Final Exam: Written Questions

W4111 – Introduction to Databases: 20231COMS4111W002

# Foundations

## F1 – Benefits of DBMS (q1)

Before DBMS, organizations managed data by writing application programs that manipulated data in files. List 5 problems with the application-file approach and succinctly state of DBMS solve the problem.

* File Size: With the application-file approach, the size of the file may grow to be too large, which causes problems with storage, reading/writing, etc. DBMS solve this issue by storing files in efficient formats & optimized query methods.
* Security: With static files, it is difficult to manage authorization and issues like read/write access—especially with many people interacting with the data. DBMS administrators have control over access & security, as well as the ability to manage/validate data with built-in systems in DBMS.
* Concurrent Access: If too many people try to access a file (e.g. a csv file) at the same time, processes like reads/writes or even loading the file may become extremely slow and cumbersome to manage. With DBMS, optimized processes allow for consistent and smooth operations, even with many users.
* Data Inconsistency: If the same data is stored in multiple files (e.g. client phone numbers), changes in one file may not be reflected in others, and it is difficult to manually ensure consistent changes. DBMS solves this issue by maintaining a central database, with features like foreign key constraints.
* Data Integrity: In an application-file approach, there are limited ways to ensure validity in data (e.g. birth dates cannot be negative). DBMS solves this issue with built-in validity mechanisms such as check constraints, unique constraints, etc.

## F2 – Types of Data (q2)

Briefly explain structured data, semi-structured data and unstructured data. Give an example of each type of data.

* Structured Data: Data that is formatted and organized into a standardized structure (e.g. tabular data). There is a well-defined schema and model which can be identified, and data can often be stored in relational databases. Examples are financial data (daily asset prices), or transactions data in a company.
* Unstructured Data: Data that has no pre-defined structure or model. This type of data typically cannot be stored in typical relational databases. Examples are social media posts, emails, videos, etc.
* Semi-structured Data: Data that has some structure, with some organization (e.g. metadata or tags/labels) but no rigidly defined schema. Examples are JSON/XML files.

## F3 – Physical Data Independence (q3)

What is physical data independence? What is a benefit?

In DBMS, there are multiple levels in the schema. The physical level is concerned with physical storage matters. The logical level is concerned with the conceptual organization of the database, with constraints, definitions, etc. Physicla data independence refers to the idea that we can change the physical level without affecting the logical level. For instance, we can replace hard-disks with SSD, without affecting logical organization or applications that depend on the data. Benefits include greater flexibility, scalability, and opportunities for optimization.

Source: https://www.tutorialspoint.com/dbms/dbms\_data\_independence.htm

## F4 – Concepts (q4)

Explain the following concepts and give an example of each:

* Data manipulation language (DML): this is programming language used to manipulate data inside a database. This can be used for tasks like selects, inserts, updates, deletes, etc. An example is:

SELECT \* FROM Customers

* Data definition language (DDL): this is used to define the schema and structure of a database. It is used for tasks like creating, modifying, and updating databases. An example is:

CREATE TABLE Customers id varchar(5)

* Procedure DML: procedural languages—including DDL—specify what result to return, as well as what exact steps to take. An example is using Pandas to handle data in Python

df["Profit"] = df["Sales"] \* 0.2

* Declarative DML: declarative languages specify what end result to return without specifying how to achieve the result. An example is SQL statements

SELECT \* FROM Customers WHERE Age > 34

Source: https://www.mn.uio.no/ifi/studier/masteroppgaver/asr/declarative-vs-procedural-data-manipulation-langua.html

## F5 – Modeling (q5)

Briefly explain the following concepts and the role for each concept in data modeling. Give an example of the benefit of each level:

* conceptual model: determines the entity names and relationships in the schema. The purpose of the conceptual model is to organize basic concepts and define relationships, and this step is usually carried out with business stakeholders.
* logical model: determines attributes and key constraints. The purpose of this step is to determine how to implement the conceptual model in a database, and develop a concrete structure.
* physical model: determines table names, column names, data types. This step is concerned with implementing a schema design into a physical database (e.g. in MySQL, mongoDB, etc.)

