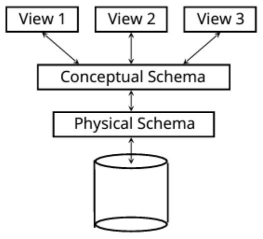
Introduction to databases QUESTIONS

Lecture 1

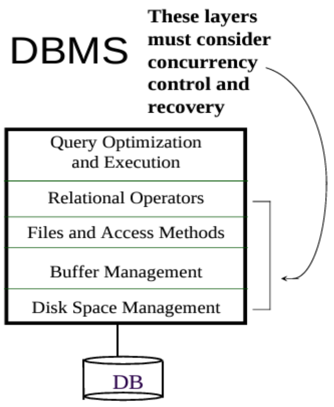
1) **What is DBSM?**
   DBMS is a software package designed to store and manage databases. (very large, integrated collection of data)
2) **Files vs. DBMS?**
   Files involve manual management of data, specialized code, and face challenges like inconsistency and limited crash recovery. DBMS automates data handling, optimizes queries, ensures consistency with multiple users, provides robust crash recovery, and includes built-in security features.
3) **What is the data model?**
   A data model is a collection of concepts for describing data. The relational model of data is the most widely used model today. Main concept: relation, basically a table with rows and columns. Every relation has a schema, which describes the columns, or fields.
   (A data model is a conceptual framework that outlines how data is organized and structured, defining relationships between elements like entities and attributes. It serves as a blueprint for designing databases and information systems.)
4) **What is data independence?**
   Data independence means apps are shielded from knowing how data is stored. It has two aspects:
   1. Logical Data Independence: Protects from changes in the logical structure of data.
   2. Physical Data Independence: Protects from changes in the physical structure of data.

   Also, data independence is one of the most important benefits of using a DBSM!
5) **Describe the levels of abstraction in DBMS.**
   Levels of abstraction in DBMS involve:
   - Views: Describing how users perceive the data.
   - Conceptual Schema: Defining the logical structure.
   - Physical Schema: Describing the files and indexes used.



6) **What is a transaction?**
   A transaction is an atomic sequence of actions on a database, including reads and writes. For a transaction to be considered successful, it must leave the database in a consistent state, assuming the database was consistent at the transaction's start. Users can define basic integrity constraints that the Database Management System (DBMS) enforces, but the DBMS doesn't fully understand the data's semantics (e.g., it does not understand how the interest on a bank account is computed).
7) **Describe the structure of a DBMS.**

The structure of a typical Database Management System (DBMS) follows a layered architecture. Here are the key components:


DBMS
These layers must consider concurrency control and recovery
Query Optimization and Execution
Relational Operators
Files and Access Methods
Buffer Management
Disk Space Management
DB

- Query Optimization and Execution: responsible for optimizing queries and executing them efficiently.
- Relational Operators: it involves operations related to relational databases, such as selections, projections, joins, etc.
- Files and Access Methods: deals with managing files and defining access methods to retrieve data from those files.
- Buffer Management: it involves managing a buffer or cache in memory to store frequently accessed data, optimizing data retrieval.
- Disk Space Management: handles the allocation and deallocation of disk space for the storage of database files.

Lecture 1 – 2

1) **What is the relational database model?**
   The relational model of data is the most widely used model today. Main concept: relation, basically a table with rows and columns. Every relation has a schema, which describes the columns, or fields.
2) **What is the integrity constraint?**
   It is a condition that must be true for any instance of the database. ICs are specified when schema is defined and checked when relations are modified. Integrity constraints: domain, entity, key, reference, unique.
   A legal instance of a relation is one that satisfies all specified ICs. DBMS should not allow illegal instances. If the DBMS checks ICs, stored data is more faithful to real-world meaning. Avoids data entry errors, too!
3) **Describe the concept of a primary key.**
   A primary key is a set of one or more fields (columns) in a database table that uniquely identifies each record (row) in that table. The primary key must satisfy two conditions:
   - Uniqueness: No two distinct records in the table can have the same values in all the primary key fields. This ensures that each record is uniquely identifiable.
   - Irreducibility: The primary key should be irreducible, meaning that no subset of the key fields can guarantee the uniqueness of records. In other words, removing any part of the primary key would violate the uniqueness constraint.
4) **What is the foreign key?**
   Set of fields in one relation that is used to `refer’ to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a `logical pointer’.

A foreign key is a column in one table that refers to the primary key in another table, creating a link between them to enforce referential integrity and establish relationships in a relational database.

5) **What is referential integrity?**
Referential integrity ensures that relationships between tables in a database are maintained by enforcing that foreign key values in one table match primary key values in another, preventing inconsistencies and orphaned records, i.e., no dangling references.

6) **Describe the basic operations of relational algebra.**
Projection: Works on a single relation R; returns a relation, which contains only those attributes (columns) that are defined by the list S. The projection eliminates duplicates.
Selection: Works on a single relation R; returns a relation, which contains only those tuples (rows) from the relation that satisfy a given condition (predicate).
Renaming
Union: Combining rows from two tables without duplicates.
Difference: Retrieving rows from one table that do not exist in another.
Cross Product (Cartesian Product): Generating a combination of all possible pairs of rows between two tables, resulting in a new table with a few rows equal to the product of the original tables' rows.

