MapReduce, and extended relational algebra? Spark's RDD (Resilient Distributed Dataset) API is a low-level API that forms the core abstraction for distributed data processing. It provides transformations (like *map*, $filter$, *reduceByKey*) that generalize operations found in both MapReduce (e.g., *map* is similar to Map, *reduceByKey* is similar to Reduce) and extended relational algebra (e.g., $filter$ for selection, *map* for projection). Essentially, RDDs subsume most of the operations possible with MapReduce and relational algebra, offering a more flexible and unified dataflow programming model.

Question 16

What is SparkSQL? What is its relationship to RDD API?

a) **SparkSQL:** A Spark module for working with structured data. It provides a programming interface for Spark that allows users to query data using SQL or a DataFrame/Dataset API.

b) **Relationship to RDD API:** SparkSQL is built on top of the RDD API. DataFrames and Datasets in SparkSQL are essentially distributed collections of rows with a schema, which are implemented as RDDs internally. SparkSQL leverages Spark's Catalyst Optimizer to optimize queries, often compiling them down to efficient RDD operations.

Question 17

Why does Databricks recommend using SparkSQL/DataFrame API instead of RDD API nowadays? Databricks (the company behind Spark) recommends using SparkSQL/DataFrame API over the lower-level RDD API for several reasons:

- **Optimization:** DataFrames carry schema information, allowing Spark's Catalyst Optimizer to perform significant query optimizations (e.g., predicate pushdown, column pruning) that are not possible with RDDs.
- **Ease of Use:** DataFrame/Dataset API is more user-friendly and expressive for common data manipulation tasks, resembling Pandas DataFrames.
- **Performance:** Due to optimizations, DataFrames often yield better performance for structured data operations.

Question 18

Briefly describe 2 query optimizations implementing in SparkSQL.

a) **Predicate Pushdown:** This optimization pushes filtering conditions (WHERE clauses) as close to the data source as possible. This reduces the amount of data that needs to be read and shuffled, leading to significant performance improvements.

b) **Column Pruning:** This optimization identifies and removes columns that are not needed for the query result. By selecting only the necessary columns from the source, it reduces the amount of data processed, transferred over the network, and stored in memory. Other optimizations include join reordering, common subexpression elimination, and code generation.

Question 19

What is a "data lakehouse"? A "data lakehouse" is a new paradigm in data architecture that combines the best features of data lakes and data warehouses. It aims to provide the flexibility, scalability, and low cost of data lakes (which store raw, unstructured, and semi-structured data) with the data management features, schema enforcement, transaction support, and performance of data warehouses. This allows for both traditional BI and advanced analytics (ML) on the same platform.

Question 20

What is the difference between Koalas and DataFrame on Spark?

a) **Spark DataFrame:** Spark's native high-level API for structured data, offering distributed data processing capabilities.

b) **Koalas (now part of PySpark):** A Python library that implements the pandas API on top of Apache Spark. Its primary purpose is to make it easier for pandas users to transition to Spark for big data tasks by providing a familiar API while leveraging Spark's distributed processing power.

c) **Difference:** Koalas is essentially a wrapper or an implementation of the pandas API that uses Spark DataFrames under the hood. While a Spark DataFrame is the core distributed data structure in Spark's high-level API, Koalas provides a syntax and functionality that mirrors pandas, allowing users to write pandas-like code that executes on a Spark cluster.

Quizzes and Midterm

Quiz 1

Question 1. Which of these structured data models has native support for mixed types in columns?

a) Matrix
b) Relation
c) All of the three
d) DataFrame

Answer: d) DataFrame

- *Explanation:* DataFrames, commonly found in libraries like Pandas or Spark, are designed to handle heterogeneous data types within different columns, similar to a spreadsheet or a table in a relational database. Matrices typically require all elements to be of the same numeric type. Relations (in the context of relational databases) enforce strict data types for each column, though they can vary *between* columns.

Question 2. In what way is data access control often a challenge in the data sourcing stage?

a) Limits access to some ML software
b) Prevents use of some data
c) Raises computational resource costs

d) Requires downsampling of data

Answer: b) Prevents use of some data

- *Explanation:* Data access control mechanisms, while crucial for security and privacy, can often restrict data scientists from accessing necessary datasets due to permissions, compliance regulations, or internal policies. This directly impacts the ability to source and utilize all potentially relevant data for an ML project.

Question 3. Which type of integrity constraint ensures that a value in one table must match a value in another table?

a) Functional dependency
b) Domain integrity
c) Referential integrity
d) Entity integrity

Answer: c) Referential integrity

- *Explanation:* Referential integrity is enforced using foreign keys, which establish a link between data in two tables. It ensures that a foreign key value in the referencing table corresponds to an existing primary key value in the referenced table, preventing orphaned records and maintaining consistency across related data.

Question 4. What is the purpose of a primary key in a relational database table?

a) To improve performance
b) To ensure data is ordered
c) To ensure data is unique
d) To enforce referential integrity

Answer: c) To ensure data is unique

- *Explanation:* The fundamental purpose of a primary key is to uniquely identify each record (row) in a table. While it can be used to improve performance (often by indexing) and is essential for referential integrity, its core role is to guarantee uniqueness.

Question 5. What is the purpose of a foreign key in a relational database table?

a) To improve performance
b) To ensure data is ordered
c) To enforce referential integrity
d) To ensure data is unique

Answer: c) To enforce referential integrity

- *Explanation:* As discussed in Question 3's explanation, a foreign key's primary role is to enforce referential integrity. It creates a link between two tables, ensuring that relationships between data are valid and consistent.

Quiz 2

Question 1. If the primary key of a relation consists of the attributes A;B, then no record can have A = B.

a) True
b) False

Answer: b) False

- *Explanation:* A primary key consisting of attributes (A, B) means that the *combination* of values for A and B must be unique for each record. It does not mean that the individual values of A and B cannot be equal within a record. For example, if (A,B) is the primary key, a tuple (5,5) is perfectly valid, as long as there isn't another tuple (5,5) in the table. The constraint is on the uniqueness of the *pair*, not on the individual values within the pair.

Question 2. The table Arc(x,y) currently has the following tuples (note there are duplicates): (1,2), (1,2), (2,3), (3,4), (3,4), (4,1), (4,1), (4,1), (4,2). Compute the result of the query: Which of the following tuples is in the result? SQL query:

```
SELECT a1.x, a2.y, COUNT(*)
FROM Arc a1, Arc a2
WHERE a1.y = a2.x
GROUP BY a1.x, a2.y;
```

a) (1,2,4)
b) (3,3,1)
c) (2,4,6)
d) (4,3,1)

Answer: d) (4,3,1)

- *Explanation:* This query performs a self-join on the 'Arc' table where 'a1.y = a2.x', and then groups by 'a1.x' and 'a2.y' to count occurrences. Let's trace the join and grouping:
  - The 'WHERE a1.y = a2.x' condition looks for paths (x -¿ y -¿ z).
  - Consider 'a1.x = 4'. The tuples in 'Arc' starting with 4 are (4,1), (4,1), (4,1), (4,2).
  - If 'a1.x = 4' and 'a1.y = 1', then 'a2.x' must be 1. The tuples in 'Arc' with 'x = 1' are (1,2), (1,2). This would lead to (4,2) combinations. Since there are 3 instances of (4,1) in 'a1' and 2 instances of (1,2) in 'a2', this contributes $3 \times 2 = 6$ to the count for (4,2).
  - If 'a1.x = 4' and 'a1.y = 2', then 'a2.x' must be 2. The tuples in 'Arc' with 'x = 2' is (2,3). This leads to (4,3) combinations. Since there is 1 instance of (4,2) in 'a1' and 1 instance of (2,3) in 'a2', this contributes $1 \times 1 = 1$ to the count for (4,3).
  - Therefore, the tuple (4,3,1) is present in the result.

Question 3. For any set of attributes X, the set X+ is a superkey.

a) True
b) False

Answer: b) False

- *Explanation:* X+ (the closure of X) represents all attributes that are functionally determined by X. While a key (and thus a superkey) must functionally determine all other attributes in a relation, the set X+ itself is not necessarily a superkey. A superkey is a set of attributes that *uniquely identifies* tuples. X+ tells you what attributes X can *determine*, not necessarily that X itself is a unique identifier. A superkey must contain a candidate key. X+ might not contain a candidate key, or X itself might not be minimal. For X to be a superkey, its closure X+ must include all attributes of the relation.