Source: Midterm

## F6 – Application Architectures (q6)

Briefly explain:

* Two-tier database application architecture

Also known as a client-server architecture, there are 2 tiers in this model. The client layer represents the user interface, and communicates with the server layer to access/manipulate data. The server layer stores and manages the data. It receives requests from the client layer, processes them, and sends them back to the client.

* Three-tier application architecture

Three-tier applications refer to things like Jupyter Notebook. There is a client layer (e.g. internet browser) which provides an interface for the user to interact with the server by writing query code to it. There is a server layer which processes the queries written by the user and sends it to the connected database. The data/storage layer, (e.g. MySQL database) then runs the query and returns the results to the server, which displays the results through the browser.

## F7 – Database Administrators (DBA) (q7)

List 5 tasks/functions that a DBA performs. Do DBAs typically use DDLs or DMLs?

* Database Design: A DBA may work with business partners to design database schema and management solutions.
* Database Availability: A DBA must ensure that users are able to access the database easily and efficiently.
* Database Security: A DBA manages permissions and access privileges for different users.
* Data Moves: A DBA may oversee database storage, e.g. moving from a physical database to a cloud database.
* Data Backup: A DBA must ensure that data is periodically backed up, such that data can be restored when crashes occur, for instance.

Source: https://www.educative.io/answers/what-are-the-roles-of-a-database-administrator

# Relational Model

## R1 – Domain (q8)

Explain the importance of atomicity of domains. *float* is a type. An example of a *domain* might be a person’s *weight*. The type for *weight* might be a float, but give an example of how *float* is not the *domain.*

Atomicity is an important property in domains. By ensuring that a domain value cannot be broken down into smaller pieces, we can ensure consistency in a database, with no partial/missing values. If we don’t break down a name into First, Last, Middle, for instance, we cannot treat “John F. Kennedy” the same as “Metallica,” because they have different properties.

In the given example, the type for *weight* is not its domain—the domain is a subset of the type. Specifically, negative values such as “-54.23” are not included in the domain of *weight*, but is included within the type *float.*

## R2 – Keys (q9)

Briefly define and explain the following concepts:

* super key: a key is a superkey if its values can identify a unique row in a relation
  + e.g. {ID}, {ID, name} are superkeys in *Instructor*
* candidate key: a candidate key is a minimal super key (i.e. we need every attribute in it to form a super key).
  + e.g. {course\_id} in *Course*
* primary key: one can select a candidate key to be a primary key
  + e.g. {ID} in *Student*
* foreign key: a key is a foreign key if a value in one relation appears in another relation
  + e.g. {dept\_name} in *Student* references *Department*

## R3 – Operators (q10)

The slides associated with the recommended text book list six basic relational operators

* select: 𝞼
* project: 𝝅
* union: U
* set difference: –
* Cartesian product: x
* rename: ⍴

Surprisingly, the list does not include *join: ⋈.* This is because join it is possible to derive join from a relational expression using more basic operators.

Briefly explain how to derive join from basic operators.

We can derive a join using select and cartesian product. A theta join of 2 tables is the cartesian product of the 2 followed by a select on the theta condition. Formally:

What is the importance of the relational algebra being *closed under the operators* for the derivation.

“Closed under the operators” refers to the fact that an operation on two relations also produces at its output another relation. This is important, because the output of the cartesian product is another relation, on which we can perform a select.

## R4 – Equivalent Queries (q11)

Briefly explain the concept of equivalent queries. Later lectures explained an important use of the concept. What is that use?

Equivalent queries are different queries that return the same result table. An important use mentioned in later lecture has to do with query processing. In the query optimization step, optimizers generate several equivalent queries from the input, and select the one with the lowest cost. This allows the optimizer to choose the best query execution plan.

# SQL

## S1 – Foundation (q12)

Codd’s Rule 0 states

**Rule 0:** The *foundation rule*:

For any system that is advertised as, or claimed to be, a relational database management system, that system must be able to manage databases entirely through its relational capabilities.

