Advance Management Data Notes - Prepared From PPT Slides

SOLUTIONS:-

1. Three-Level Architecture:-

- **External Level:** The users' view of the database describes that part of the database that is relevant to each user
- **Conceptual Level:** The logical view of the database describes what data is stored in the database and the relationships among the data.
- **Internal Level**: The physical representation of the database on the computer describes how the data is stored in the database

2. Data Independence

A major objective for the three-level architecture is to provide data independence, which means that upper levels are unaffected by changes to lower levels.

There are two kinds of data independence:

- Logical Data Independence: The immunity of the external schemas to changes in the conceptual schema
- Physical Data Independence: The immunity of the conceptual schema to changes in the internal schema

3. Concepts of the (Extended) Entity Relationship Model:

The Extended Entity-Relationship Model is a more complex and high-level model that extends an E-R diagram to include more types of abstraction, and to more clearly express constraints. All of the concepts contained within an E-R diagram are included in the EE-R model, along with additional concepts that cover more semantic information.

These additional concepts include:- generalization/specialization, union, inheritance, and subclass/super class.

4. Concepts of the relational model

- **Relation**: A relation is a table with columns and rows
- **Attribute**: An attribute is a named column of a relation.
- **Domain**: A domain is the set of allowable values for one or more attributes.
- **Primary key**: the candidate key that is selected to identify tuples uniquely within the relation.
- **Foreign key**: an attribute, or set of attributes, within one relation that matches the candidate key of some (possibly the same) relation.
- **Integrity constraints**:
  - **Domain integrity**: Describes the restriction on the set of allowed values for an attribute.
  - **Entity Integrity**: No attribute of a primary key can be null.
  - **Referential Integrity**: If a foreign key exists in a relation, either the foreign key value must match a candidate key value of some tuple in its home relation or the foreign key value must be wholly null.

Referential integrity is a relational database concept, which states that table relationships must always be consistent. In other words, any foreign key field must agree with the primary key that is referenced by the foreign key.

Views: The dynamic result of one or more relational operations operating on the base relations to produce another relation. A view is a virtual relation that does not necessarily exist in the Database but can be produced upon request by a particular user, at the time of request.

Null represents a value for an attribute that is currently unknown or is not applicable for this tuple.

5. Relational Algebra: Operations selection, projection, Cartesian product, equijoin, semi-join

Selection: SELECT * FROM R WHERE Predicates

$$\boxed{\sigma_{predicate}(R)}$$

Projection: Select DISTINCT a1 … an FROM R

$$\boxed{\Pi_{a_1, \ldots, a_n}(R)}$$

$$\boxed{R \times S}$$

Cartesian product: SELECT * FROM R, S

Semi Join

$$\boxed{R \triangleright_F S = \Pi_A(R \bowtie_F S)}$$

Equi Join:- $R \bowtie_{R.d = S.d} S$

6. What led to the introduction of PL/SQL and SQL/PSM?

Initial versions of SQL have been computationally incomplete (without programming constructs). Later versions of SQL could be embedded in a high-level programming language, but produced an impedance mismatch, because of the mixing of different programming paradigms.

SQL/PSM:

- SQL has been extended to a full programming language
- The extensions are known as SQL/PSM (Persistent Stored Modules)

PL/SQL: PL/SQL (Procedural Language SQL) is the Oracle version of an SQL programming language and has concepts similar to modern programming languages, such as variable and constant declarations, control structures, exception handling, and modularization.

7. What are cursors, exceptions, stored procedures (No syntactic details)?

Cursor: A SELECT statement can be used if the query returns exactly one row.

To handle a query that can return an arbitrary number of rows, PL/SQL uses cursors to allow the rows of a query result to be accessed one at a time. The cursor can be advanced by one to access the next row.

A cursor must be

- Declared and opened before it can be used
- Closed to deactivate it after it is no longer required

Exceptions:

- An exception is an identifier in PL/SQL raised during the execution of a block that terminates its main body of actions, although some final actions can be performed.
- An exception can be raised automatically or explicitly using the RAISE statement
- To handle raised exceptions, separate routines called exception handlers are specified.
- The exception handler itself is defined at the end of the PL/SQL block.

Stored Procedures:

- Sub program in PL/SQL that can take parameter and be invoked. They can modify and return data passed to them as a parameter.
- A stored procedure is a **subroutine** available to applications that access a **relational database management system (RDBMS).** Such procedures are stored in the database **data dictionary.**
- Typical uses for stored procedures include **data-validation** (integrated into the database) or access-control mechanisms.

8. How is a procedure different from a function?

Both can modify and return data passed to them as a parameter, but a function can only return a single value to the caller.

9. What are triggers?

A trigger is an SQL (compound) statement that is executed automatically by the DBMS as a side effect of a modification to a named table.

Triggers are stored programs, which are automatically executed or fired when some events occur

- A BEFORE trigger is fired before the associated event occurs
- An AFTER trigger is fired after the associated event occurs
- An INSTEAD OF trigger is fired in place of the trigger event

Row-level triggers (FOR EACH ROW) that execute for each row of the table that is affected by the triggering event.

Statement-level triggers (FOR EACH STATEMENT) that execute only once even if multiple rows are affected by the triggering event.

10. Advantage & Disadvantages of trigger:

Advantages

- Elimination of redundant code
- Simplifying modifications
- Increased security
- Improved integrity
- Improved processing power
- Good fit with the client-server architecture

Disadvantage:

- Performance Overhead

- Complexity
- Hidden functionality
- Cascading effects
- Cannot be scheduled

11. Discuss the general characteristics of advanced database applications

- large number of data types
- Less portable example: CAD, CAM, OIS
- Real-time access and large number of users, For example: CAD, Network Management System
- Large volume of data, For example: GIS, CAM

12. Discuss why the weaknesses of the relational data model and relational DBMSs may make them unsuitable for advanced database applications?

- Poor representation of real-world entities
- Semantic overloading
- Poor support for integrity and enterprise constraints
- Homogeneous data structure
- limited operations
- Difficulty handling recursive queries
- Impedance mismatch
- Other problems associated with concurrency, schema changes and poor navigational access

13. What is the Non-First Normal Form?

- A table is Non-First Normal Form if it has one or more column(s) that hold multiple values or the table has repeating columns.
- The First Normal Form (1NF) is the fundamental requirement for the relational model and states that values of attributes have to be atomic.

14. What are Nest, Unnest, and intersection join operations?

- **Nest**: Creates a set of values from one or more attributes if the values of the remaining attributes are identical.
- **Unnest**: Transformation of Nest into 1NF is called Unnest
- **Intersection Join**: If there is a nonempty intersection set between qualifying attributes, tuples are associated.

15. Discuss observer, mutator, constructor functions?

1. **Observer (get) Function**: which returns the current value of the attribute?
2. **Mutator(set) Function:** which sets the value of the attribute to a value specified as a parameter
3. **Constructor Function:**

- It is automatically defined to create new instances of a structured type
- Has the same name and type as the UDT
- Takes zero arguments

16. Discuss some extensions of ORDBMS to overcome limitations of RDBMS?

1. Row types and reference types
2. User-defined types
3. Super type / subtype relationships
4. User-defined procedures, functions, and methods
5. Table inheritance

6. Collection types (arrays, sets, lists, multisets)
7. Large objects

<u>16.1 Row types and reference types:-</u>

Sequence of attribute definitions that provides a new data type
A row type can be used to allow a column of a table to contain row values enables rows to be

- Stored in variables
- Passed as arguments to routines
- Returned as return values from function calls

<u>16.2 User-defined types</u>

ORDBMS allow the definition of user-defined types (UDT) which may be used in the same way as the predefined types
Two category:
****-Distinct Types**
Allow differentiation between the same underlying base types. It is not possible to treat an instance of one type as an instance of the other type.
****-Structured Type**
Consist of attribute definitions and routine declarations (methods), which can also be operator declarations.

<u>16.3 Super type / subtype relationships</u>

A subtype inherits all the attributes and methods of its Super type
A subtype can define additional attributes and methods like any other UDT and it can override inherited methods

**Table Inheritance is realized with the UNDER-clause of a CREATE TABLE statement.

<u>User-Defined Routines (UDRs)</u>

- Define methods for manipulating data
- Can return simple and complex values and should support overloading may be defined as part of a UDT or separately as part of a schema

<u>16.4 User-defined procedures, functions, and methods</u>

Procedure: May have zero or more parameters, each of which may be:

- An input parameter (in)
- An output parameter (Out)
- Both an input and output parameter(INOUT)

Function: An SQL-invoked function returns a value. Any parameters must be input parameters
Method: An SQL-invoked method is similar to a function but has some important differences:

- A method is associated with a single UDT
- The signature of every method associated with a UDT must be specified in that UDT

Three type of methods:

1. Constructor methods
2. Instance Methods
3. Static Methods

<u>16.5 Collection types (arrays, sets, lists, Multi sets)</u>

- Collection Types are type constructors that are used to define collections of other types
- Used to store multiple values in a single column of a table and can result in nested tables where a column in one table actually contains another table.