Question 4. Suppose relation R(A,B,C) has the tuples: Table:

```
A | B | C
--|---|--
0 | 1 | 2
0 | 1 | 3
4 | 5 | 6
4 | 6 | 3
```

Compute the bag union of the following three expressions, each of which is the bag projection of a grouping (γ) operation:

1. πX(γA,B,MAX(C)$\rightarrow$X(R)) 2. πX(γB,SUM(C)$\rightarrow$X(R)) 3. πX(γA,MIN(B)$\rightarrow$X(R))

Demonstrate that you have computed this bag correctly by identifying, from the list below, the correct count of occurrences for one of the elements.

a) 3 appears exactly three times
b) 6 appears exactly once
c) 1 appears exactly twice
d) 3 appears exactly twice

Answer: a) 3 appears exactly three times

- *Explanation:* Let's break down each expression:
  (a) πX(γA,B,MAX(C)$\rightarrow$X(R)):
    - Group by (A,B):
    - (0,1): MAX(C) is 3 (from (0,1,2) and (0,1,3))
    - (4,5): MAX(C) is 6 (from (4,5,6))
    - (4,6): MAX(C) is 3 (from (4,6,3))
    - Projection πX gives us: 3, 6, 3 (as a bag)
  (b) πX(γB,SUM(C)$\rightarrow$X(R)):
    - Group by B:
    - B=1: SUM(C) is 2+3=5 (from (0,1,2) and (0,1,3))
    - B=5: SUM(C) is 6 (from (4,5,6))
    - B=6: SUM(C) is 3 (from (4,6,3))
    - Projection πX gives us: 5, 6, 3 (as a bag)
  (c) πX(γA,MIN(B)$\rightarrow$X(R)):
    - Group by A:
    - A=0: MIN(B) is 1 (from (0,1,2) and (0,1,3))
    - A=4: MIN(B) is 5 (from (4,5,6) and (4,6,3))
    - Projection πX gives us: 1, 5 (as a bag)

  Now, compute the bag union of 3, 6, 3, 5, 6, 3, and 1, 5. Bag union means we combine all elements and keep their counts. Resulting Bag: 1, 3, 3, 3, 5, 5, 6, 6
  - 1 appears once.
  - 3 appears three times.
  - 5 appears twice.
  - 6 appears twice.

  Therefore, "3 appears exactly three times" is the correct statement.

Question 5. If the attribute K of a relation is a key, then no two tuples in the relation can have the same value of K.

a) True
b) False

Answer: a) True

- *Explanation:* This is the fundamental definition of a key in a relational database. A key (which can be a primary key or any other candidate key) is a set of attributes whose values uniquely identify each tuple (row) in a relation. If two tuples had the same value for K, then K would not be able to uniquely identify them, violating its definition as a key.

Quiz 3

Question 1. Which of the following kinds of predicates in a selection query is amenable to being sped up using both a hash index and B+ tree index?

a) Equal to
b) Less than
c) Not equal to
d) Greater than or equal to

Answer: a) Equal to

- *Explanation:*
  - **Hash Index:** Hash indexes are highly efficient for "equal to" (‘=‘) predicates because they directly map a key value to its storage location using a hash function. This provides very fast lookups for exact matches.
  - **B+ Tree Index:** B+ trees are also excellent for "equal to" predicates. They provide logarithmic time complexity for searching, and once the key is found, they can quickly retrieve the corresponding data.
  - **Range Queries (Less than, Greater than or equal to):** B+ trees are uniquely suited for range queries because their leaf nodes are linked in a sorted order, allowing for efficient traversal of ranges (e.g., ‘WHERE x ¿ 10‘ or ‘WHERE x BETWEEN 5 AND 15‘). Hash indexes are generally *not* efficient for range queries because the hashing function distributes keys randomly, breaking sequential order.
  - **Not Equal To:** Neither index type is particularly efficient for "not equal to" (‘!=‘) predicates. Such queries typically require scanning a large portion of the data, as most values will satisfy the condition.

  Therefore, only "equal to" queries benefit significantly from both hash and B+ tree indexes.

Question 2. What is the selectivity of the following query when applied to the following instance of the Ratings relation? SQL query:

```
SELECT * FROM Ratings WHERE NOT (Stars > 3.0);
```

Table:

RatingID	Stars	UserID	MovieID
1	4.0	794	223
2	3.0	802	034
3	4.0	795	342
4	2.0	123	425
5	5.0	322	0

a) 60%
b) 20%
c) 80%
d) 40%

Answer: d) 40%

- *Explanation:* Selectivity is the fraction of rows that satisfy a query condition. The condition is ‘NOT (Stars ¿ 3.0)‘, which is equivalent to ‘Stars ¡= 3.0‘. Let's check each tuple in the ‘Ratings‘ table:
  - RatingID 1: Stars = 4.0. Is 4.0 ¡= 3.0? No.
  - RatingID 2: Stars = 3.0. Is 3.0 ¡= 3.0? Yes. (Tuple selected)

    - RatingID 3: Stars = 4.0. Is 4.0 ¡= 3.0? No.
    - RatingID 4: Stars = 2.0. Is 2.0 ¡= 3.0? Yes. (Tuple selected)
    - RatingID 5: Stars = 5.0. Is 5.0 ¡= 3.0? No.

  Two tuples (RatingID 2 and 4) satisfy the condition. There are a total of 5 tuples in the table. Selectivity = (Number of selected tuples) / (Total number of tuples) = 2 / 5 = 0.4 = 40%.

Question 3. Given a relation with 4 attributes, how many different hash indexes can be built on this relation?

a) 14
b) 20
c) 12
d) 18

Answer: d) 18

- *Explanation:* A hash index can be built on any single attribute or any combination of attributes. If a relation has 'N' attributes, the number of possible hash indexes is the number of non-empty subsets of those attributes. This is given by the formula $2^N - 1$. For a relation with 4 attributes (let's say A, B, C, D):
    - Single attribute indexes: A, B, C, D (4 combinations)
    - Two-attribute combinations: (A,B), (A,C), (A,D), (B,C), (B,D), (C,D) (6 combinations)
    - Three-attribute combinations: (A,B,C), (A,B,D), (A,C,D), (B,C,D) (4 combinations)
    - Four-attribute combinations: (A,B,C,D) (1 combination)

  Total $= 4+6+4+1 = 15$. However, the question implies that the order of attributes matters for composite keys in some contexts (e.g., for B-trees sometimes, but for hash, order typically doesn't matter for the key itself, just for unique sets). Let's re-evaluate based on the more common interpretation in database systems where each unique set of attributes can form a composite index. The number of non-empty subsets is $2^N - 1$. For N=4, $2^4 - 1 = 16 - 1 = 15$.

  Self-correction/Refinement: The common interpretation of "different hash indexes" often refers to distinct sets of columns, leading to $2^N - 1$. However, sometimes in a multiple choice question like this, if 15 isn't an option, it hints at considering permutations or a slightly different interpretation. Let's reconsider.

  If we consider that for a composite index, say on (A,B), it is distinct from (B,A) *in some specific index implementations*, or if the choices hint at a slightly different combinatorics. However, for hash indexes, the order of attributes in a composite key typically does not change the resulting hash value or the index structure for that set of attributes. A hash function takes a tuple of values (A,B) as input; the order might be fixed by convention, but (A,B) and (B,A) as *sets of attributes* are the same.

  Let's assume the question implicitly refers to *all possible non-empty ordered subsets* of attributes if it's not strictly a "set" interpretation, or if there's a misunderstanding of how the options are derived. Let's assume the more general understanding for *composite indexes* (which can be B-tree or hash-based) where 'INDEX (A, B)' is distinct from 'INDEX (B, A)'. For N attributes: 1 attribute: N choices (e.g., A, B, C, D) = 4 2 attributes: $P(N, 2)$ permutations $= 4 \times 3 = 12$ (e.g., (A,B), (B,A), (A,C), (C,A)...) 3 attributes: $P(N, 3)$ permutations $= 4 \times 3 \times 2 = 24$ 4 attributes: $P(N, 4)$ permutations $= 4 \times 3 \times 2 \times 1 = 24$ This leads to a very large number.

  Revisiting the problem context: In database systems, an index is defined on a *set of columns*, and the order often matters for B+ trees for prefix matching, but for hash indexes, it's generally about the set. The options provided (14, 18, 12, 20) are relatively small. This strongly suggests the question might be about non-empty subsets PLUS perhaps distinct orders for a subset. Let's re-evaluate the options with the simplest interpretation: Number of attributes: 4 (A, B, C, D) Number of non-empty subsets: $2^4 - 1 = 15$. Since 15 is not an option, there's likely a nuance being tested.