Briefly explain and give examples of how the rule applies to:

* Metadata: In SQL RDBMS, metadata is also stored in relational format. Tables such as INFORMATION\_SCHEMA contain metadata about different database objects and can be accessed through typical DML queries.
* Security: In SQL RDBMS, security is implemented through specific sets of permission for different users or groups. This can be managed with GRANT and REVOKE statements. We can use typical DML commands like UPDATE, DELETE to change these permissions, and SELECT/JOIN/WHERE etc. to specify groups and users.

## S2 – NULL (q13)

Codd’s Rule 3 states

**Rule 3:** *Systematic treatment of null values*:

Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.

Briefly explain the importance of the rule for:

1. Using different database schemas defined by multiple people.

When there are multiple people defining different schema, they may define null values differently (e.g. “N/A”, “na”, “0”, “None”, “FALSE”). This creates inconsistency, which leads to difficulty in filtering null values, merging tables, etc—especially with different types.

1. SQL aggregation (group by) queries.

With aggregation queries, aggregation functions such as SUM, AVERAGE, MAXIMUM, etc. may need to handle null values differently based on the scenario—e.g. drop null values, treat them as 0, etc. To ensure this consistency, null values must be treated systematically, in order to create accurate aggregations even with missing data.

## S3 – Atomic Domains (q14)

The Columbia University directory of courses uses 20231COMS4111W002 for this section’s “key.” This is not atomic and is composed of:

* Year: 2023
* Semester: 1
* Department: COMS
* Course number: 4111
* Faculty code: W

Explain why having the non-atomic key creates problems for:

* Integrity constraints: Consider a foreign key, for instance. Referential integrity is at greater risk with composite keys, because a change in the key means every component must be propagated across referenced tables. With multiple composite keys involved in integrity constraints, this becomes a problem, as there are more opportunities for inconsistencies to arise from mistakes.
* Indexes: Indices on columns are often used to speed up queries. Often, this is created on important keys such as primary keys. With a non-atomic primary key, we need to create an index on every column of the composite key (5 columns in the example case). This demands more storage requirements and leads to suboptimal performance.

Source: https://www.wisdomjobs.com/e-university/firebird-tutorial-210/integrity-constraints-7628.html

## S4 – JOINs (q15)

Briefly explain the following concepts:

* Natural join: join tables where the 2 tables have the same values for columns with the same names
* Equi-join: explicitly specify columns that must have the same values (e.g. A.column\_x = B.column\_y)
* Theta join: join tables on an arbitrary predicate (e.g. A.column\_x > B.column\_y)
* Left join: join tables on a theta condition, but keep all tuples from the left table with matching rows from the right.
* Right join: join tables on a theta condition, but keep all tuples from the right table with matching rows from the left.
* Outer join: An outer join includes all rows from one or both tables, even if there are no matching rows in the other table. Left and Right joins are 2 types of outer joins.

## S5 – Natural Join (q16)

Briefly explain how using the natural join might produce an incorrect answer.

A natural join performs an equi-join on all columns of the same name and type. This creates problems when columns in two tables have the same name, but represent different things. Consider the “Credits” column in tables Student and Instructor. In the Student table, the column may refer to the number of credits taken by the student, while in instructor it refers to the number of credits taught by the instructor. Thus, a natural join would join on this column, which may not provide the desired information.

## S6 – Views (q17)

List three benefits/use cases for defining views.

* Security: Views can ensure different users can access different parts of a table. For instance, if we have a one-table implementation with sensitive data (e.g. salaries), we can create a view that doesn't include the salary column and use it for public applications.
* Simplification: We can create views for commonly used queries (e.g. counts of unique values by column). By storing these in views, we can simplify processes for naive users.
* Schema changes: If we have significant schema changes in the table with several applications built on it, we can create a view from the new schema that is compatible with the old version rather than rewrite all the applications built on it.

Source: Midterm

## S7 – Materialized View (q18)

What is a *materialized view?* List one advantage and disadvantage of a materialized view.

A materialized view is a view that is computed and stored in a cache. One advantage is that this improves performance by reducing the time to access a commonly-requested view, as the view does not have to be reconstructed each time. The disadvantage is that the view may become stale—if there are changes in the underlying table, a materialized view does not reflect the change until refreshed. This can lead to inconsistencies.