1. ARRAY one-dimensional array with a maximum number of elements
2. MULTISET unordered collection (duplicates allowed)
3. SET unordered collection without duplicates
4. LIST ordered collection (duplicates allowed)

16.6 Large objects

Large Objects is a data type that holds a large amount of data, such as a long text file or a graphics file:

- Binary Large Object (BLOB), a binary string that does not have a character set or collation association
- Character Large Object (CLOB)
-

17. What is a typed table? How it is different from other table types?

1. Typed table are defined based on UDT (User defined types). They can have levels in a hierarchy and describe the referenced object (from the model), and that description can be re-used in other tables or even in stored procedures. (In order to create a table with the same structure, it is not necessary to do a create like, but a create table based on the type)
2. Normal tables are defined based on columns, but that definition cannot be re-used in other objects, and it has to be redefined (redefine the column order and precision, etc.)

18. Discuss how reference types and object identity can be used
Reference Type: Can be used to uniquely identify a row within a table and to define relationships between row types
Object Identity: Object Identity is that aspect of an object that never changes and that distinguishes the object from all other objects and is independent of its name, structure and location.

19. What is a distributed Database?
Collection of logically related database nodes must be connected over a computer network to transmit data and commands among sites.
The nodes can be heterogeneous regarding

- Data
- Hardware
- Software

20. What are the main reasons for potential advantages of distributed databases?
Main Reason:

- Decompose big and unmanageable problems into smaller tasks
- Solve the parts in a coordinated way

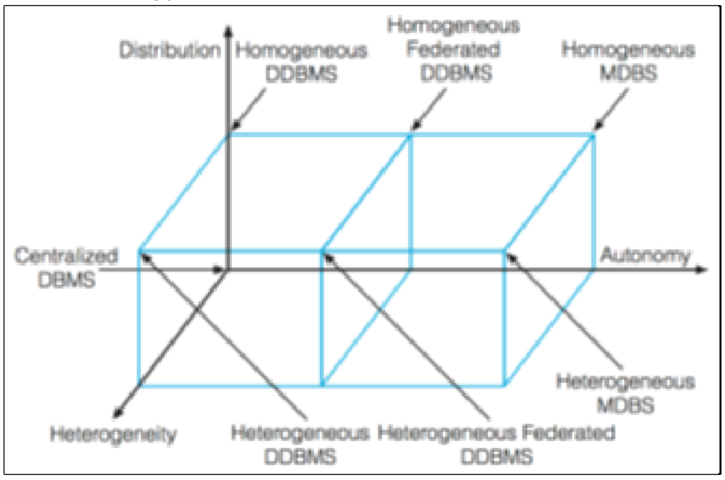
Advantage:

- Improved ease and flexibility of the application
- Increased availability
- Improved performance
- Easier expansion via scalability

21. What additional functions does a DDBMS have over a centralized DBMS?

- Security control
- Communication services
- Distributed query processing system catalog

22. Which types of Distributed DBS exist?



23. Discuss concepts of Distributed DB (Availability, Reliability, Scalability, Partition Tolerance, Autonomy, Replication, Allocation, and Transparency)

Reliability: Probability that a system is running at a certain time point. (point of time)
Availability: Probability that the system is continuously available during a time interval
Scalability: Determines the extent to expand the capacity of a distributed system while continuing to operate without interruption
Partition tolerance: A distributed system should have the capacity to continue operating while the network is partitioned.
Autonomy
Determines the extent to which individual nodes in a DDBMS can operate independently.

- **Design autonomy:** Data model usage and transaction management techniques among nodes are independent.
- **Communication autonomy:** To which extent each node can decide on sharing of information with other nodes?
- **Execution autonomy:** Users are independent to act as they wish.

Replication:-
A fragment is said to be replicated, if it is stored at more than one site.
Allocation schema:-
Describes the allocation of fragments to nodes
Transparency:-
The concept of transparency extends the general idea of hiding implementation details from end users.
A highly transparent system offers a lot of flexibility to the end user/application developer since it requires little or no awareness of underlying details on their part.

24. Discuss the architecture of a DDBMS. Within the context of a centralized DBMS, briefly explain new components introduced by the distribution of data.

DDBMS-Architecture

Global conceptual schema

- is a logical description of the whole database
- provides physical data independence from the distributed environment

Global external schemas

- provide logical data independence.

Fragmentation schema

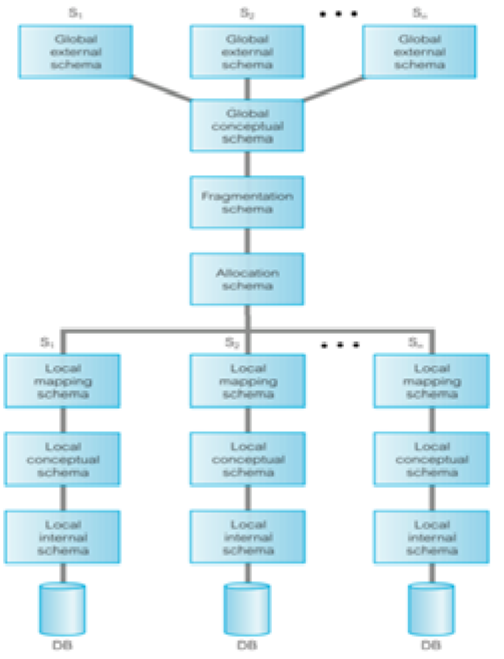
- description of how the data is to be logically partitioned

Allocation schema

- description of where the data is to be located, taking account of any replication

Local mapping schemas

- map fragments in the allocation schema into external objects in the local database



[Connolly & Begg]

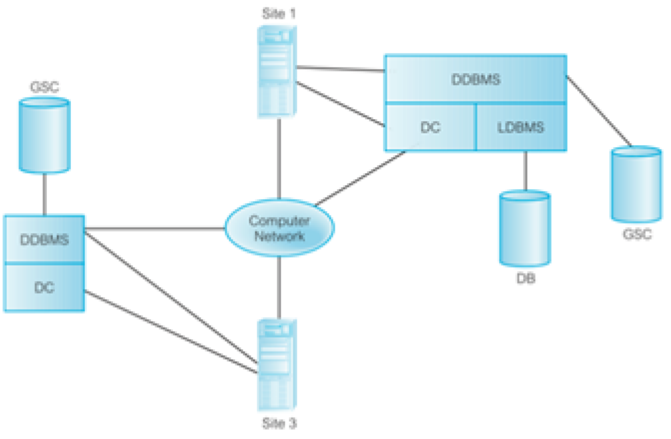
DDBMS Component Architecture

Local DBMS (LDBMS) component

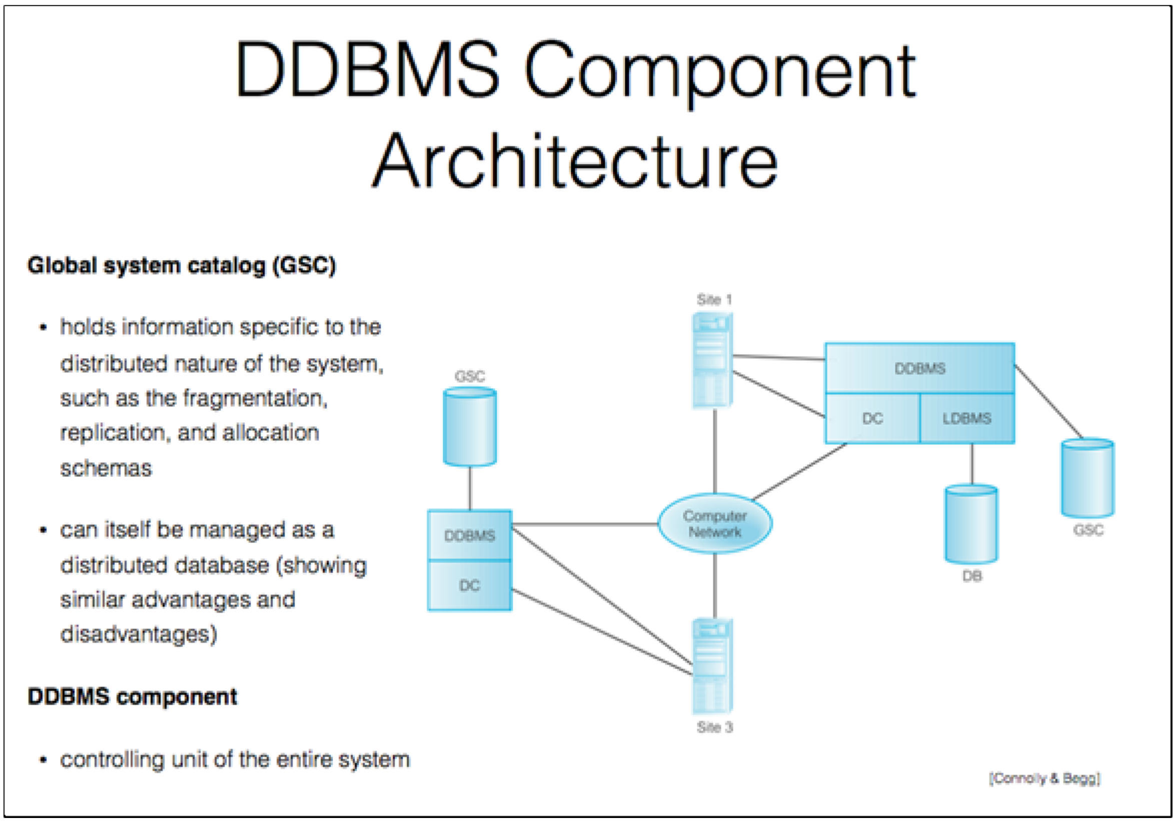
- standard DBMS, responsible for controlling the local data at each site that has a database
- has its own local system catalog that stores information about the data held at that site.