  Could it be asking for "how many different *ways* to construct a hash index" based on properties? Let's re-examine standard counting principles. If a hash index can be on *any combination of attributes*, it's the power set minus the empty set. $2^4 - 1 = 15$. Perhaps the question is not about *all* possible hash indexes, but limited to single-attribute and a specific number of composite indexes?

Let's consider the possible interpretations for why the answer is 18, as 15 is the standard combinatoric result for distinct sets of attributes. One common scenario where N-choose-K is augmented is if attributes can be indexed independently and combined, but that's usually for multi-column indexes rather than distinct hash indexes.

This specific question might be testing a very particular definition or a common misinterpretation. If we exclude the 4-attribute combination, we have $15 - 1 = 14$. This is option (a). If we consider only single attributes and pairs? $4 + 6 = 10$. Let's stick to the $2^N - 1$ as the most general interpretation for "different hash indexes" as distinct sets of columns indexed. Given that 15 is not an option and 18 is, this points to a specific pedagogical context or a slight deviation from the typical $2^N - 1$ interpretation.

Hypothesis for 18: Could it be $N^2 + N/2$ or something similar? $4^2 + 4/2 = 16 + 2 = 18$? No. N single attributes: 4 $N(N-1)/2$ combinations of 2 attributes: $4 \times 3/2 = 6$ $N(N-1)(N-2)/6$ combinations of 3 attributes: $4 \times 3 \times 2/6 = 4$ 1 combination of 4 attributes. Total $= 4 + 6 + 4 + 1 = 15$.

Let's consider if the question implies order matters for two-attribute and three-attribute combinations, but not for one or four. This would be a very unusual interpretation for "hash index" where order is typically irrelevant to the hash value itself.

Re-evaluating based on common test question pitfalls or specific curriculum: If it's about *ordered permutations* of attributes for composite keys, for N attributes, the number of distinct ordered sequences of attributes that can form a key is: $P(N,1) + P(N,2) + P(N,3) + ... + P(N,N)$ where $P(N,k) = N!/(N-k)!$ For N=4: $P(4,1) = 4$ $P(4,2) = 4 \times 3 = 12$ $P(4,3) = 4 \times 3 \times 2 = 24$ $P(4,4) = 4 \times 3 \times 2 \times 1 = 24$ Total $= 4 + 12 + 24 + 24 = 64$. This is far too high.

The most plausible scenario for 18 as an answer, given 4 attributes: If it were $N \times (N+1)/2$ for unique pairs plus singletons. This question's answer of 18 is an outlier if based purely on "distinct sets of attributes". Let's go with the provided answer and try to reverse-engineer a plausible reason, which is typical in some multiple-choice scenarios.

A common interpretation for number of ways to pick k items from N *with replacement* where order matters, or something complex.

Let's consider a scenario that might lead to 18: Number of single attribute indexes: 4 Number of 2-attribute combinations (order does not matter): 6 Number of 3-attribute combinations (order does not matter): 4 Number of 4-attribute combinations (order does not matter): 1 Total combinations = 15.

The only way to get to 18 would be if certain "types" of indexes were allowed. Perhaps 4 (single) + 6 (pairs, unordered) + some other category. What if it's $N^2 + 2$? No.

The most likely, if not purely combinatoric, reason for 18: This specific number (18 for N=4) sometimes appears in contexts where there are N single-column indexes, and then a specific number of composite indexes based on some pattern. Without further context or a specific formula provided by the course, deriving 18 purely from standard set theory on 4 attributes is difficult if 15 (which is $2^N - 1$) is the expected number of unique attribute *sets*.

However, let's consider that sometimes the question implies *ordered pairs* for composite keys, and then also single keys. If we take single keys (4) and ordered pairs of keys ($P(4,2) = 12$), that sums to $4 + 12 = 16$. Still not 18. If it includes $P(4,2)+$ single column indexes.

This question's answer (18) is the most difficult to justify based on standard database theory for "number of distinct hash indexes" where a hash index is typically defined over a *set* of attributes. The answer of 15 ($2^N - 1$) is the standard combinatorial result. However, for the sake of providing an explanation consistent with the provided answer, this implies a counting method that goes beyond simple power set cardinality. Without a specific rule or context from the course, it's hard to explain 18 precisely. It might stem from an obscure counting rule or a specific interpretation of "different hash indexes" where certain permutations are counted, even if not typical for hash index definitions. Given this ambiguity, for educational purposes, I'll state the standard understanding and acknowledge the discrepancy.

Revised Explanation for Q3: A hash index can be built on any single attribute or any combination of attributes. The number of unique sets of attributes from a set of N attributes is $2^N - 1$ (excluding the empty set). For 4 attributes, this would be $2^4 - 1 = 15$. However, the given answer is 18. This suggests a different counting method or interpretation. One possible, though less common, interpretation could involve considering certain ordered combinations of attributes as "different" indexes, or perhaps a context-specific rule from the course material that is not standard $2^N - 1$ combinatorics. Without further information, the most direct explanation based on fundamental database principles points to 15. If 18 is definitively the correct answer in your course, it implies a nuance that should be clarified by your instructor or course materials. For the purpose of this explanation, we will acknowledge the given answer.

Question 4. Compute the intersection of the relations R(A,B,C) and S(A,B,C): Table R:

a	b	c
1	2	3
4	2	3
4	5	6
2	5	3
1	2	6

Table S:

a	b	c
2	5	3
2	5	4
4	5	6
1	2	3

Which of the following tuples appears in R ∩ S?

a) (1,2,3)
b) (2,5,4)
c) (2,4,3)
d) (1,2,6)

Answer: a) (1,2,3)

- *Explanation:* The intersection of two relations (R ∩ S) includes only those tuples that are present in *both* relations. Let's compare the tuples:
  - (1,2,3): Present in R, Present in S. -¿ IN INTERSECTION
  - (4,2,3): Present in R, Not in S.
  - (4,5,6): Present in R, Present in S. (So (4,5,6) would also be in the intersection, but it's not an option).
  - (2,5,3): Present in R, Present in S. (So (2,5,3) would also be in the intersection, but it's not an option).
  - (1,2,6): Present in R, Not in S.
  - (2,5,4): Not in R. (So not in intersection)

  From the given options, only (1,2,3) is present in both R and S.

Question 5. Suppose relation R(A,B) has the tuples: relation R(A,B):

A	B
1	2
3	4
5	6

and the relation S(B,C,D) has tuples:

a	b	c
2	4	6
4	6	8
4	7	9

Compute the theta-join of R and S with the condition R.A ¡ S.C AND R.B ¡ S.D. Then, identify from the list below one of the tuples in R $\bowtie_{R.A<S.C \text{ AND } R.B<S.D}$ S. You may assume the schema of the result is (A, R.B, S.B, C, D).

a) (5,6,2,4,6)
b) (3,4,5,7,9)
c) (1,2,2,6,8)
d) (1,2,2,4,6)

Answer: d) (1,2,2,4,6)

- *Explanation:* A theta-join combines tuples from two relations based on a specified condition. The schema of the result is a concatenation of the schemas of R and S. The join condition is ‘R.A ¡ S.C AND R.B ¡ S.D‘.
  Let’s iterate through each tuple in R and compare it with each tuple in S:
  From R: (1,2)
  - Against S: (2,4,6)
  - R.A ¡ S.C (1 ¡ 4)? Yes.
  - R.B ¡ S.D (2 ¡ 6)? Yes.
  - Both conditions met. Result tuple: (1,2,2,4,6). This matches option (d).
  - Against S: (4,6,8)
  - R.A ¡ S.C (1 ¡ 6)? Yes.
  - R.B ¡ S.D (2 ¡ 8)? Yes.
  - Both conditions met. Result tuple: (1,2,4,6,8). (Not an option)
  - Against S: (4,7,9)
  - R.A ¡ S.C (1 ¡ 7)? Yes.
  - R.B ¡ S.D (2 ¡ 9)? Yes.
  - Both conditions met. Result tuple: (1,2,4,7,9). (Not an option)

  From R: (3,4)
  - Against S: (2,4,6)
  - R.A ¡ S.C (3 ¡ 4)? Yes.
  - R.B ¡ S.D (4 ¡ 6)? Yes.
  - Both conditions met. Result tuple: (3,4,2,4,6). (Not an option)
  - Against S: (4,6,8)
  - R.A ¡ S.C (3 ¡ 6)? Yes.
  - R.B ¡ S.D (4 ¡ 8)? Yes.
  - Both conditions met. Result tuple: (3,4,4,6,8). (Not an option)
  - Against S: (4,7,9)
  - R.A ¡ S.C (3 ¡ 7)? Yes.
  - R.B ¡ S.D (4 ¡ 9)? Yes.
  - Both conditions met. Result tuple: (3,4,4,7,9). (Not an option)

  From R: (5,6)
  - Against S: (2,4,6)
  - R.A ¡ S.C (5 ¡ 4)? No. (Condition fails)

– Against S: (4,6,8)
– R.A ¡ S.C (5 ¡ 6)? Yes.
– R.B ¡ S.D (6 ¡ 8)? Yes.
– Both conditions met. Result tuple: (5,6,4,6,8). (Not an option)
– Against S: (4,7,9)
– R.A ¡ S.C (5 ¡ 7)? Yes.
– R.B ¡ S.D (6 ¡ 9)? Yes.
– Both conditions met. Result tuple: (5,6,4,7,9). (Not an option)

From the given options, only (1,2,2,4,6) is a valid result of the theta-join.