## S8 – View Updates (q19)

Explain two scenarios in view definition for which it is not possible to update the underlying tables?

If the view includes aggregation such as SUM, AVG, COUNT, etc. we cannot update the underlying tables, as the view is based on groups of rows rather than a single row. Therefore, we cannot know which row to update. Another problem is when the view includes JOINs. If the view contains columns from multiple tables that are not part of the same primary key, we cannot determine which underlying table should be updated if a change is made to a column in the view that is not part of the primary key.

## S9 – Primary/Unique (q20)

What is the main difference between a primary key constraint and a unique constraint?

Primary keys must uniquely identify each row in a relation; thus, they cannot have null values.

We can think of unique keys as the non-primary key candidate keys. They also uniquely identify each row, but they also allow for one NULL value, unlike a primary key. Also, a table can have multiple unique key constraints, but only one primary key constraint,

## S10 – Cascade (q21)

The *Classic Models* databases have several foreign key constraints. Two examples are:

1. *orders.customerNumber → customers.customerNumber*
2. *orderdetails.orderNumber → orders.orderNumber*

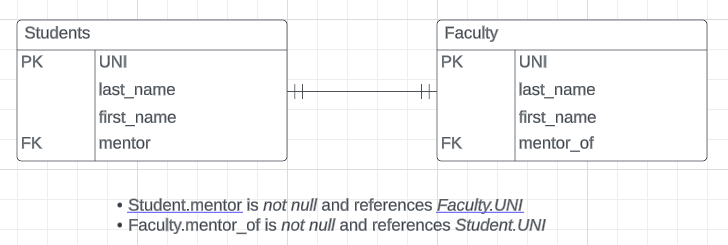
Briefly explain the concept of *cascading actions* relative to foreign keys. For which of the two examples above might cascading make sense?

In the context of foreign keys, cascading actions refer to automatic changes in related tables after a change in the referenced table (e.g. UPDATE, DELETE, INSERT).

In the example above, cascading makes sense for the first example. If we delete a customer from Customers, it makes sense to delete all orders from that customer in Orders. For the second one, cascading does not make sense. Order numbers typically do not change, as they reflect transactions that have already occurred.

## S11 – Foreign Keys and Transactions (q22)

Consider the logical data model below.



Some DBMS support deferring enforcing foreign key constraints until transaction commit. How would that capability help with the above model?

When foreign key constraints are enforced only after transaction commit, we conduct constraint checks until after the full transaction instead of after every INSERT or UPDATE. In the above model, the two tables reference one another. Thus, if we attempt to CREATE or INSERT in one first, foreign key constraints fail by definition, as the corresponding entry does not exist yet in the other table. By waiting until after the full transaction (e.g. INSERTs in both tables), we can ensure FK constraints are checked after both tables are—theoretically—implemented.

## S12 – Complex Check Constraints (q23)

Some databases do not support complex check constraints. Consider the following constraint:

check (time\_slot\_id in (select time\_slot\_id from time\_slot))

Assume the DBMS does not support subqueries in check constraints. What database capability would you use to implement equivalent functionality?

We can use triggers instead. Triggers are executed automatically after specific database events, such as INSERTs or DELETEs. Thus, we can define a trigger for INSERTs on this table to check the subquery result for every INSERT, and raise an error if the check is not satisfied.

## S13 – Asset (q24)

What is the difference between an *Assert* constraint and a *Check* constraint?

The main difference is that Assert constraints are applied to the whole database, while Check constraints are applied to specific tables. This implies Assert constraints can be more complex, specifying conditions involving multiple tables.

Source: https://stackoverflow.com/questions/2443322/what-is-the-difference-between-triggers-assertions-and-checks-in-database

## S14 – Types, Domains (q25)

Some relational DBMS support *user defined types* and *user defined domains.* Briefly explain the concepts and benefits.