Data communications (DC) component

- software that enables all sites to communicate with each other
- contains information about the sites and the links



[Connolly & Begg]



DDBMS Component Architecture

Global system catalog (GSC)

- holds information specific to the distributed nature of the system, such as the fragmentation, replication, and allocation schemas
- can itself be managed as a distributed database (showing similar advantages and disadvantages)

DDBMS component

- controlling unit of the entire system

25. Degree of homogeneity of the DDBMS software.

- If all servers (or individual local DBMSs) use identical software and all users (clients) use identical software, the DDBMS is called homogeneous; otherwise, it is called heterogeneous.

26. Degree of local autonomy.

If there is no provision for the local site to function as a standalone DBMS, then the system has no local autonomy.
On the other hand, if direct access by local transactions to a server is permitted, the system has some degree of local autonomy.
Second possible answer:
(Each site is capable of independently processing user requests that require access to local data (that is, each site has some degree of local autonomy) and is also capable of processing data stored on other computers in the network.

27. Federated DBMS

Applications share a global view (schema) of the federation of databases.
A federated database (FDB) system is a hybrid of a distributed DBMS and a centralized DBMS:

- A distributed system for global users
- A centralized system for local users

28. Distribution transparency?

- In a distributed database system, transparency means that the DDBMS hides all the added complexities of distribution, allowing users to think that they are working with a single centralized system.(Distribution transparency is the property of distributed databases by the virtue of which the internal details of the distribution are hidden from the users).

29. Fragmentation transparency?

Fragmentation transparency: Two types of fragmentation are possible:

- **Horizontal fragmentation** distributes a relation (table) into sub-relations that are subsets of the tuples (rows) in the original relation.
- **Vertical fragmentation** distributes a relation into sub-relations where each sub-relation is defined by a subset of the columns of the original relation.

A global query by the user must be transformed into several fragment queries. Fragmentation transparency makes the user unaware of the existence of fragments.
(Fragmentation transparency enables users to query upon any table as if it were unfragmented. Thus, it hides the fact that the table the user is querying on is actually a fragment or union of some fragments)

30. Replication transparency

Copies of the same data objects may be stored at multiple sites for better availability, performance, and reliability.
Replication transparency ensures that replication of databases are hidden from the users. It enables users to query upon a table as if only a single copy of the table exists.

31. Multi database System:

Distributed DBMS in which each site maintains complete autonomy

- Distribution is realized by an additional software layer on top of the local systems
- Users can access and share data without requiring full database schema integration
- Users can manage their own databases without centralized control

32. What is a fragment of a relation? What are the main types of fragments? Why is fragmentation a useful concept in distributed database design?

What is fragment of a relation: (Internet) Relation may be divided into a number of sub-relations, which are distributed

What are the main types of fragment?

<u>Horizontal Fragmentation (Sharding):</u>

- Horizontal fragmentation divides a relation R horizontally by grouping rows to create subsets of tuples (shards).

<u>Vertical Fragmentation:</u>

- A vertical fragment of a relation keeps only certain attributes of a relation R and can be specified by a $\pi Li(R)$ operation in the relational algebra.

<u>Mixed Fragmentation:</u>

- Horizontal and vertical fragmentation can be intermixed.

32.1 Why is fragmentation useful concept in distributed database design?

- Several queries can be executed in parallel
- Allowing parallel execution of a single query

Security: Data not required by local applications is not stored
Efficiency: Data is stored close to where it is most frequently used.

33. Why is data replication useful in DDBMSs? What typical units of data are replicated?

Data Replication is the process of storing data in more than one site or node. This is necessary for improving the availability of data. There can be full replication, in which a copy of the whole database is stored at every site. There can also be partial replication, in which case, some fragments (important frequently used fragments) of the database are replicated and others are not replicated.

- Improve application reliability and availability.
- Improve read performance

- Less communication overhead

Typical data Units:
ROW, COLUMN(Fragments of table), whole table, complete replication set. Availability, Increased parallelism, Less Data Movement over Network

34. What is meant by data allocation in distributed database design? What typical units of data are distributed over sites?
Data allocation: Each (copy of a) fragment must be assigned to a particular site in the distributed system.
Typical data Units:
Different Types of fragments: Horizontal, Vertical, Mixed Fragmentation

35. How is a horizontal partitioning of a relation specified? How can a relation be put back together from a complete horizontal partitioning?
Horizontal partitioning divides a table into multiple tables. Each table then contains the same number of columns, but fewer rows. For example, a table that contains 1 billion rows could be partitioned horizontally into 12 tables, with each smaller table representing one month of data for a specific year.

The primary key is duplicated to allow the original table to be reconstructed.
Using join operation to reconstruct them.

36. How is a vertical partitioning of a relation specified? How can a relation be put back together from a complete vertical partitioning?
Vertical partitioning involves creating tables with fewer columns and using additional tables to store the remaining columns. Normalization also involves this splitting of columns across tables, but vertical partitioning goes beyond that and partitions columns even when already normalized.
The primary key is duplicated to allow the original table to be reconstructed.
Using join operation to reconstruct them.

37. What are the different stages of processing a query in a DDBMS?
- **Query Mapping**: The input query on distributed data is specified formally using a query language. It is then translated into an algebraic query on global relations using the global conceptual schema.
- **Localization**: The distributed query is mapped on the global schema to separate queries on individual fragments using data distribution and replication information.
- **Global Query Optimization:** Selecting a strategy from a list of candidates that is closest to optimal. A list of candidate queries can be obtained by permuting the ordering of operations within a fragment query generated by the previous stage. The total cost is a weighted combination of CPU cost, I/O costs, and communication costs.
- **Local Query Optimization:** The techniques are similar to those used in centralized systems.

38. Discuss the different techniques for executing an equijoin of two files located at different sites. What main factors affect the cost of data transfer?
Technique 1:
Transfer second table (from 2nd site) to 2nd table (first site) and join them, then transfer the result of join into site 3
Technique 2:

Transfer 1st table (from 1st site) to the 2nd table (second site) and join them, then transfer the result of join into site 3

Main factors: We have to choose the site and table which contains less data in order to minimize the data transfer cost

39. Discuss the semi join method for executing an equijoin of two files located at different sites. Under what conditions is an equijoin strategy efficient?
1. Send the joining column's jc of one relation R to the site where the other relation S is located
2. Join jc with S
3. Project the jc and attributes required in the result from S and transfer them to the site of R
4. Join the transferred columns with R

Efficient: When the numbers of columns which are supposed to transfer are small

40. Discuss Distributed Catalog Management (Internet)

- Catalog in a distributed DBMS keeps track of how data is distributed (fragmented) across sites
- The distributed database catalog entries must specify site(s) at which data is being stored in addition to data in a system catalog in a centralised DBMS. Because of data partitioning and replication, this extra information is needed. There are a number of approaches to implementing a distributed database catalog.

1. Centralised - Keep one master copy of the catalog
2. Fully replicated - Keep one copy of the catalog at each site
3. Partitioned - Partition and replicate the catalog as usage patterns demand
4. Centralised/partitioned - Combination of the above

41. Which problems can arise while processing distributed transactions?
- Dealing with multiple copies of data items
- Failure of individual sites
- Failure of communication links between nodes when network partitioning occurs
- Distributed commit
- Distributed deadlock

42. Which techniques can guarantee consistency of replicated data items?
1. Primary Site Technique
2. Primary Site with Backup Site
3. Primary Copy Technique

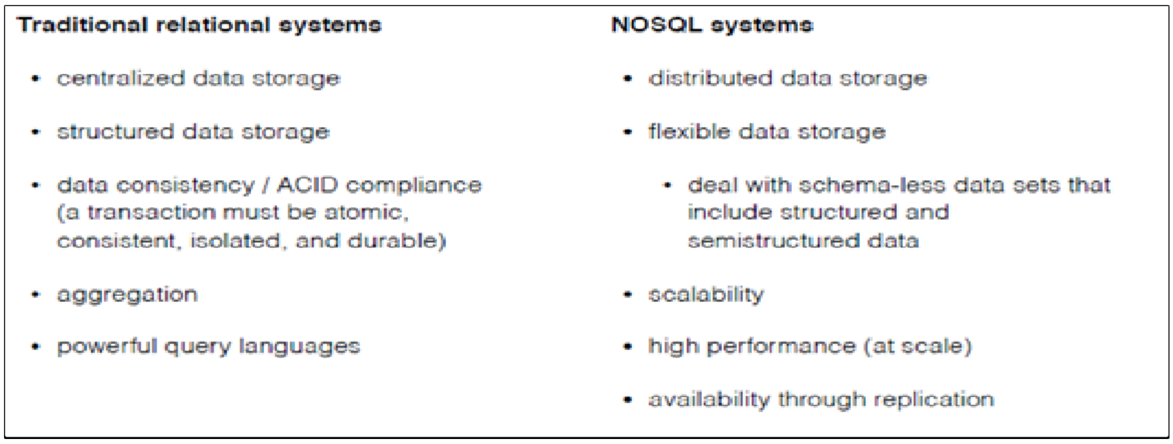
43. Compare the primary site method with the primary copy method for distributed concurrency control. How does the use of backup sites affect each?