Quiz 4

Question 1. Which of the following is not a syntactic feature of XML documents?

a) Middle tag
b) None of the three
c) Start tag
d) End Tag

Answer: a) Middle tag

- *Explanation:* XML documents are structured using a hierarchy of elements, defined by 'start tags' (e.g., '¡element¿') and 'end tags' (e.g., '¡/element¿'). There is no concept of a "middle tag" in XML syntax. Well-formed XML requires every start tag to have a matching end tag (or be an empty-element tag, e.g., '¡element/¿').

Question 2. Which capability of semi-structured data models enables storing arbitrary non-normalized data unlike the relational data model?

a) Schema-later approach
b) None of the three
c) Different attributes across records
d) Nesting of records

Answer: d) Nesting of records

- *Explanation:* Semi-structured data models (like JSON or XML) allow for hierarchical or 'nested' structures within a single record. This means that a single field can contain an entire sub-document or array of values, enabling the representation of complex, non-normalized data without requiring the flattening of structures into multiple tables, which is characteristic of the relational model. While "different attributes across records" is also a feature of semi-structured data flexibility, 'nesting of records' is the primary mechanism that directly allows for the arbitrary non-normalized, hierarchical data storage that contrasts sharply with the flat, normalized tables of relational databases.

Question 3. Which of these paradigms of parallelism is most common in data systems?

a) Shared disk
b) Shared CPU
c) Shared nothing
d) Shared memory

Answer: c) Shared nothing

- *Explanation:* The 'shared nothing' architecture is the most prevalent paradigm for scalable, distributed data systems (like Hadoop, Spark, NoSQL databases). In this architecture, each node in the cluster has its own dedicated CPU, memory, and storage, and they communicate only by passing messages over a network. This design minimizes contention, provides high fault tolerance, and allows for near-linear scalability, as adding more nodes directly increases total resources.

Question 4. Which component of Dask divides up the work to different nodes?

a) Worker
b) Client
c) Scheduler
d) Dispatcher

Answer: c) Scheduler

- *Explanation:* In Dask's distributed architecture, the 'Scheduler' is the central component responsible for coordinating computation. It receives tasks from the 'Client', builds the task graph, optimizes it, and then dispatches these tasks to the 'Workers' for execution. It also monitors the workers and manages data transfer between them.

Question 5. Which data partitioning strategy enables full scalability along both the number of rows and number of columns of a matrix?

a) Row-oriented
b) Tile-oriented
c) Column-oriented
d) All of the three

Answer: b) Tile-oriented

- *Explanation:*
  - 'Row-oriented' partitioning divides a matrix by rows, allowing scalability along the number of rows but not inherently along columns.
  - 'Column-oriented' partitioning divides a matrix by columns, allowing scalability along the number of columns but not inherently along rows.
  - 'Tile-oriented' partitioning (also known as block partitioning) divides the matrix into smaller, rectangular blocks or "tiles." This strategy allows for independent processing of these blocks and enables scalability in both row and column dimensions, as new rows or columns can be added, and the matrix can continue to be processed in these smaller, manageable tiles across a distributed system.

Question 6. Which capability of semi-structured data models enables storing arbitrary non-normalied data unlike the relational data model?

a) scheme-later approach
b) none of these
c) different attributes across records
d) nesting of records

Answer: d) nesting of records

- *Explanation:* This is a repeat of Question 2. As explained, the ability to embed complex structures within a single field (like lists within a field or sub-documents) directly allows for non-normalized, hierarchical data storage, which is a key distinction from relational models.

Question 7. Which data partitioning strategy enables full scalability along both the number of rows and number of columns of a matrix?

a) row-oriented
b) tile-oriented
c) column-oriented
d) all three

Answer: b) tile-oriented

- *Explanation:* This is a repeat of Question 5. As explained, 'tile-oriented' partitioning divides a matrix into rectangular blocks, offering scalability in both row and column dimensions for distributed processing.

Quiz 5

Question 1. Which of the following systems capabilities is a key differentiator of data "lakehouses" as against data warehouses?

a) Integration with business intelligence tools
b) SQL as a user-facing language
c) Loose coupling of file format with query stack
d) Automated query optimization

Answer: c) Loose coupling of file format with query stack

- *Explanation:*
  - 'Data warehouses' traditionally tightly couple their storage format (often proprietary or highly optimized for their specific query engine) with the query engine itself.
  - 'Lakehouses', however, combine the flexibility of data lakes (which store data in open formats like Parquet, ORC in object storage) with the analytical capabilities of data warehouses. This means they offer a 'loose coupling of the file format with the query stack'. You can use various query engines (Spark SQL, Presto, Hive, etc.) on the same underlying data files in the data lake, without being locked into a single vendor or format.
  - Options a), b), and d) are common features of both modern data warehouses and lakehouses, so they are not key differentiators *between* them.

Question 2. What was the main novel technical capability of Spark relative to parallel RDBMSs when it was introduced?

a) Optimized query execution
b) Scales to multi-node clusters
c) High-level querying/API
d) Lineage-based fault tolerance

Answer: d) Lineage-based fault tolerance

- *Explanation:* When Spark was introduced, while it offered capabilities like optimized query execution and high-level APIs similar to what parallel RDBMSs aimed for, and scalability to multi-node clusters (which MapReduce also did), its truly novel technical capability was 'lineage-based fault tolerance' through Resilient Distributed Datasets (RDDs). Spark could reconstruct lost partitions of data by replaying the transformations (the "lineage") that produced them, rather than relying on full data replication or checkpoints, leading to significant performance gains, especially for iterative algorithms and interactive queries, compared to the disk-heavy MapReduce.

Question 3. Which of the following feature engineering steps requires only a Map-only job to fully scale using MapReduce?

a) Whitening
b) Pairwise feature interactions
c) One-hot encoding
d) All of the three

Answer: b) Pairwise feature interactions

- *Explanation:*
  - A 'Map-only job' in MapReduce means that each input record can be transformed independently without needing to aggregate or combine information from other records.
  - 'Pairwise feature interactions' (e.g., creating a new feature by multiplying two existing features $F1 \times F2$) can be done entirely within the Map phase. For each input record, you simply access the values of $F1$ and $F2$ for that record and compute the new feature. This is a local transformation per record.

  - ‘One-hot encoding‘ often requires knowing the *global* vocabulary or distinct values for a categorical feature to create the appropriate number of binary columns. While mapping can start, a Reduce step (or a global pass beforehand) is typically needed to collect all unique categories.
  - ‘Whitening‘ (or Z-score normalization) requires computing global statistics like the mean and standard deviation of a feature across the *entire dataset*. This necessitates a Reduce step to aggregate these statistics before the actual transformation can occur.

Therefore, only ‘pairwise feature interactions‘ can be fully scaled using only a Map-only job.

Question 4. Which of the following is not a major type of data cleaning tasks?

a) Local edits to cell values
b) Reconciling across tuples
c) Synthesizing table values
d) Reconciling values in a column

Answer: c) Synthesizing table values

- *Explanation:*
  - ‘Data cleaning‘ primarily involves identifying and correcting errors, inconsistencies, and inaccuracies in data.
  - ‘Local edits to cell values‘ (e.g., correcting typos, standardizing formats within a single cell) are core cleaning tasks.
  - ‘Reconciling across tuples‘ (e.g., entity resolution, deduplication where information from multiple records is combined or corrected) is a major cleaning task.
  - ‘Reconciling values in a column‘ (e.g., ensuring consistency of units, handling missing values in a column) is also a key cleaning activity.
  - ‘Synthesizing table values‘ refers to generating new data or inferring values that were not originally present, often for purposes like data augmentation or imputation, but it’s not typically classified as a ”cleaning” task, which focuses on rectifying *existing* erroneous data.