A user defined type uses existing data types with certain restrictions in order to create a custom type. For instance, we can define a type Dollars as numeric(12,2). User defined domains work similarly. For domains, we can add additional constraints, such as *not null*, as well as default values. A benefit of using a user defined type or domain is that we can create more consistency. For instance, by setting type Dollars as numeric(12,2), we ensure that Dollar type values have 2 decimal points, which is consistent with application logic using dollars. For domains, we can also ensure that values outside a column’s domain are restricted. By including a check constraint in the user defined domain, we can check that every entry is a valid value.

## S15 – SQL Injection (q26)

Poorly written web applications can suffer from SQL Injection (Attacks). Briefly explain the concept.

SQL Injection attacks occur when we ask users for an input (e.g. userID), but the user inputs a SQL statement that is run on the database. For instance, consider the following code:

txtUserId = getRequestString("UserId");  
txtSQL = "SELECT \* FROM Users WHERE UserId = " + txtUserId;

The intent of the code is for users to input their UserID, such that they can see their own information. If the user inputs UserID= “1045 OR 1=1,” the second expression 1=1 always evaluates to TRUE, and the entire table is returned, exposing others’ information as well.

Source: https://www.w3schools.com/sql/sql\_injection.asp

## S16 – Functions, Procedures, Triggers (q27)

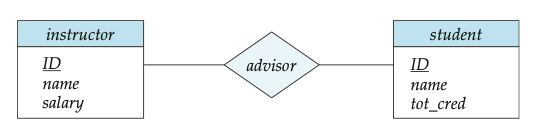
Briefly lists two differences between:

* Functions and Procedures
  + Functions return results, while procedures stores the result in the **out** parameter
  + A function can be called in a procedure, but not the other way around
* Functions and Triggers
  + Functions are called explicitly, while triggers are executed automatically in response to an action
  + Functions return a result, while triggers cannot return values on execution
* Triggers and Procedures
  + Procedures can be executed manually, while triggers are executed automatically
  + We cannot pass parameters to triggers, but we can do so for procedures.

# Entity – Relationship Modeling

## E1 – Implementing Relationships (q28)

The book’s entity-relationship modeling notation explicitly represents relationships. For example,



Crow’s Foot notation, which we used in class examples, does not support relationships. What type of entity did we use instead? Give two examples/reasons that require using the entity type.

We use an associative entity instead. More specifically, in a relational model, one way to model relationships is to create a relation for the relationship itself.

One reason this type of entity might be needed is many-to-many relationships. Consider books and authors, for instance. One book may have many authors, while one author may have many books—thus, we cannot use a foreign key to represent the relationships. We can thus create an associative entity that contains this information.

Another reason is that the relationship may have many attributes. The relationship between student and instructor (above) may have a category (e.g. major-advisor, minor-advisor, thesis-advisor, etc.). To represent this, we use an associative entity.

## E2 – Types of Relationships (q29)

Briefly explain the following concepts:

* Binary and Non-Binary Relationships
  + This refers to the association between entities. A relationship between two entities, e.g. Student and Courses, is a binary relationship. On the other hand, a relationship with multiple entities is considered non-binary—an example is the relationship between Patients, Doctors, and Clinics.
* Relationship Cardinality
  + Cardinality refers to the number of entities in the other entity set that a given entity can be associated to. There are four types for binary relationships (Examples are Instructor-Student):

One-to-one: Students have one advising instructor, Instructors have one advisee

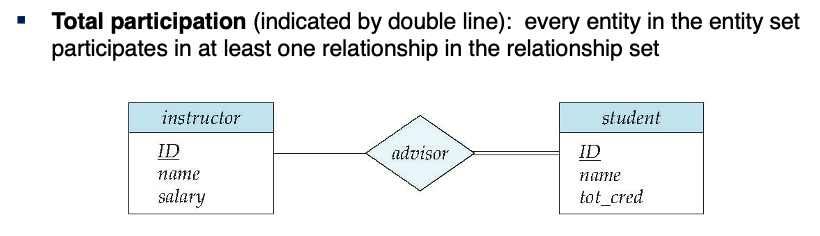
Many-to-one: Students can have many advisors, Instructors have one advisee

One-to-many: Students have one advisor, Instructors can advise many Students

Many-to-many: Students have many advisors, Instructors have many advisees

## E3 – Participation (q30)

An important concept in ER modeling is *relationship participation.*



Use Lucidchart to draw an equivalent diagram in Crow’s Foot notation. What capability of SQL database definition would you use to enforce total participation?