<u>Primary Site Technique:</u>
A single primary site is designated to be the coordinator site for all database items.
➔ All requests for locking or unlocking are sent at the primary site.
<u>Advantage</u>
- Simple extension of the centralized lock mechanism Disadvantages
- Potential system bottleneck
- System reliability and availability is limited, since a failure of the primary site stops the system

<u>Primary Copy Technique:</u>
A particular copy of each data item is designated as a distinguished copy

- The distinguished copies of different data items are stored at different sites to distribute the load of lock coordination among various sites
- A failure of one site affects any transactions that are accessing locks on items whose primary copies reside at that site, but other transactions are not affected
- Reliability and availability can be further enhanced by using backup sites

44. Discuss application of traditional relational systems and NOSQL systems



Traditional relational systems	NOSQL systems
• centralized data storage	• distributed data storage
• structured data storage	• flexible data storage
• data consistency / ACID compliance (a transaction must be atomic, consistent, isolated, and durable)	• deal with schema-less data sets that include structured and semistructured data
• aggregation	• scalability
• powerful query languages	• high performance (at scale)
	• availability through replication

45. CAP Theorem:
In a distributed system with data replication only two of the following properties can be guaranteed at the same time:

- Consistency: the nodes will have the same copies of a replicated data item visible for various transactions
- Availability: each read or write request for a data item will either be processed successfully or will receive a message that the operation cannot be completed
- Partition Tolerance: the system can continue to operate if the network connecting the nodes has a fault that results in two or more partitions, where the nodes in each partition can only communicate among each other

46. Eventual Consistency:
If no new updates are made to a given data item eventual consistency informally guarantees that eventually all accesses to that item will return the last updated value.

47. Sharding

- Often, data collections of NOSQL applications can have many billions of records.
- This vast amount of data may be accessed by millions of users concurrently.
- So it is not possible (or practical) to store all records in one node.
- Sharding (horizontal partitioning) is often employed in NOSQL systems to distribute the records to multiple nodes.
- By combining sharding and replicating the shards both load balancing as well as data availability are improved.

48. Schema-less data
Semi structured Data (schema-less / self-describing data)

- Data may have a certain structure, but not necessarily an identical structure
- Some attributes may be shared, but some other attributes may be used by only a few entities

- There is no predefined schema; instead the schema information is mixed in with the data values

49. Categories of NOSQL Systems (Examples in lecture 11 and 12)

1. Document stores

- Document-based NOSQL systems store data as collections of similar documents.
- Resemble complex objects
- Do not require to specify a schema, but are specified as self-describing data
- Each document can have different data elements (attributes)
- Can be specified in various formats, such as XML or JSON (JavaScript Object Notation)
- Are accessible via their document id

Examples: Mongo DB

2. Key-value stores

- Every data item (value) must be associated with a unique key
- Retrieving the value by supplying the key must be very fast
- Value

1. Can have very different formats for different key-value storage systems:
2. String / array of bytes: the application using the key-value store has to interpret the structure of the data value
3. Structured data rows (tuples) similar to relational data
4. Semi structured data using a self-describing data format

Example: Amazon Dynamo DB, Project Voldemort

3. Wide column stores

A wide column store is a type of key-value database. It uses tables, rows, and columns, but unlike a relational database, the names and format of the columns can vary from row to row in the same table.

Working Principle

- Vertical partitioning - tables are partitioned by column into column families
- Each column family is stored in its own files
- Versioning of data values is allowed
- The key is multidimensional (in contrast to key-value stores)

Examples: Google distributed storage system (BigTable), Apache Hbase

4. Graph-Databases

Data is organized as a graph, which is a collection of nodes, relationships, and properties.

Node

- Can contain properties
- Can contain labels, which groups nodes with the same label into subsets for querying purposes

Relationship

- Is directed, each relationship has a start node and an end node
- Can contain properties
- Has a relationship type, which helps to identify similar relationship types for querying purposes

Properties

Store the data items associated with nodes and relationships as list of key-value pairs

Example: Neo4j

5. Hybrid Systems

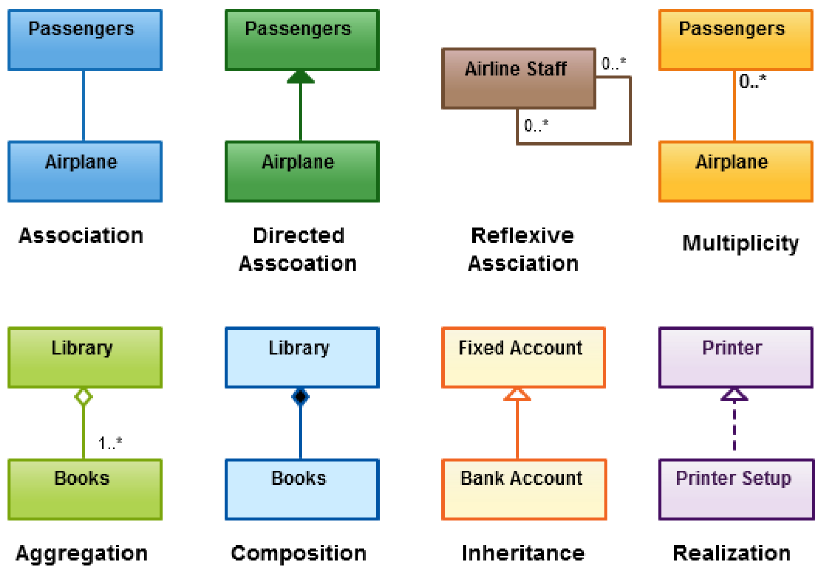
Hybrid SQL-NoSQL database solutions combine the advantage of being compatible with many SQL applications and providing the scalability of NoSQL ones.
Example: Xeround

44. What is consistent Hashing?
Consistent hashing

- Special kind of hashing, which minimizes the number of keys that have to be remapped when the size of a hash table is changed
- Assumes that the result of the hash function h(key) is an integer value, usually in the range 0 to Hmax = 2n-1, where n is chosen based on the desired range for the hash values

AMD Tutorials:
Concepts of Unified Modelling Language:-

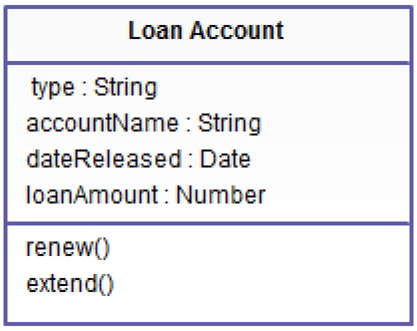
UML Class Diagram



Relationships in UML class diagrams

What are the Class Diagrams?

Class diagrams are the main building block in object-oriented modeling. They are used to show the different objects in a system, their attributes, their operations and the relationships among them. The following figure is an example of a simple class:



In the example, a class called “loan account” is depicted. Classes in class diagrams are represented by boxes that are partitioned into three:

- The top partition contains the name of the class.
- The middle part contains the class’s attributes.
- The bottom partition shows the possible operations that are associated with the class.

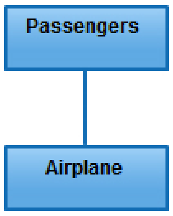
The example shows how a class can encapsulate all the relevant data of a particular object in a very systematic and clear way. A class diagram is a collection of classes similar to the one above.

Relationships in Class Diagrams

Classes are interrelated to each other in specific ways. In particular, relationships in class diagrams include different types of logical connections. The following are such types of logical connections that are possible in UML:

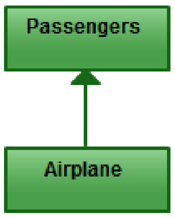
- Association
- Directed Association
- Reflexive Association
- Multiplicity
- Aggregation
- Composition
- Inheritance/Generalization
- Realization

Association



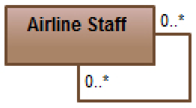
Association is a broad term that encompasses just about any logical connection or relationship between classes. For example, passenger and airline may be linked as above:

Directed Association



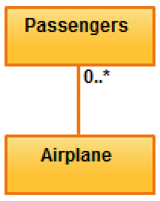
Directed Association refers to a directional relationship represented by a line with an arrowhead. The arrowhead depicts a container-contained directional flow.