Question 5. What is an outlier?

a) A data point that is duplicated in the dataset
b) A data point that is missing from the dataset
c) A data point that deviates from the norm
d) A data point that has a value of zero

Answer: c) A data point that deviates from the norm

- *Explanation:* An ‘outlier‘ is a data point that significantly differs from other observations. It’s an observation that lies an abnormal distance from other values in a random sample from a population. Outliers can indicate variability in a measurement, experimental errors, or a novelty. They are distinct from duplicates (repeated entries), missing values (absent data), or zero values (which can be normal).

Midterm

Question 1. Consider the following chart showing Pareto tradeoffs of 4 different ML models on two metrics of interest. Suppose you are told that model D is Pareto-optimal. For which of the following metrics on the X axis will that make sense?
Chart description:
Axes: y-axis Prediction accuracy, x-axis Metric TBD
Points: A(1,2), B(1,1), C(2,1), D(2,2)

a) Training throughput (examples per second)
b) Training cost (dollars)

c) Space footprint (bytes)

d) Inference latency (seconds)

Answer: a) Training throughput (examples per second)

- *Explanation:* Pareto optimality in this context means that a model is optimal if you cannot improve one metric without worsening another. The Y-axis is "Prediction accuracy," which is a "higher-is-better" metric. For a model D(2,2) to be Pareto-optimal when compared to A(1,2), B(1,1), C(2,1):
  - If X-axis is "Training cost" (lower is better): D (cost 2) is worse than A (cost 1) in cost for the same accuracy (accuracy 2). So D would not be Pareto-optimal.
  - If X-axis is "Space footprint" (lower is better): Similar to cost, D (footprint 2) is worse than A (footprint 1) for the same accuracy. So D would not be Pareto-optimal.
  - If X-axis is "Inference latency" (lower is better): Similar again, D (latency 2) is worse than A (latency 1) for the same accuracy. So D would not be Pareto-optimal.
  - If X-axis is "Training throughput (examples per second)" (higher is better):
    * Compared to A(1,2): D (throughput 2, accuracy 2) is better than A (throughput 1, accuracy 2) in throughput, and equal in accuracy. So A is dominated by D.
    * Compared to B(1,1): D (throughput 2, accuracy 2) is better than B (throughput 1, accuracy 1) in both. So B is dominated by D.
    * Compared to C(2,1): D (throughput 2, accuracy 2) is better than C (throughput 2, accuracy 1) in accuracy, and equal in throughput. So C is dominated by D.

    In this case, D is not dominated by any other model, making it Pareto-optimal. This makes sense only if the X-axis metric is also "higher-is-better".

Question 2. Which structured data model(s) support(s) in-place edits to the data?

a) Relation

b) DataFrame

c) Matrix

d) All of the three

Answer: d) All of these

- *Explanation:*
  - 'Relation' (in a relational database): Relational databases are designed for transactional workloads, which include 'UPDATE' operations that perform in-place edits to data.
  - 'DataFrame' (e.g., in Pandas or R): DataFrames are mutable objects in memory. You can directly assign new values to cells or columns, effectively performing in-place edits.
  - 'Matrix' (e.g., in NumPy or R): Matrices, when stored in mutable data structures, allow for direct modification of individual elements (e.g., $matrix[row, col] = new_value$).

  Therefore, all three data models, when implemented in typical software systems, support in-place edits to data.

Question 3. Which of the following best describes the relational data model?

a) A data model that organizes data into a hierarchy of entities and relationships

b) A data model that organizes data into a tree structure

c) A data model that organizes data into a set of tables with columns and rows

d) A data model that organizes data into a graph structure

Answer: c) A data model that organizes data into a set of tables with columns and rows

- *Explanation:* This is the fundamental definition of the relational data model, as introduced by Edgar Codd. Data is organized into two-dimensional 'tables' (also called 'relations'), where each table consists of named 'columns' (attributes) and unordered 'rows' (tuples). Relationships between tables are established through shared attributes (keys).

Question 4. Here is a table representing a relation S Table:

StudentID	Name	Age	Gender
1	John Smith	22	Male
2	Jane Smith	20	Female
3	Zhang San	21	Male

Identify:

1. The attributes of S.
2. The schema of S.
3. The tuples of S.
4. The components of the tuples for each attribute of S.

Which of the following is NOT a true statement about relation S?

a) Age is an attribute of S
b) S has four attributes
c) (3, Zhang San, 21, Male) is a tuple of R
d) S has four tuples

Answer: d) S has four tuples

- *Explanation:* Let’s evaluate each statement:
  - **a) Age is an attribute of S:** True. ‘Age‘ is clearly listed as a column header, meaning it’s an attribute (or schema element) of the relation.
  - **b) S has four attributes:** True. The attributes are ‘StudentID‘, ‘Name‘, ‘Age‘, and ‘Gender‘. There are indeed four distinct attributes.
  - **c) (3, Zhang San, 21, Male) is a tuple of R:** True. This is exactly one of the rows (tuples) in the provided table. (Note: The question says ”a tuple of R” but the table is named S. Assuming it refers to relation S.)
  - **d) S has four tuples:** False. The table explicitly shows three rows of data: 1. (1, John Smith, 22, Male) 2. (2, Jane Smith, 20, Female) 3. (3, Zhang San, 21, Male) Therefore, relation S has three tuples, not four.

Question 5. Suppose relations R(A,B) and S(B,C,D) have the tuples shown below: Tables: relation R(A,B):

A	B
1	2
3	4
5	6

relation S(B,C,D):

B	C	D
2	4	6
4	6	8
4	7	9

SQL query:

```sql
SELECT A, R.B, S.B, C, D
FROM R, S
WHERE R.A < S.C AND R.B < S.D
```

Then, identify one of the tuples in the result from the list below.

a) (5,6,2,4,6)
b) (3,4,5,7,9)
c) (3,4,4,7,8)
d) (1,2,2,4,6)

Answer: d) (1,2,2,4,6)

- *Explanation:* We are performing a theta-join with the condition ‘R.A ¡ S.C AND R.B ¡ S.D‘. Let’s systematically check each R tuple against each S tuple:
  From R: (1,2)
  - Against S: (B=2, C=4, D=6)
  - Condition 1: R.A ¡ S.C (1 ¡ 4)? True.
  - Condition 2: R.B ¡ S.D (2 ¡ 6)? True.
  - Both True. Result tuple: (A=1, R.B=2, S.B=2, C=4, D=6) $\rightarrow$ **(1,2,2,4,6)**. This matches option (d).
  - Against S: (B=4, C=6, D=8)
  - Condition 1: R.A ¡ S.C (1 ¡ 6)? True.
  - Condition 2: R.B ¡ S.D (2 ¡ 8)? True.
  - Both True. Result tuple: (1,2,4,6,8). (Not an option)
  - Against S: (B=4, C=7, D=9)
  - Condition 1: R.A ¡ S.C (1 ¡ 7)? True.
  - Condition 2: R.B ¡ S.D (2 ¡ 9)? True.
  - Both True. Result tuple: (1,2,4,7,9). (Not an option)

  Since we found a match, we can stop here. The tuple (1,2,2,4,6) is a valid result.

Question 6. Suppose R(A,B) has a tuple t: (A = 2, B = NULL). The following query outputs t. SQL query:

```
SELECT *
FROM R
WHERE (((A IS NULL) OR (B = 2)) AND (B IS NULL))
```

a) True
b) False

Answer: b) False

- *Explanation:* Let’s evaluate the ‘WHERE‘ clause for tuple t: (A=2, B=NULL). The condition is ‘(((A IS NULL) OR (B = 2)) AND (B IS NULL))‘
  1. ‘A IS NULL‘: Is 2 IS NULL? False. 2. ‘B = 2‘: Is NULL = 2? Unknown (in SQL, comparison with NULL yields Unknown). 3. ‘B IS NULL‘: Is NULL IS NULL? True.
  Now substitute these truth values into the overall expression: ‘((False OR Unknown) AND True)‘
  - ‘False OR Unknown‘ evaluates to ‘Unknown‘.
  - ‘Unknown AND True‘ evaluates to ‘Unknown‘.

  In SQL, a ‘WHERE‘ clause only returns rows where the condition evaluates to ‘TRUE‘. Since the condition evaluates to ‘UNKNOWN‘ (not ‘TRUE‘), the tuple t will ‘NOT‘ be output by this query. Therefore, the statement is False.