7) **Describe the additional operations of RA.**
Intersection: Extracting common rows from two tables.
Join: Combining columns from two tables based on a related condition, creating a new, larger table.
Division: The quotient of the relations R and S includes only those tuples which cover the relation S. R/S

8) **Express the additional operations of RA by using the basic operations.**
Division: Disqualified *x* values: $\pi_x((\pi_x(A) \times B) - A)$
A/B: $\pi_x(A)$ - all disqualified tuples
Intersection: R∩S=R−(R−S)
Join: $R \bowtie c\ S = \sigma_C(R \times S)$

9) **Present all variants of the operation join.**
Condition join: $\boldsymbol{\theta}$-joinof the relations R and S is a cartesian product where we keep only those tuples which satisfy the condition θ. $R \bowtie_c S = \sigma_c(R \times S)$
Equi-Join: A special case of condition join where the condition c contains only equalities.
Natural Join: Equijoin on all common fields.
Semijoin of relations R and S is equal to natural join where we keep only the attributes of the left relation R. $R \rhd S$

Outer joins (left, right and full): Left outer join of R and S returns a natural join where tuples of R having no matching values in common attributes of S are also included in the result. Unknown values of the attributes are set to NULL. R⟕S, R⟖S, R⟗S

10) Express the operator divide with other operations of RA.

Idea: For A/B, compute all x values that are not `disqualified' by some y value in B.
x value is disqualified if by attaching y value from B, we obtain an xy tuple that is not in A. Disqualified x values: $\pi_x\big((\pi_x(A) \times B) - A\big)$
A/B: $\pi_x(A)$ all disqualified tuples

11) Describe the syntax of the relational calculus.

Relational Calculus has variables, constants, comparison ops, logical connectives and quantifiers. Expressions in the calculus are called formulas. An answer tuple is essentially an assignment of constants to variables that make the formula evaluate to true. It comes in two flavors

- TRC: Variables range over (i.e., get bound to) tuples.
  Example: {t|P(t)}, where t is a tuple variable and P(t) is a formula.
- DRC: Variables range over domain elements (= field values).
  Example: {t|∃u(R(t,u)∧Q(u))}, where t and u are variables, R is a relation, and Q is a condition.

Both TRC and DRC are simple subsets of first-order logic.

12) Present the general form of a query expressed with the relational calculus.

Query has the form: { T | p(T) } where T is the only free variable. Answer includes all tuples T that make the formula p(T) be true.

13) Describe the bound and free variables in the expressions of the relational calculus.

Bound Variables: Variables quantified by a quantifier (∃ or ∀) in a formula. Binding occurs within the scope of a quantifier. Example: In {t|∃u(P(t,u))}, u is a bound variable.
Free Variables: Variables not explicitly quantified, existing outside the scope of quantifiers. Scope extends across the entire expression. Example: In {t|∃u(P(t,u))}, t is a free variable.

14) How can the division of RA be expressed in relational calculus?

{ P | ∃S∈Sailors ∀B∈Boats (∃R∈Reserves(S.sid=R.sid ∧ R.bid=B.bid ∧ P.name=S.sname)) }
The division operation in relational algebra, denoted as R÷S, can be expressed in the relational calculus. Here's a representation in Tuple Relational Calculus (TRC):
{t|∀u(u∈S⇒∃v(t,u,v)∈R)}. In this expression:

- t represents a tuple variable from the result relation.
- u represents a tuple variable from the relation S.
- v represents a tuple variable from the relation R.
- ∀ denotes the universal quantifier (for all).
- ⇒ denotes implication (if...then).

15) **Explain the concept of a safe query.**

A safe query in a relational database is one that is guaranteed to produce a finite result, preventing the possibility of an infinite result set. This ensures efficient query execution and avoids unintended consequences.

16) **Explain the relational completness of a query language.**

Query languge can express every query that is expressible in relational algerba/calculus. This completeness ensures that the query language is versatile enough to handle a wide range of database queries and manipulations.

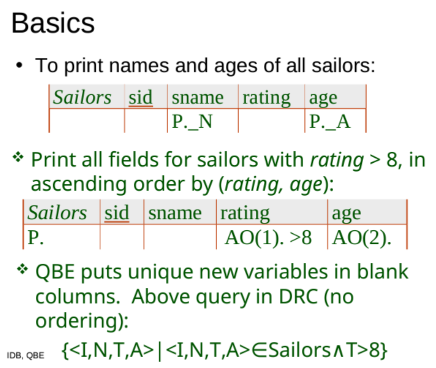
Lecture 3 – 4

1) **Present the basic syntax of SQL query language.**

SELECT [DISTINCT] target-list FROM relation-list WHERE qualification

- relation-list: A list of relation names (possibly with a range-variable after each name).
- target-list: A list of attributes of relations in relation-list
- qualification: Comparisons (Attr op const or Attr1 op Attr2, where op is one of <, >, =, ≠, <=, >= combined using AND, OR and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are not eliminated!