Reflexive Association



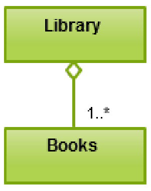
This occurs when a class may have multiple functions or responsibilities. For example, a staff member working in an airport may be a pilot, aviation engineer, a ticket dispatcher, a guard, or a maintenance crew member. If the maintenance crew member is managed by the aviation engineer there could be a managed by relationship in two instances of the same class.

Multiplicity



Multiplicity is the active logical association when the cardinality of a class in relation to another is being depicted. For example, one fleet may include multiple airplanes, while one commercial airplane may contain zero to many passengers. The notation 0..* in the diagram means "zero to many".

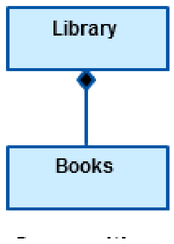
Aggregation



Aggregation refers to the formation of a particular class as a result of one class being aggregated or built as a collection. For example, the class "library" is made up of one or more books, among other materials. In aggregation, the contained classes are not strongly dependent on the lifecycle of the container. In the same example, books will remain so even when the library is dissolved. To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

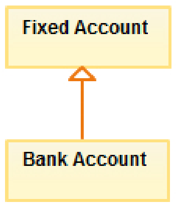
Composition



The composition relationship is very similar to the aggregation relationship. with the only difference being its key purpose of emphasizing the dependence of the contained class to the life cycle of the container class. That is, the contained class will be obliterated when the container class is destroyed. For example, a shoulder bag's side pocket will also cease to exist once the shoulder bag is destroyed.

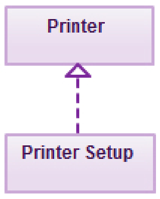
To show a composition relationship in a UML diagram, use a directional line connecting the two classes, with a filled diamond shape adjacent to the container class and the directional arrow to the contained class.

Inheritance / Generalization



Inheritance refers to a type of relationship wherein one associated class is a child of another by virtue of assuming the same functionalities of the parent class. In other words, the child class is a specific type of the parent class. To show inheritance in a UML diagram, a solid line from the child class to the parent class is drawn using an unfilled arrowhead.

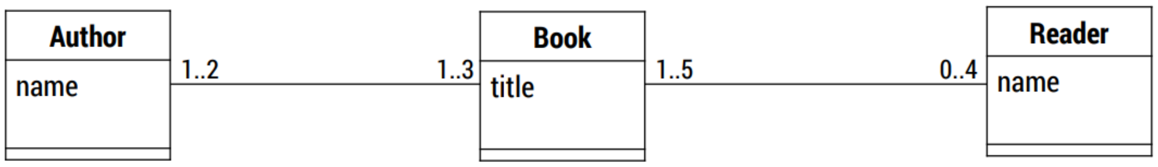
Realization



Realization denotes the implementation of the functionality defined in one class by another class. To show the relationship in UML, a broken line with an unfilled solid arrowhead is drawn from the class that defines the functionality of the class that implements the function. In the example, the printing preferences that are set using the printer setup interface are being implemented by the printer.

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

1. Consider the following UML diagram:



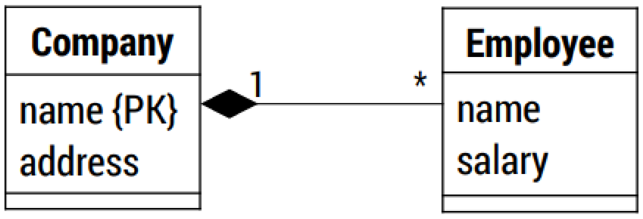
If there are 6 Authors

- **What's the minimum and maximum number of Books?**
  Min = 3 (6/2 * 1), Max = 18 (6/1 * 3)
- **What's the minimum and maximum number of Readers?**
  Min = 0 (3 * 0 = 0), Max = 72(18 * 4)

If there are 6 Readers

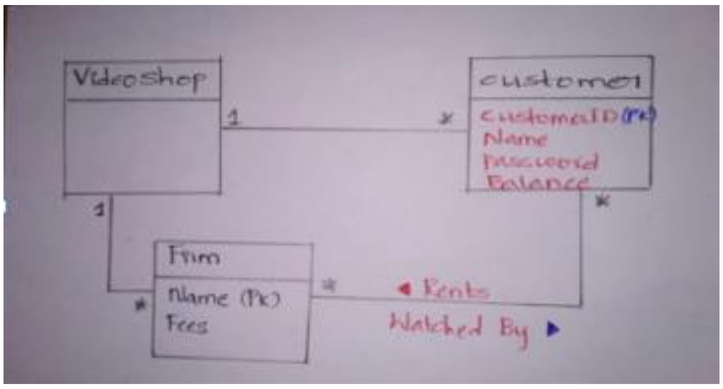
- **What's the minimum and maximum number of Books?**
  Min = 2. Max = Infinity
- **What's the minimum and maximum number of Authors?**
  Min = 1. Max = Infinity

2. Consider the following UML diagram True or False?



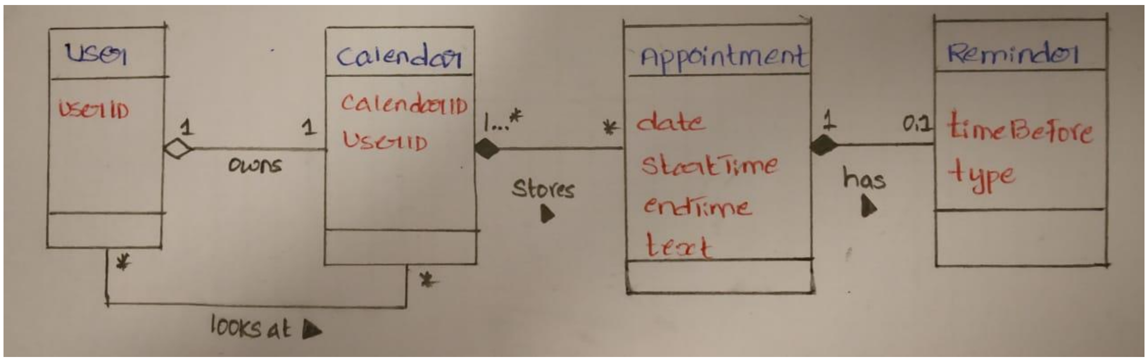
- No two companies can have the same name. (**TRUE**)
- No two employees can have the same name. (**FALSE**)
- No two companies can be at the same address. (**FALSE**)
- No two employees can work at the same address. (**FALSE**)
- Each employee works for at least one company. (**TRUE**)
- No employees work for more than one company. (**TRUE**)
- Each company has at least one employee. (**FALSE**)
- Two employees with the same name cannot work for the same company. (**FALSE**)
- Two employees with the same name cannot work for different companies. (**FALSE**)

2. There is an online rental video shop. Customers of this shop may log in with their customer ID and their password. Then they can rent films that each has a different name and a rental fee, but they may only watch it, if they have enough prepaid balance with the shop.



3. Draw an UML diagram that models this information Translate the UML diagram to a relational schema.

People use digital calendars for managing their appointments. Everybody has its own calendar and may browse each other's calendar and it is possible to book common appointments. An appointment starts and ends at a given time on a given day and is described by a text. It can be specified that the calendar owner gets reminded of an appointment and also how long before the appointment this should happen. Reminders are of different kinds: a signal in the computer's loudspeaker, a pop-up window with the description, or an e-mail containing the description.



Relational Schema:-

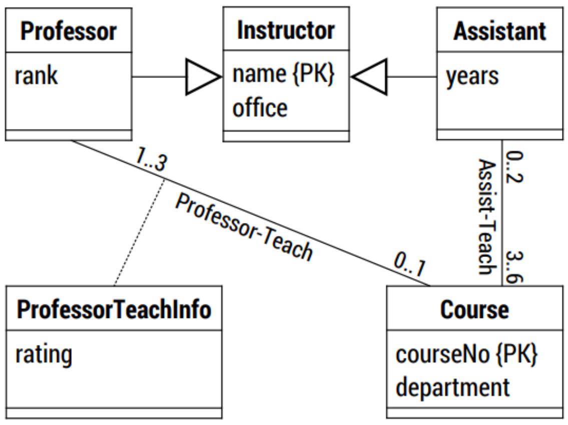
User (<u>userID</u>)

Calendar (<u>calendarID</u>, **userID**)

Appointment (<u>appointmentID</u>, **calendarID**, date, startTime, endTime, Text)

Reminder (<u>**appointmentID**</u>, timeBefore, Type)