Question 7. Suppose the relation R(a,b,c) has the tuples: Table: Table R:

```
| a | b | c |
|---|---|---|
| 0 | 1 | 2 |
| 0 | 1 | 3 |
| 4 | 5 | 6 |
| 4 | 6 | 3 |
```

Compute the generalized projection $\pi_{B,A+C,B}(R)$, and then identify from the list below one of the tuples in this projection.

a) (10,5,10)
b) (1,1,3)
c) (1,3,0)
d) (6,7,6)

Answer: d) (6,7,6)

- *Explanation:* A generalized projection applies expressions to the attributes of each tuple and then projects the results. For each tuple (a, b, c) in R, we need to compute (b, a+c, b).
  1. Tuple (0, 1, 2):
     - b = 1
     - a+c = 0+2 = 2
     - b = 1
     - Result: (1, 2, 1)
  2. Tuple (0, 1, 3):
     - b = 1
     - a+c = 0+3 = 3
     - b = 1
     - Result: (1, 3, 1)
  3. Tuple (4, 5, 6):
     - b = 5
     - a+c = 4+6 = 10
     - b = 5
     - Result: (5, 10, 5)
  4. Tuple (4, 6, 3):
     - b = 6
     - a+c = 4+3 = 7
     - b = 6
     - Result: (6, 7, 6)

  The resulting bag of tuples from the projection is $\{(1,2,1),(1,3,1),(5,10,5),(6,7,6)\}$. From the given options, (6,7,6) is present in this result.

Question 8. TRUE/FALSE: A left outer join can always be written using a right outer join

a) False
b) True

Answer: b) True

- *Explanation:* Yes, this is true. A 'LEFT OUTER JOIN' (e.g., 'A LEFT OUTER JOIN B') keeps all records from the left table (A) and the matching records from the right table (B). If there's no match, NULLs are placed for B's columns. A 'RIGHT OUTER JOIN' (e.g., 'A RIGHT OUTER JOIN B') keeps all records from the right table (B) and matching records from the left table (A). Crucially, 'A LEFT OUTER JOIN B' is logically equivalent to 'B RIGHT OUTER JOIN A'. You just swap the tables around the 'RIGHT OUTER JOIN' keyword.

Question 9. Consider the following query on the Netflix database schema discussed in the lectures with Ratings relation alias R and Movies relation alias M. Which of the queries listed afterward is logically equivalent to this query? $\sigma_{\text{Year=2021} \wedge \text{Stars=5.0}}(R \bowtie M)$

a) $\sigma_{\text{Year=2021}}(R) \bowtie \sigma_{\text{Stars=5.0}}(M)$

b) $\sigma_{\text{Year}=2021 \wedge \text{Stars}=5.0}(R) \bowtie M$

c) $\sigma_{\text{Year}=2021 \wedge \text{Stars}=5.0}(M) \bowtie R$

d) $\sigma_{\text{Year}=2021}(M) \bowtie \sigma_{\text{Stars}=5.0}(R)$

Answer: d) $\sigma_{\textbf{Year}=2021}(M) \bowtie \sigma_{\textbf{Stars}=5.0}(R)$

- *Explanation:* This question tests the "pushdown" optimization rule in relational algebra. Selection (σ) operations can be pushed down before a join ($\bowtie$) if the selection condition only involves attributes from one of the relations being joined. The original query is $\sigma_{\text{Year}=2021 \wedge \text{Stars}=5.0}(R \bowtie M)$.
  - 'Year' is an attribute of 'Movies (M)'.
  - 'Stars' is an attribute of 'Ratings (R)'.

  Therefore, we can push down the 'Year = 2021' selection to 'M' and the 'Stars = 5.0' selection to 'R' *before* performing the join. This results in: $(\sigma_{\text{Year}=2021}(M)) \bowtie (\sigma_{\text{Stars}=5.0}(R))$ This is exactly option (d). This optimization is crucial for performance as it reduces the size of the relations before the potentially expensive join operation.

Question 10. A primary index is necessarily also the following type of index?

a) None of the three

b) Composite

c) Unique

d) Secondary

Answer: c) Unique

- *Explanation:*
  - A 'primary index' is an index built on the primary key of a table.
  - A 'primary key' by definition must contain unique values and uniquely identify each row in the table.
  - Therefore, an index built on a primary key (a primary index) must necessarily enforce 'uniqueness'.
  - It is not necessarily a 'composite' index (it can be on a single attribute).
  - It is also not necessarily a 'secondary' index; in fact, a primary index is often the main or clustering index that dictates the physical storage order of data.

Question 11. Consider the following schema:

```
Student (snum: integer, sname: string, major: string, level: string, age: integer)
Class (cname: string, meets_at: time, room: string, fid: integer)
Faculty (fid: integer, fname: string, depid: integer)
Enrolled (snum: integer, cname: string)
```

a) Write the SQL statements required to create all of the above relations, including all primary and foreign keys

```
Answer:
CREATE TABLE Student (
snum INT AS PRIMARY KEY,
sname VARCHAR(50),
major VARCHAR(50),
level VARCHAR(50),
age INTEGER)

CREATE TABLE Class (
fid INTEGER AS PRIMARY KEY
cname VARCHAR(50),
meets_at TIME,
room VARCHAR(50))
```

```
Create Table Faculty (
fid INT AS PRIMARY KEY,
fname VARCHAR(50),
depid INT)

CREATE TABLE Enrolled (
snum INT AS PRIMARY KEY,
cname VARCHAR(50))
```

Evaluation and Correction for 11a):

- **Student table:** ‘snum INT PRIMARY KEY‘ is the correct syntax. ‘AS‘ is not needed.
- **Class table:** The primary key should be ‘cname‘ (class name), not ‘fid‘. ‘fid‘ is a foreign key referencing ‘Faculty‘. Also, ‘fid‘ is a foreign key, not a primary key.
- **Faculty table:** Looks correct.
- **Enrolled table:** This is a many-to-many relationship, and its primary key should be a composite key of both ‘snum‘ and ‘cname‘. Both ‘snum‘ and ‘cname‘ are foreign keys.

  Corrected SQL for 11a):

```
CREATE TABLE Student (
    snum INTEGER PRIMARY KEY,
    sname VARCHAR(50),
    major VARCHAR(50),
    level VARCHAR(50),
    age INTEGER
);

CREATE TABLE Faculty (
    fid INTEGER PRIMARY KEY,
    fname VARCHAR(50),
    depid INTEGER
);

CREATE TABLE Class (
    cname VARCHAR(50) PRIMARY KEY,
    meets_at TIME,
    room VARCHAR(50),
    fid INTEGER,
    FOREIGN KEY (fid) REFERENCES Faculty(fid)
);

CREATE TABLE Enrolled (
    snum INTEGER,
    cname VARCHAR(50),
    PRIMARY KEY (snum, cname),
    FOREIGN KEY (snum) REFERENCES Student(snum),
    FOREIGN KEY (cname) REFERENCES Class(cname)
);
```

b) Write the SQL statements that finds the names of all senior students (Student.level = “Junior”) who have less than 25 years old (Student.age¡25) and enrolled in a class taught by Prof. Gupta (Faculty.fname= “Gupta”).

```
Answer:
SELECT DISTINCT s.name, s.level, s.age
FROM Student AS s, Faculty as f
WHERE s.level='Junior' AND s.age > 25 AND f.name ='Gupta'
```

Evaluation and Correction for 11b):

- The join conditions between ‘Student‘, ‘Enrolled‘, ‘Class‘, and ‘Faculty‘ are missing.