A picture containing text, font, screenshot, line

Description automatically generated

A simple way to enforce total participation is to implement the above through a foreign key. We can include a foreign key in student that references instructor, or the advisor table. By specifying *not null* in the DDL, we can ensure that every student has at least one associated advisor.

## E4 – Weak Entity (q31)

Briefly explain the concept of a *weak entity.* Give an example from the Classic Models database.

A weak entity is an entity whose existence depends on another entity set called the identifying set. The primary key of the identifying entity with additional discriminator attributes uniquely identify weak entities. An example from Classic Models is Orders and OrderDetails.

## E5 – Specialization (q32)

Briefly explain the following concepts relative to implementing inheritance/specialization in an SQL schema.

* incomplete/complete
  + Complete specialization means every higher-level entity belongs to a lower-level entity set, while incomplete specialization means that may not be the case. For instance, if there are 2 lower-level tables Student and Instructor, and the higher People table does not include any other type of Person, it is complete.
* Disjoint/overlapping
  + Overlapping refers to the fact that an entity can belong to more than one specialized entity sets, while disjoint implies that an entity can belong to only one. For instance, from the above example, in an overlapping specialization a Person can be both a Student and an Instructor, while in disjoint a Person must be one or the other.

# Normalization

## N1 – Duplicate/Redundant Data (q33)

A primary reason for schema normalization is to eliminate duplicate/redundant data. What are two problems that redundant/duplicate data can cause?

* Data Inconsistency: When the same data is stored in multiple locations in a database, it becomes challenging to maintain consistency. Any update to one copy of the data may not be reflected in other copies, leading to data inconsistencies and reliability issues.
* Increased Storage Costs: Storing redundant data requires more disk space, which increases storage costs. In addition, maintaining and updating redundant data requires more time and effort, which increases maintenance costs.

## N2 – Decomposition (q34)

Briefly explain the concept of *lossless decomposition* in normalization.

Lossless decomposition refers to the ability to break down a table into smaller tables without losing any information. This can be done by identifying functional dependencies between attributes and creating separate tables for each set of attributes that depend on the same subset of the primary key. Practically speaking, this allows us to reconstruct the original table from the smaller tables through foreign keys without losing information.

## N3 – Functional Dependency (q35)

Briefly explain the following concepts:

* Functional Dependency: Simply put, there is FD between attributes A and B if the values in A determines the values in B. For instance, all columns in a table are functionally dependent on the primary key (by definition).
* Closure of Functional Dependencies: The closure of functional dependencies is a process of computing all the functional dependencies that exist in a database table. Specifically, an FD *f* is implied by a set of FD’s *F* if *f* holds whenever *F* holds. The closure of *F* (*F\**) refers to the set of all FD’s implied by *F.*

## N4 – BCNF (q36)

Consider the sample university database that comes with the recommended textbook.

Consider a hypothetical relation:

*in\_dep (ID, name, salary, dept\_name, building, budget )*

Why is the relation not in BCNF?

In this schema, the FD dept\_name -> budgets holds. However, dept\_name is not a superkey, because a department may have several different instructors. Thus, the relation is not in BCNF.

## N5 – Third Normal Form (q37)

Briefly explain the difference between BCNF and 3rd Normal Form.

3rd Normal Form is essentially a relaxed version of BCNF (all BCNF is in 3NF, but not the other way around). 3NF is satisfied if BCNF is satisfied OR, if for all α → *β* in *F*+, each attribute *A* in *β* – α is contained in a candidate key for *R.* The idea behind this is to ensure dependency preservation.

## N6 – Armstrong’s Axioms (q38)

Briefly list Armstrong’s Axioms for Functional Dependencies.

1. Reflexivity: If X is a set of attributes, then X -> X. This means that any set of attributes is functionally dependent on itself.
2. Augmentation: If X -> Y, then XZ -> YZ for any set of attributes Z. This means that adding the same set of attributes to both sides of a functional dependency does not change the dependency.
3. Transitivity: If X -> Y and Y -> Z, then X -> Z. This means that if two functional dependencies hold, and the dependent attribute of one is the determinant of the other, then there is a transitive functional dependency between the determinants and dependents.