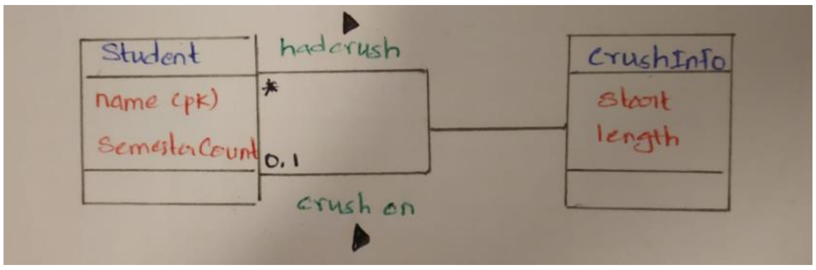
4. Consider the following UML diagram



- **What is the minimum and maximum total number of instructors for a given course?**
  Solution: Min: 0 + 1 = 1, Max: 2 + 3 = 5
- **What is the minimum and maximum teaching load (number of courses) for professors and for assistants?**
  Solution:
  P-Min: 0, P-Max: 1
  A-Min: 3, A-Max: 6
- **Translate the UML diagram to a relational schema.**
  Solution:
  Professor (Name, Office, Rank, courseNo, Rating)
  Assistant (Name, Office, Years)
  Course (courseNo, Department)
  AssistTech (Name, courseNo)
- **Specify a minimal key for each relation in your solution to part c.**
  Solution:
  Professor (Name)
  Assistant (Name)
  Course (courseNo)
  AssistTech (Name, courseNo)
- **Suppose by default attribute values cannot contain null. Does your solution to part c require any attributes to permit null values?**
  Solution: Professor.courseNo and Professor.rating must permit nulls

5. Draw an UML diagram that models this information Translate the UML diagram to a relational schema

Consider a tiny social network containing university students and their "crushes" (desired romantic relationships). Each student may have a crush on at most one other student, and associated with each crush is the beginning and length of time the crush has been going on. Students have a name and a semester count, and names are unique. Hint: Make sure to capture the asymmetry and multiplicity of the crush relationship



Relational Schema:-

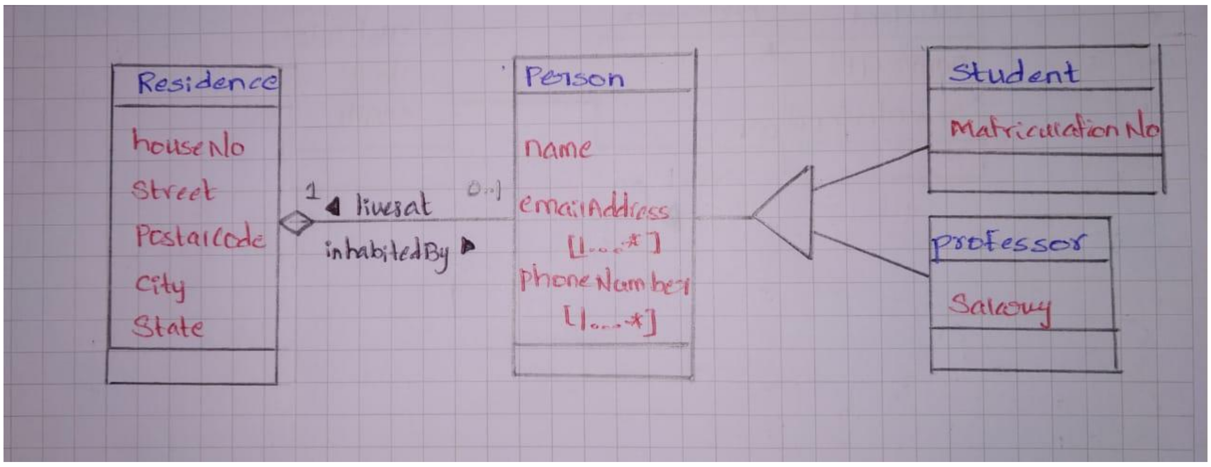
Student (<u>name</u>, semesterCount)

CrushInfo (**<u>studentName</u>**, **crushOnStudentName**, start, length)

NOTE: minimal keys are underlined.

6. Draw an UML diagram that models this information. Translate the UML diagram to a relational schema

Every person has a name, a residence, at least one email-address and it also may have several phone-numbers. Each residence consists of a street name, a house number, a city, a postal code and a state. Their lives only one person at each residence, but there may be empty residences without any inhabitants. There are two types of persons: students, which have a matriculation number and professors, which have a salary.



Relational Schema:-

Residence (<u>residenceID</u>, houseNo, street, postalCode, city, state)

Email (**<u>residenceID</u>**, <u>address</u>)

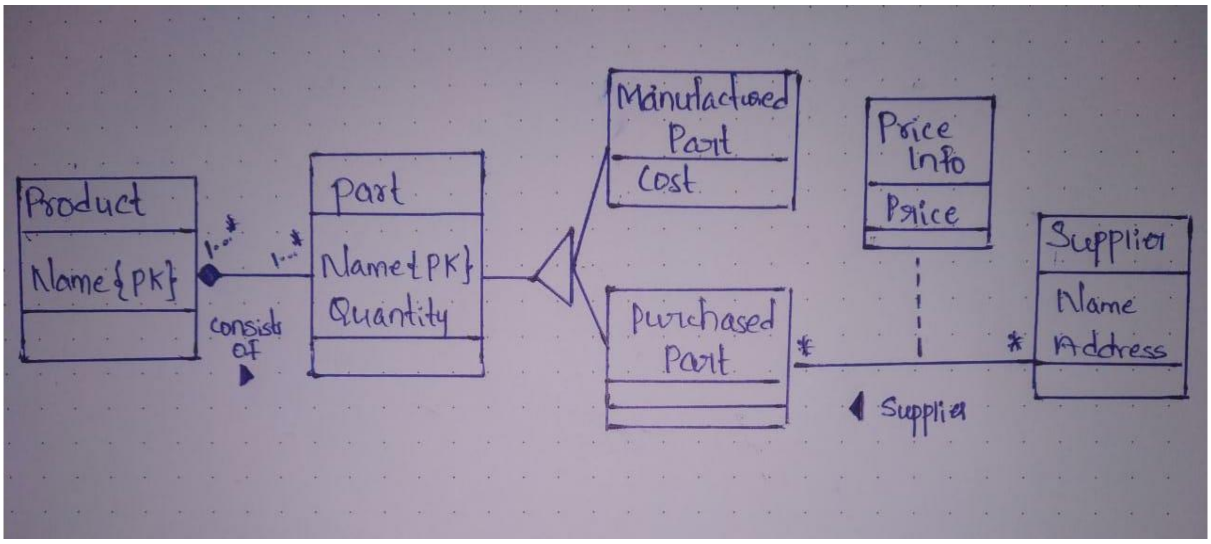
Phone (**<u>residenceID</u>**, <u>number</u>)

Student (**<u>residenceID</u>**, name, matriculationNo)

Professor (**<u>residenceID</u>**, name, salary)

7. Draw an UML diagram that models this information. Translate the UML diagram to a relational schema

A company produces products that are assembled from parts. A part is either manufactured internally or bought from a supplier. Each product and part has unique names. The company keeps track of the quantity in stock of the different parts and knows its own manufacturing costs. For Purchased parts it has different suppliers, each with their own prices. The suppliers are known by Name and address.



Relational Schema:-

Product (<u>Name</u>)

Part (<u>Name</u>, Quantity, Cost)

ProductPart (<u>**productName**</u>, <u>**partName**</u>)

PurchasedPart (<u>**partName**</u>)

ManufacturedPart (<u>**partName**</u>, Cost)

Supplier (<u>supplierID</u>, Name, Address)

SupplierPrice (<u>**supplierID**</u>, <u>**partName**</u>, Price)

8. Draw an UML diagram that models this information. Translate the UML diagram to a relational schema

A bank's database needs to store information about employees, branches of the bank and customers. Each employee has a name, a salary and a phone and is assigned to exactly one branch of the bank, with the start date at that branch recorded. Each employee is managed by at most one other employee. Each branch has a name and an address. A customer has a name and a address and some combination of one or more loans from the bank and/or one ore more deposit accounts with the bank. Each loan is identified by a loan-number with a record of the current balance. A complete history of loan payments is recorded giving the payment date and payment amount for each payment of the loan. There are never two payments for the same loan recorded for the same date. Each loan is administered by exactly one branch of the bank. An account is identified by an account number, with an additional indication of the current balance. An account may be a checking account (with an allowable overdraft amount recorded) or a saving account (with an interest rate recorded).