- The ‘age‘ condition is ‘s.age ¿ 25‘ in the answer, but the question asks for ‘Student.age ¡ 25‘.
- The question asks for ‘Student.level = ”Junior”‘, which is correctly used.
- The ‘DISTINCT‘ keyword is good as a student might be in multiple classes taught by Prof. Gupta. **Corrected SQL for 11b):**

```
SELECT DISTINCT S.sname
FROM Student AS S
JOIN Enrolled AS E ON S.snum = E.snum
JOIN Class AS C ON E.cname = C.cname
JOIN Faculty AS F ON C.fid = F.fid
WHERE S.level = 'Junior'
  AND S.age < 25
  AND F.fname = 'Gupta';
```

c) Write the SQL statements that finds the names of faculty members for whom the combined enrollment of the courses that they teach is less than three.

```
Answer:
SELECT DISTINCT f.name
FROM Faculty as f, Enrolled as e
WHERE SUM(e.snum) < 3
```

Evaluation and Correction for 11c):

- The ‘SUM(e.snum)‘ is incorrect; ‘snum‘ is a student ID, not an enrollment count. To count enrollment, you need to count distinct ‘snum‘ values for each class, then sum those counts per faculty. Or, simpler, ‘COUNT(E.snum)‘ for all enrollments associated with a faculty member.
- Missing joins between ‘Faculty‘, ‘Class‘, and ‘Enrolled‘.
- Missing ‘GROUP BY‘ clause for aggregation.
  Corrected SQL for 11c):

```
SELECT F.fname
FROM Faculty AS F
JOIN Class AS C ON F.fid = C.fid
JOIN Enrolled AS E ON C.cname = E.cname
GROUP BY F.fid, F.fname
HAVING COUNT(E.snum) < 3; -- COUNT(E.snum) counts the number of enrollments
                         -- if distinct students are required, use COUNT(
                             DISTINCT E.snum)
```

d) For each level, print the level and the average age of students for that level.

```
Answer:
SELECT AVERAGE(s.age) as avg_age, s.level
FROM STUDENTS as s
GROUP BY s.level
```

Evaluation and Correction for 11d):

- ‘AVERAGE‘ should be ‘AVG‘.
- Otherwise, this query is correct. **Corrected SQL for 11d):**

```
SELECT S.level, AVG(S.age) AS avg_age
FROM Student AS S
GROUP BY S.level;
```

e) Write the SQL statements that, for each faculty member that has taught classes only in room ’R2010’, prints the faculty member’s name and the total number of classes she or he has taught.

```
Answer:
SELECT DISTINCT f.name, COUNT(c.name)
FROM Faculty AS f, Classes AS c
JOIN f.fid on c.fid
WHERE c.name='R2010'
```

Evaluation and Correction for 11e):

- The join syntax ‘JOIN f.fid on c.fid‘ is incorrect. It should be ‘ON f.fid = c.fid‘.
- The ‘WHERE c.name=’R2010’‘ condition only selects classes in ’R2010’. To find faculty who *only* taught in ’R2010’, you need a more complex condition (e.g., using ‘NOT EXISTS‘ or ‘EXCEPT‘ or ‘GROUP BY‘ with ‘HAVING‘).
- ‘COUNT(c.name)‘ should be ‘COUNT(DISTINCT c.cname)‘ if counting unique classes.
- Missing ‘GROUP BY‘.

  Corrected SQL for 11e):

```
SELECT F.fname, COUNT(DISTINCT C.cname) AS num_classes
FROM Faculty AS F
JOIN Class AS C ON F.fid = C.fid
GROUP BY F.fid, F.fname
HAVING EVERY(C.room = 'R2010'); -- PostgreSQL/SQL:2003 "EVERY" or equivalent
    logic
                             -- For more portable SQL:
-- HAVING COUNT(CASE WHEN C.room = 'R2010' THEN 1 END) = COUNT(C.cname);
-- OR, using NOT EXISTS:
/*
SELECT F.fname, COUNT(C.cname) AS num_classes
FROM Faculty AS F
JOIN Class AS C ON F.fid = C.fid
WHERE NOT EXISTS (
    SELECT 1
    FROM Class AS C2
    WHERE C2.fid = F.fid AND C2.room != 'R2010'
)
GROUP BY F.fid, F.fname;
*/
```

f) Write the SQL statements that computes the average age of all students who are majoring in “Data Science” and enrolled in “DSC 80” and “DSC 100”

```
Answer:
SELECT s.major
FROM Students AS s, Classes AS c
WHERE c.name ='DSC80' AND c.name='DSC100' AND s.major='Data Science'
```

Evaluation and Correction for 11f):

- The query should ‘SELECT AVG(s.age)‘, not ‘s.major‘.
- The condition ‘c.name =’DSC80’ AND c.name=’DSC100’‘ is logically impossible for a single class name. A student needs to be enrolled in *both* classes. This requires either two separate join paths or using subqueries/‘HAVING‘ with ‘GROUP BY‘.
- Missing joins with ‘Enrolled‘.

  Corrected SQL for 11f):

```
SELECT AVG(S.age) AS average_age
FROM Student AS S
WHERE S.major = 'Data Science'
  AND S.snum IN (SELECT E.snum FROM Enrolled AS E WHERE E.cname = 'DSC 80')
  AND S.snum IN (SELECT E.snum FROM Enrolled AS E WHERE E.cname = 'DSC 100');
```

(Note: Assumes ”DSC 80” and ”DSC 100” are exact class names including the space).

g) Write the SQL statements that find the names of students not enrolled in any class.

```
Answer:
SELECT s.name
FROM Students as s, Enrolled as e
JOIN s.snum ON e.snum
WHERE e.snum =0
```

Evaluation and Correction for 11g):

- The join ‘JOIN s.snum ON e.snum‘ is incorrect syntax. It should be ‘ON S.snum = E.snum‘.
- ‘WHERE e.snum = 0‘ is an incorrect way to find students not enrolled. It implies there’s an enrollment record with ‘snum=0‘ for non-enrolled students, which is not standard.
- The correct way to find non-enrolled students is using ‘LEFT JOIN‘ and checking for ‘NULL‘ in the joined table, or using ‘NOT EXISTS‘ or ‘NOT IN‘.

  Corrected SQL for 11g):

  ```
  SELECT S.sname
  FROM Student AS S
  LEFT JOIN Enrolled AS E ON S.snum = E.snum
  WHERE E.snum IS NULL;
  ```

  (Alternatively, using ‘NOT EXISTS‘ or ‘NOT IN‘ is also valid).

h) Find the names of students that are either junior (Student.level = “Junior”) or enrolled in at most 2 courses.

```
Answer:
SELECT s.level
FROM Students AS s, Enrolled as e
JOIN s.snum ON e.sum
WHERE s.level='Junior' AND e.snum<=2
```

Evaluation and Correction for 11h):

- The query should ‘SELECT s.sname‘, not ‘s.level‘.
- The join condition is again syntactically incorrect (‘ON e.sum‘ should be ‘ON S.snum = E.snum‘).
- The condition ‘e.snum ¡= 2‘ is nonsensical for counting courses; ‘e.snum‘ is a student ID. You need to count the number of courses per student and then filter.
- The ‘AND‘ should be an ‘OR‘ as the question states ”either... or...”.

  Corrected SQL for 11h):

  ```
  SELECT DISTINCT S.sname
  FROM Student AS S
  WHERE S.level = 'Junior'
     OR S.snum IN (
        SELECT E.snum
        FROM Enrolled AS E
        GROUP BY E.snum
        HAVING COUNT(E.cname) <= 2
     );
  ```

Question 12. Consider the following schema: Supplier (<u>sid: integer</u>, sname: string, city: string , state: string) Part (<u>pid: integer</u>, pname: string, size: string, color: string) Supply (<u>sid: integer</u>, <u>pid: integer</u>, cost: real)

The keys are underlined, and the domain of each attribute is listed after the attribute name. The “Supply” relation consists of the prices for parts supplied by the Suppliers. ($\rho_R(E)$ renames the result of expression E as R) Write the SQL queries corresponding to the following RA expressions:

a)

$$\pi_{\textbf{sid}}(\pi_{\textbf{pid}}(\sigma_{\textbf{city='La Jolla'} \wedge \textbf{state='CA'}}\textbf{Supplier})) \bowtie \sigma_{\textbf{cost}>100}\textbf{Supply}$$

```
Answer:
SELECT DISTINCT Supply.sdi
FROM Supply, Supplier
WHERE Supplier.sdi =Supply.sdi
AND Supplier.city ='La Jolla'
AND Supplier.state ='CA'
AND Supply.cost >100
```

Evaluation and Correction for 12a):

- The 'WHERE Supplier.sdi =Supply.sdi' is a typo and should be 'Supplier.sid = Supply.sid'.
- The 'FROM Supply, Supplier' implies a cross-join, and the 'WHERE' clause acts as an implicit join condition. This is acceptable, but explicit 'JOIN' syntax is preferred.
- The relational algebra expression suggests a slightly different flow: 1. Select Suppliers in La Jolla, CA. 2. Project 'sid' from those suppliers. 3. Select 'Supply' records where 'cost ¿ 100'. 4. Join the results of steps 2 and 3 on 'sid'. 5. Project 'sid' from the final join. Your SQL correctly captures the join and selection logic to get the 'sid's that match both conditions. The structure of the RA expression with an intermediate projection on 'pid' then 'sid' on the 'Supplier' side is a bit unusual if only 'sid' is ultimately needed for the join. However, the SQL achieves the correct final set of SIDs.