# Big Data

## B1 – MapReduce (q39)

Briefly define the following concepts in MapReduce:

1. Map
2. Reduce
3. Shuffle

MapReduce is a parallel processing model to process and do operations on large amounts of data.

* Map: The map() step outputs a key-value pair for every record, which is later passed into reduce() for aggregation.
* Reduce: The reduce() step gathers the outputs from map() then aggregates them into a full result.
* Shuffle: In a parallel system, data for different reduce keys must be available across machines, so that reduce() on one machine has access to all values for a given key. The shuffle step performs this exchange and sorts the key-value pairs.

## B2 – Algebraic Operation (q40)

Modern big data processing systems introduce the concepts of:

1. Directed acyclic graphs.

* DAGs are used to specify the series of operators to run on a set of data. This is similar to a SQL query evaluation graph, but the graph is explicitly defined, with customized operators and customized parallelism.

1. Algebraic operations.

* Modern big data systems allow users to create their own algebraic operators beyond the operators in SQL. In MapReduce, for instance, users define operations that take an input and produce key-value outputs.

Briefly the concepts.

Prof. Ferguson suggested in lectures some hypothetical algebraic operators for IMDB namebasics. An excerpt of the data is below. Suggest a couple of hypothetical operators to transform the data.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **nconst** | **name** | **dob** | **dod** | **primaryProfessions** | **knownFor** |
| nm0000001 | Fred Astaire | 1899 | 1987 | soundtrack,actor,miscellaneous | tt0053137,tt0050419,tt0045537,tt0072308 |
| nm0000002 | Lauren Bacall | 1924 | 2014 | actress,soundtrack | tt0117057,tt0071877,tt0038355,tt0037382 |
| nm0000003 | Brigitte Bardot | 1934 |  | actress,soundtrack,music\_department | tt0056404,tt0049189,tt0057345,tt0054452 |
| nm0000004 | John Belushi | 1949 | 1982 | actor,soundtrack,writer | tt0077975,tt0078723,tt0072562,tt0080455 |
| nm0000005 | Ingmar Bergman | 1918 | 2007 | writer,director,actor | tt0050976,tt0060827,tt0083922,tt0050986 |
| nm0000006 | Ingrid Bergman | 1915 | 1982 | actress,soundtrack,producer | tt0036855,tt0038787,tt0034583,tt0038109 |
| nm0000007 | Humphrey Bogart | 1899 | 1957 | actor,soundtrack,producer | tt0042593,tt0037382,tt0034583,tt0043265 |
| nm0000008 | Marlon Brando | 1924 | 2004 | actor,soundtrack,director | tt0070849,tt0078788,tt0068646,tt0047296 |
| nm0000009 | Richard Burton | 1925 | 1984 | actor,soundtrack,producer | tt0061184,tt0059749,tt0057877,tt0087803 |
| nm0000010 | James Cagney | 1899 | 1986 | actor,soundtrack,director | tt0029870,tt0042041,tt0035575,tt0031867 |

One possible operator is one that takes the Name column and splits it into first, middle, last, title, etc. Another operator is one that parses primaryProfessions and splits it into dummy variables for the different categories. These operations can be performed in parallel (e.g. we split the data into groups of 1million rows).

## B3 – Concepts (q41)

Briefly explain the following concepts:

* Data Warehouse: a repository of data from multiple sources stored at a central location with a central schema. This allows stakeholders to gather data from different sources (e.g. firm divisions) as well as store historical data not directly used in day-to-day operations.
* Data Lake: unlike a data warehouse, which attempts to process/organize into a unified structure. In a data lake, we merely dump raw unstructured data into a central location—cleaning and analyzing is done when needed.
* Extract-Transform-Load: these are the steps in data warehousing. We first extract data from multiple sources (e.g. excel, Mongo, SQL). We then transform the data into the form we want (all relational, graph, etc.). Finally, we load it into the data warehouse.