Relational Schema:-

Employee (<u>employeeID</u>, Name, Salary, Phone, **managerID**, **branchID**, startDate)

Branch (<u>branchID</u>, Name, Address)

Customer (<u>customerID</u>, Name, Address)

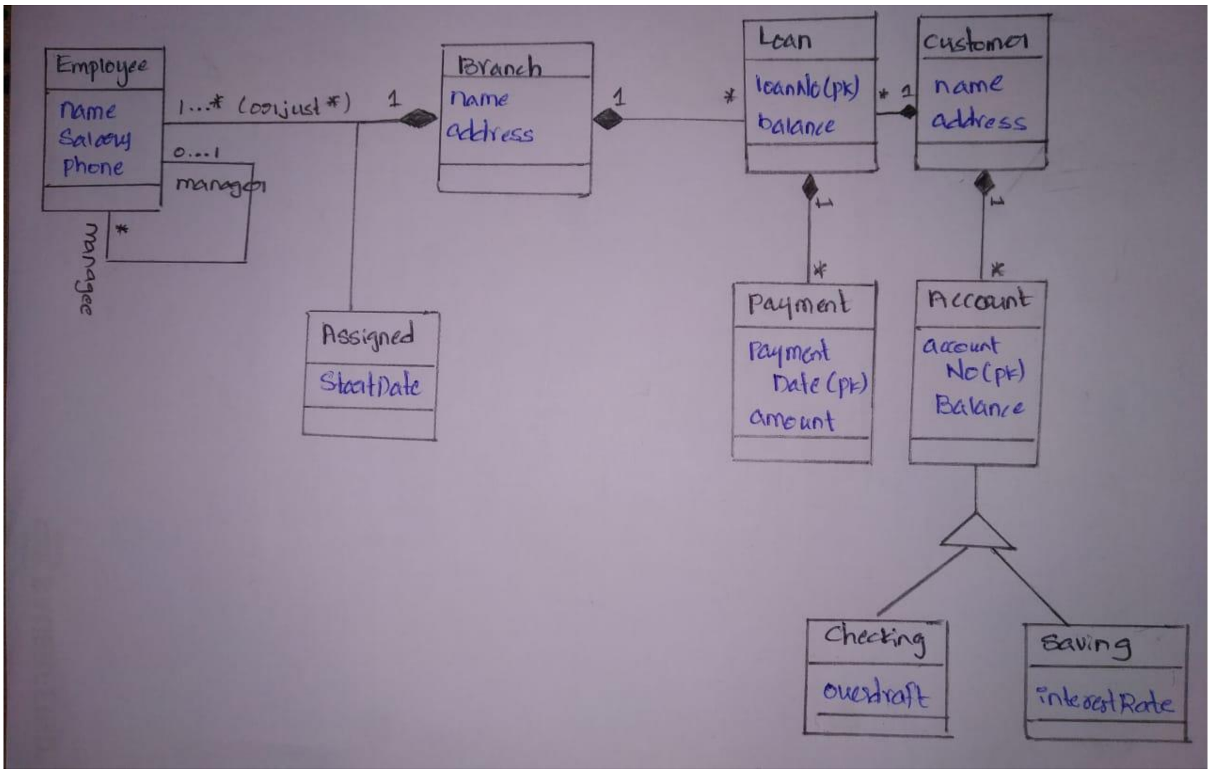
Loan (<u>loanNo</u>, Balance, **branchID**, **customerID**)

Payment (<u>**loanNo**</u>, <u>paymentDate</u>, Amount)

Account (<u>accountNo</u>, **customerID**, Balance)

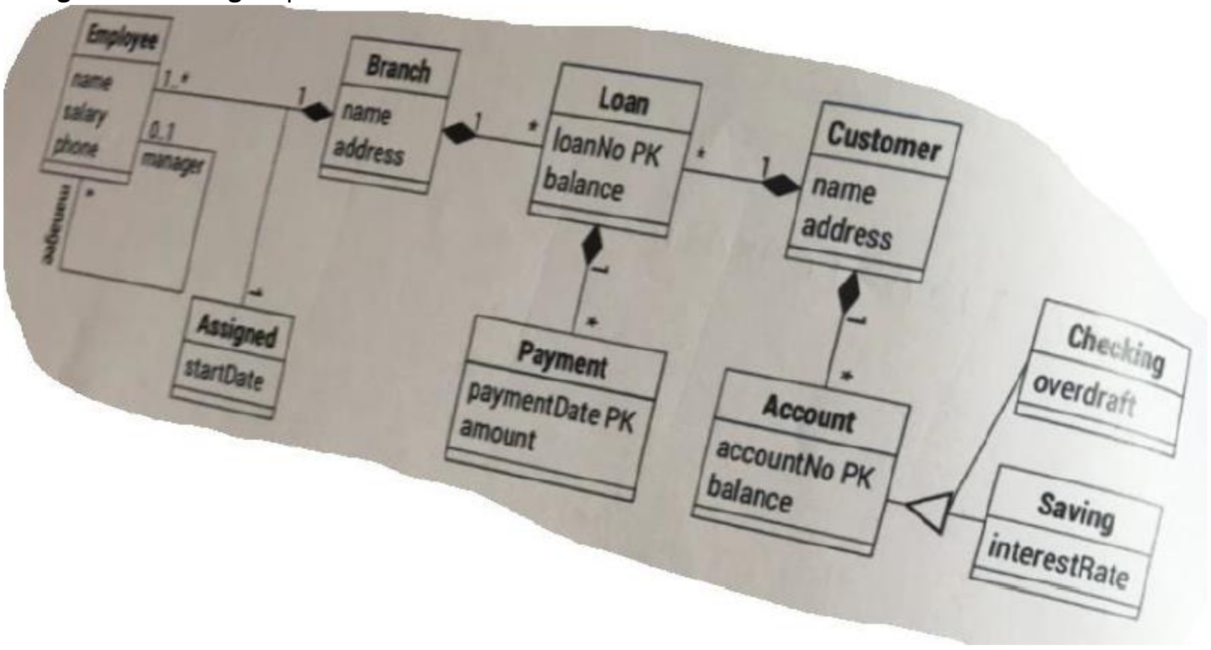
Checking (<u>**accountNo,**</u> overdraft)

Saving (<u>**accountNo**</u>, interestRate)



PREVIOUS YEAR QUESTION PAPERS

2. Logical Modelling 15 points



- Drive a relational schema from the diagram!
- Underline primary keys!
- Mark **foreign keys** in bold!

Hint: The primary key for employee, Branch and customer are not given in the diagram (you can assume artificial IDs (EmployeeID, BranchID, CutomerID)).

Question from WS 2019: Something Similar one with letters A, B,C)

Employee (EmployeeID, name, salary, phone, BranchID)
Branch (BranchID**,** name, address)
Assigned (startDate, **BranchID, EmployeeID**)
Loan (LoanNo, Balance, **BranchID, CustomerID**)
Payment (PaymentDate, amount, **LoanNo)**
Account (accountNo, balance, **CustomerID)**
Checking (Overdraft, **accountNo,** checkID)
Saving (IntrestRate, **accountNo**, SaveID)
Customer (customerID, name, address)

3. Weaknesses and extensions of the relational model 12(6+3+3) Points
3.1 Name and explain at least three weaknesses of the relational data model/RDBSs!
1. Poor representation of “real-world” entities
Entities are described by multi-value attribute. Is inefficient to many joins during query process
2. Semantic overloading
There is no mechanism to distinguish between entities and relationship
3. Limited operations
The relation model has only fixed set of operations
4. Impedance mismatch
This approach produces an impedance mismatch because different programming paradigms are mixed

3.2 Why is it difficult to use a conventional programming language such as C in connection with SQL?

1. This approach produces an impedance mismatch, because different programming Paradigms are mixed.
2. Common high-level languages (such as C) are procedural languages that can handle only one row of data at a time
3. SQL is a declarative language that handles many rows of data at a time.

3.3 What are cursors and how are they used?

A SELECT statement can be used if the query returns one and only one row. To handle a query that can return an arbitrary number of rows (that is, zero, one, or more rows), SQL uses cursors to allow the rows of a query result to be accessed one at a time.

A cursor must be

- Declared and opened before it can be used
- Closed to deactivate it after it is no longer required

Once the cursor has been opened, the rows of the query result can be retrieved one at a time using a FETCH statement.

4. Triggers 9 (2+3+4) points

4.1 What is a trigger?

A trigger is an SQL statement that is executed automatically by the DBMS as a side effect of a modification

- The triggering event - to a named table.

The act of executing a trigger is sometimes known as firing the trigger.

The basic format of the CREATE TRIGGER statement is as follows:

```
CREATE TRIGGER TriggerName
BEFORE | AFTER | INSTEAD OF <triggerEvent> ON <TableName>
[REFERENCING <oldOrNewValuesAliasList>]
[FOR EACH {ROW | STATEMENT}]
[WHEN (triggerCondition)]
<triggerBody>
```

4.2 Question from WS 2019: What are the Difficulties of designing trigger?

4.2 Name at least three use-cases for triggers?

Use Cases

- Validating input data and maintaining complex integrity constraints
- Supporting alerts that action needs to be taken when a table is updated in some way
- Refreshing derived attributes after an update operation
- Maintaining audit information, by recording the changes made, and by whom
- Supporting replication

4.3 Consider an empty PostgreSql database and execution of the following lines of code

```
CREATE TABLE IF NOT EXISTS numbers (number INTEGER NOT NULL);
CREATE OR REPLACE FUNCTION trigger_func() RETURNS TRIGGER AS $$
BEGIN
        IF NEW.number % 2 THEN
        INSERT INTO numbers VALUES (NEW.number + 1);
        ELSE
        RETURN NULL;
        END IF;
        NEW.number = NEW.number - 1;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER  trigger_exec AFTER INSERT ON numbers
FOR EACH ROW
        EXECUTE PROCEDURE trigger_func();
        INSERT INTO numbers VALUES (1), (5), (10), (15), (20);
        SELECT * FROM numbers ORDER BY number;
```

Solution: 1, 2, 5, 6, 10, 15, 16, 20.