  Corrected SQL for 12a) (Minor fix and explicit join):

```
SELECT DISTINCT S.sid
FROM Supplier AS S
JOIN Supply AS Su ON S.sid = Su.sid
WHERE S.city = 'La Jolla'
  AND S.state = 'CA'
  AND Su.cost > 100;
```

b)

$$\pi_{\textbf{city}}(\sigma_{\textbf{c}>1000}(\gamma_{\textbf{city,count(*)}\ \textbf{c}}(\textbf{Part} \bowtie (\sigma_{\textbf{cost}<100}\textbf{Supply}) \bowtie (\sigma_{\textbf{state='CA'}}\textbf{Supplier})))$$

```
Answer:
SELECT Supplier.city
FROM Part as p
JOIN Supply ON p.pid = Supply.pid
JOIN Supplier ON supply.sid =Supplier.sid
WHERE Supply.cost < 100
AND Supplier.state= 'CA'
AND COUNT(Supplier.city)> 1000
```

Evaluation and Correction for 12b):

- The relational algebra expression uses $\gamma_{\text{city,count(*)}\ \text{c}}$ which means 'GROUP BY city' and then count how many tuples (after the joins and selections) fall into each city. The 'c ¿ 1000' is then a 'HAVING' clause.
- The provided SQL puts 'COUNT(Supplier.city)¿ 1000' in the 'WHERE' clause, which is a syntax error because aggregate functions cannot be used directly in 'WHERE'. It needs a 'GROUP BY' and 'HAVING'.
- The final 'SELECT Supplier.city' needs to be part of the 'GROUP BY' clause.

  Corrected SQL for 12b):

```
SELECT S.city
FROM Part AS P
JOIN Supply AS Su ON P.pid = Su.pid
JOIN Supplier AS S ON Su.sid = S.sid
WHERE Su.cost < 100
  AND S.state = 'CA'
GROUP BY S.city
HAVING COUNT(*) > 1000;
```

c)

$$\rho_{R1}(\pi_{\textbf{sid}}((\pi_{\textbf{pid}}(\sigma_{\textbf{size}>200}\textbf{Part})) \bowtie \textbf{Supply}))$$

$$\rho_{R2}(\pi_{\textbf{sid}}(\sigma_{\textbf{state='WA'}\vee\textbf{state='CA'}}\textbf{Supplier}))$$

$$R1 \cup R2$$

```
Answer:
SELECT DISTINCT Supply.sdi
FROM Part AS p
JOIN Supply ON p.pid=Supply.pid
WHERE p.size > 100
UNION
SELECT sid
FROM Supplier AS s
WHERE s.state= 'WA' OR s.state= 'CA'
```

Evaluation and Correction for 12c):

- In the first part, ‘p.size ¿ 100‘ is used in SQL, but RA has ‘size ¿ 200‘. Let’s assume the RA is the source of truth, so it should be ‘¿ 200‘.
- The ‘Supply.sdi‘ typo should be ‘Supply.sid‘.
- The ‘UNION‘ operator is correct for relational algebra union. ‘DISTINCT‘ is implied by ‘UNION‘ in SQL, but ‘UNION ALL‘ would correspond to bag union. Given the RA ‘union‘ symbol, ‘UNION‘ (which removes duplicates) is appropriate.
- The ‘SELECT sid FROM Supplier AS s‘ is correct.

  Corrected SQL for 12c):

  ```
  SELECT DISTINCT Su.sid
  FROM Part AS P
  JOIN Supply AS Su ON P.pid = Su.pid
  WHERE P.size > 200 -- Corrected based on RA expression
  UNION
  SELECT S.sid
  FROM Supplier AS S
  WHERE S.state = 'WA' OR S.state = 'CA';
  ```

Question 13. Consider the Netflix database schema discussed in the lectures: Schema: R (RatingID, Stars, RateDate, UID, MID) U (UID, Name, Age, JoinDate) M (MID, Name, Year, Director)

For each query given below, write a CREATE INDEX statement that can help speed up that query. Make sure to clearly mention the index type and SearchKey.

a) SELECT Director FROM M WHERE Year ¿= 2022;

```
Answer:
CREATE INDEX index_movie_year ON M USING BTREE (Year)
```

Evaluation for 13a):

- Correct. A B+ tree index is excellent for range queries (‘¿=‘). The search key is ‘Year‘.

b) SELECT Name FROM U WHERE Age = 18;

```
Answer:
CREATE INDEX index_user_age_hash ON U USING HASH (Age)
```

Evaluation for 13b):

- Correct. A hash index is highly efficient for equality lookups (‘=‘). The search key is ‘Age‘. A B+ tree would also work, but hash is typically faster for exact matches.

c) SELECT * FROM R, U WHERE R.UID = U.UID;

```
Answer:
CREATE INDEX index_rating_uid_hash ON R USING HASH (UID);
```

Evaluation for 13c):

- Partially correct. An index on ‘R.UID‘ (the foreign key) is good. An index on ‘U.UID‘ (the primary key) is also crucial. For join conditions, it’s generally beneficial to index both sides of the join. Either hash or B-tree would work. **Improved answer for 13c):**

```
        CREATE INDEX index_rating_uid_hash ON R USING HASH (UID);
        -- AND
        CREATE INDEX index_user_uid_hash ON U USING HASH (UID);
```

d) SELECT * FROM R WHERE RateDate ¿= 01/01/2021;

```
Answer:
CREATE INDEX index_rating_date ON R USING BTREE (RateDate)
\end{lstlisting>
\textbf{Evaluation for 13d):}
\begin{itemize}
   \item Correct. A B+ tree index is ideal for range queries on dates. The search key
       is RateDate.
\end{itemize}

\item Given the following schema:
    schema:
    Product($\underline{\text{ProductID}}$, Brand, Type, Price)
    Orders($\underline{\text{OrderID}}$, ProductID, OrderDate, Amount)

    The primary key columns are underlined, and the foreign key ProductID in the Product
        table references the ProductID column in the Product table.
    Write a SQL query that calculates the total sales for each brand and type including
        brands and types that have no sales, and orders for which the product information
        is missing, i.e., their product ID is NULL. The query must be executable on SQLite
        , which means you can only use SQL constructs that are supported by SQLite
    \begin{lstlisting}
    Answer:
    SELECT outer_q.Brand, outer_q.Type, SUM(outer_q.sales) AS total_sales
    FROM (
    /* 1. Products (LEFT JOIN) -> keeps rows with zero sales */
        SELECT p.Brand AS Brand,
               p.Type AS Type,
               COALESCE(p.Price * o.Amount, 0) AS sales
         FROM Product AS p
         LEFT JOIN Orders AS o
               ON o.ProductID = p.ProductID

        UNION ALL
    /* 2. Orders whose ProductID IS NULL -> treat as "unknown" */
        SELECT '(unknown)' AS Brand, -- label of your choice
               '(unknown)' AS Type,
                SUM(o2.Amount) AS sales -- no price info
         FROM Orders AS o2
        WHERE o2.ProductID IS NULL
        GROUP BY Brand, Type -- collapses to one row
        ) AS outer_q
    GROUP BY outer_q.Brand, outer_q.Type
    ORDER BY outer_q.Brand, outer_q.Type;
```

Evaluation for 14):

- This is a very well-structured and complex query that correctly addresses all parts of the requirement:
  - ‘LEFT JOIN Product‘ with ‘Orders‘ correctly includes products with no sales (by using ‘COALESCE(p.Price * o.Amount, 0)‘ to handle NULL sales as 0).
  - The ‘UNION ALL‘ with the second subquery ‘(SELECT ... WHERE o2.ProductID IS NULL)‘ correctly handles orders with missing product information, labeling them as ’(unknown)’ and summing their amounts (as price info is missing).
  - The outer ‘GROUP BY Brand, Type‘ aggregates the sales for all categories, including the ’unknown’ one.
  - ‘ORDER BY Brand, Type‘ provides a structured output.

    - The use of ‘COALESCE‘ and ‘UNION ALL‘ is standard SQL and supported by SQLite.

- Minor point: The second subquery has ‘GROUP BY Brand, Type‘ but selects ‘’(unknown)’ AS Brand, ’(unknown)’ AS Type‘, so it will always collapse to a single row for ’(unknown)’ brand and type. This is functionally correct for the goal.

  Overall Assessment for 14): This is an excellent solution to a challenging SQL problem, demonstrating a strong grasp of ‘JOIN‘ types, aggregation, ‘UNION‘, and handling ‘NULL‘ values.