QUESTION FOR WINTER 2019:-

```
CREATE TABLE IF NOT EXISTS numbers (number INTEGER NOT NULL);
CREATE OR REPLACE FUNCTION trigger_func() RETURNS TRIGGER AS $$
BEGIN
FOR num IN 0 . . NEW.number – 1 LOOP
DELETE FROM numbers WHERE number = num;
IF NOT FOUND THEN
        INSERT INTO numbers VALUES (num);
END IF;
END LOOP;
RETURN NULL;
```

5. Non-First Normal Form 8(3+2+3) Points

5.1 Define the Non-First Normal Form (NF2)!

A table is Non-First Normal Form if it has one or more column(s) that hold multiple values or the table has repeating columns.

The First Normal Form (1NF) is the fundamental requirement for the relational model and states that values of attributes have to be atomic.

Which first normal form is violated, such as non-relational databases. These are said to be in non-first normal form, and have the particularity of having multi-valued attributes.

Objects of the Non First Normal Form are defined as follows:

- Every atomic value (such as integer, float, string) is an object.
- If a1, a2, … , an are distinct attribute names and o1, o2, … , on are objects, then (a1:o1, a2:o2, … , an:on) is a tuple object.
- If o1, o2, … , on are objects, then {o1, o2, … , on} is a set object

5.2 What is the purpose of the Nest Operation?

“It Create a set of values from one or more attributes, if the values of the remaining attributes are identical.”

5.3 Given is the depicted relation ProjStaff, which associates staff members to projects: Nest ProjStaffby S_ID and draw the resulting NF2-relation!

ProjStaff	
P_ID	S_ID
1	1
3	1
2	2
3	4
3	5
1	2
1	5
4	4

P_ID	S_ID
1	{1,2,5}
3	{1,4,5}
2	2
4	4

6. Data Fragmentation 11(5+6) Points

Given in the following relation Student, Which Describes student data:

Student			
Matriculation Number	**Name**	**StudyCourse**	**Home Address**
134	**Archimed**	**Web Engineering**	**Regentstrasse 2,london**
252	**sophokies**	**Auto-Software**	**Broad street 5,newyork**
363	**phythagores**	**Auto-Software**	**Alex platx4,berlin**
464	**hippok**	**Web Engineering**	**lover 7,paris**

6.1 Create a complete horizontal partition based on the assumption, that students of Web Engineering will be stored and processes at a different site than students of Automotive Software Engineering. Give all necessary SQL-statements, which describe the content of each required fragment!

SELECT * from student where studyCourse = ‘webengineering’;
SELECT * from student where studyCourse = ‘Automotivesoftware’;

How can the Student relation be reconstructed from the created fragments?
Student relation can be reconstructed of horizontal fragmentation can be performed using union operation on fragments.

6.2 For a certain use-case only attributes *Name* and *StudyCourse* of Students are required. Another use-case only need attributes *Name* and *HomeAddress* of each student. Create a vertical fragment for each use case! Specify the content of your fragments by using SQL-statements!

SELECT Name, studyCourse FROM STUDENT;
SELECT Name, homeAddress FROM STUDENT;

Can the student relation be reconstructed from your fragments? If yes, give the necessary SQL-statement. If not, explain!
No, it cannot be reconstructed. Because in above SQL-statement primary key is missing;

7. Distributed Database 16(6+9+1) Points
7.1 What are advantages of Distributes Databases? Give at least three examples and explain them briefly!

- Better representation of organizational structures
- Improved shareability and local autonomy
- increased availability and reliability (due to replication)
- improved performance
- Economics - it may cost less to create a network of smaller computers with the power of a single large computer

1. Reflects organizational structure many organizations are naturally distributed over several locations.
2. Improved shareability and local autonomy The geographical distribution of an organization can be reflected in the distribution of the data; users at one site can access data stored at other sites. Data can be placed at the site close to the users who normally use that data.
3. Improved availability In a centralized DBMS, a computer failure terminates the operations of the DBMS. However, a failure at one site of a DDBMS or a failure of a communication link making some sites inaccessible does not make the entire system inoperable. Distributed DBMSs are designed to continue to function despite such failures.
4. Improved reliability because data may be replicated so that it exists at more than one site, the failure of a node or a communication link does not necessarily make the data inaccessible.
5. Improved performance as the data is located near the site of "greatest demand," and given the inherent parallelism of distributed DBMSs, speed of database access may be better than that achievable from a remote centralized database.

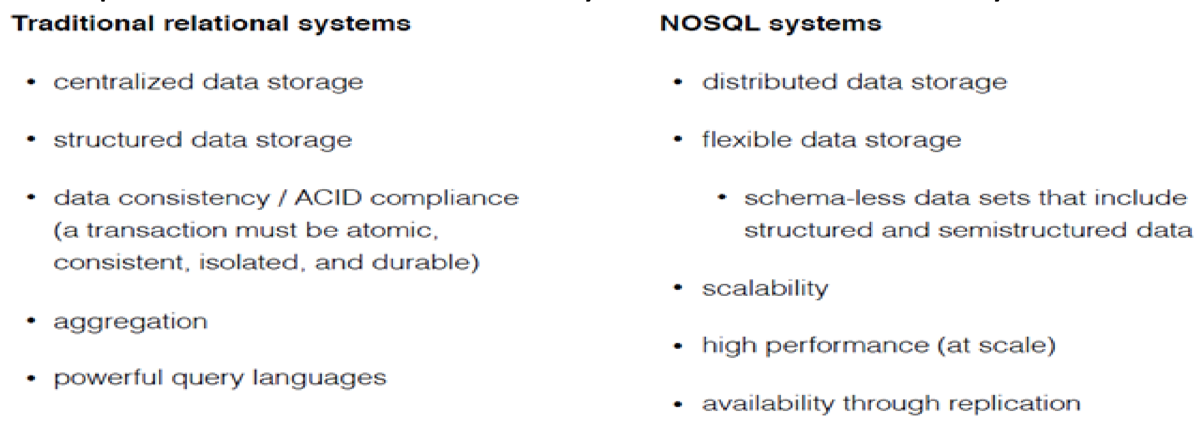
7.2 Given are two relations R_1, R_2, and two distributes sites (Notes) N_1, N_2:
R_1 is locates on N_1 and contains 100 tuples. Each tuple need 100 bytes.
R_2 is locates on N_2 and contains 200 tuples. Each tuple need 200 bytes.
A distributed query joins R_1 with R_2, the result relation contains 50 tuples. Each result tuple needs 50 bytes.
Which possibilities exits have to execute the distributed query when the result is needed at node N_1?
R1= 100 N1=100 R1N1=100*100=10000bytes
R2= 200 N2=200 R2N2=200*200=40000BYTES
RESULT=R1 * R2 = 50bytes * 50bytes = 2500
Strategy 1: Transfer R2 to site1 and perform join at site1;
Strategy 2: Transfer R1 to site2 and perform join at site2 send the result at site1;

How much bytes have to be transferred over the network for each variant?
Startegy1: 40000=40000bytes must be transferred.
Startegy2: 10000+2500=12500bytes must be transferred.

Which variant should be preferred?
Strategy 2 should be preferred.
Query processing use semi join to reduce the number and size of tuples before transferring them to another site

7.3 Why is reducing the amount of network data transfer so important for distributes databases?
Network data transfer is an important cost factor in DDBs, because in most cases it is relatively slow compared to CPU or I/O transfer speeds ➔ Reducing the amount of network data transfer is an important optimization criterion in DDBMS.

8. NOSQL Systems 13(4+4+1+2+2) Points
8.1 Compare the focus of traditional relational systems with the focus of NOSQL systems!



Traditional relational systems	NOSQL systems
• centralized data storage	• distributed data storage
• structured data storage	• flexible data storage • schema-less data sets that include structured and semistructured data
• data consistency / ACID compliance (a transaction must be atomic, consistent, isolated, and durable)	• scalability
• aggregation	• high performance (at scale)
• powerful query languages	• availability through replication

8.2 Explain the CAP theorem!
In a distributed system with data replication only two of the following properties can be guaranteed at the same time:

- Consistency the nodes will have the same copies of a replicated data item visible for various transactions
- Availability each read or write request for a data item will either be processed successfully or will receive a message that the operation cannot be completed
- Partition Tolerance: the system can continue to operate if the network connecting the nodes has a fault that results in two or more partitions, where the nodes in each partition can only communicate among each other

8.3 How are joins realized in most NOSQL systems?
Many NOSQL systems do not provide join operations as part of the query language itself
➔ The Joins need to be implemented in the application programs

8.4 What is Eventual Consistency?
If no new updates are made to a given data item, eventual consistency informally guarantees that eventually all accesses to that item will return the last updated value.

8.5 Which four main categories of NOSQL systems do you know?
Categories of NOSQL Systems
1. Document Stores: Examples: MongoDB
Document-based NOSQL systems store data as collections of similar documents are accessible via their document id
2. Key-value Stores: Example: Amazon Dynamo DB,
Every data item (value) must be associated with a unique key
Retrieving the value by supplying the key must be very fast
3. Wide column Stores: Examples Google distributed storage system (BigTable), Apache, Hbase
4. Graph-Databases: Example: Neo4j
5. Hybrid Systems: Example: Xeround