2) **Present the basic syntax of QBE.**



Basics

- To print names and ages of all sailors:

Sailors	sid	sname	rating	age
		P._N		P._A

- Print all fields for sailors with *rating* > 8, in ascending order by (*rating, age*):

Sailors	sid	sname	rating	age
P.			AO(1). >8	AO(2).

- QBE puts unique new variables in blank columns. Above query in DRC (no ordering):

IDB, QBE $\{\langle I,N,T,A\rangle \mid \langle I,N,T,A\rangle \in Sailors \wedge T>8\}$

3) **Describe the concept of join in SQL and QBE.**

SQL Join: In SQL, a join combines rows from different tables based on related columns using keywords like JOIN. Types of joins include INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN.

QBE Join: In QBE, a join is visually represented by linking related fields in different tables using a graphical interface, indicating the relationships between tables.

4) **Present the nested queries in SQL.**

A very powerful feature of SQL: a WHERE clause can itself contain an SQL query! (Actually, so can FROM and HAVING clauses.)

The WHERE clause can contain a subquery to filter rows based on conditions derived from another query's results.
The FROM clause can include subqueries to create derived tables or temporary result sets.
The HAVING clause can utilize subqueries for conditional filtering after grouping.
Subqueries can be used in the SELECT clause to retrieve a single value or a set of values that become part of the result set.
The IN operator can be used with a subquery to check if a value matches any value in the result set of the subquery.

5) **Describe the role of statements [not] exists and [not] unique in SQL.**
Exists: Used to check whether a subquery returns any rows.
Unique: Used to check whether a specified column contains unique values.

6) **How to implement division (RA operation) in SQL and QBE?**
QBE: Visually express conditions ensuring that for every tuple in B, there exists a tuple in R that links to all tuples in A.

7) **Describe the use of aggregation functions in SQL and QBE.**
SQL: Aggregate functions in SQL, such as COUNT, SUM, AVG, MAX, and MIN, operate on sets of values and return a single value. They are often used with the GROUP BY clause to perform calculations on groups of rows.
QBE: aggregation is achieved by visually grouping and summarizing data in a tabular form. While not as explicit as SQL, QBE allows users to design queries by manipulating tables and indicating desired summary operations.

Lecture 4 – 6

1) **Name and describe a few external storage devices.**
Disks: Can retrieve random page at fixed cost. But reading several consecutive pages is much cheaper than reading them in random order.
Tapes: Can only read pages in sequence. Cheaper than disks; used for archival storage.
File organization: Method of arranging a file of records on external storage. Record id (rid) is sufficient to physically locate record. Indexes are data structures that allow us to find the record ids of records with given values in index search key fields
Architecture: Buffer manager stages pages from external storage to main memory buffer pool. File and index layers make calls to the buffer manager.

2) **What is the file organization? What alternatives we have?**
File organization: Method of arranging a file of records on external storage. Record id (rid) is sufficient to physically locate record. Indexes are data structures that allow us to find the record ids of records with given values in index search key fields.
Many alternatives exist, each ideal for some situations, and not so good in others:

- Heap (random order) files: Suitable when typical access is a file scan retrieving all records.
- Sorted Files: Best if records must be retrieved in some order, or only a `range' of records is needed.
- Indexes: Data structures to organize records via trees or hashing. Like sorted files, they speed up searches for a subset of records, based on values in certain ("search key") fields. Updates are much faster than in sorted files.

3) **Describe the concepts of a search key, a data entry, an index entry, and a data record.**

Search key: A field or combination of fields used to search, locate, and retrieve records in a database. It's the attribute based on which data is organized for efficient retrieval.
Data entry: A single piece of information or a set of related information entered a database. It represents a unit of data, such as a record or a field value.
Index entry: A data structure that maps a search key to the location of corresponding data in a file or database. It facilitates faster data retrieval.
Data record: A collection of related data fields treated as a unit. It represents a complete set of information about an entity (e.g., person, product, or event).

4) **What are the alternatives for a data entry k*?**

In a data entry k* we can store:

- Data record with key value k, or
- <k, rid of data record with search key value k>, or
- <k, list of rids of data records with search key k>

Choice of alternative for data entries is orthogonal to the indexing technique used to locate data entries with a given key value k.

5) **What is a primary/secondary index? What is a clustered/unclustered index?**

If the search key contains primary key, then called primary index, ensuring a unique and efficient means of accessing records.
Secondary Index: The search key contains a candidate key (not necessarily the primary key), providing an alternative access path to data.
Clustered Index: The order of data records mirrors or is close to the order of data entries in the index. It directly affects the physical order of data in the table.
Unclustered Index: The order of data records is independent of the order of data entries in the index. It provides an alternative access path without influencing the physical order of data.

6) **Present ISAM indexes.**

File creation: Leaf (data) pages allocated sequentially, sorted by search key; then index pages allocated, then space for overflow pages.

Index entries: <search key value, page id>; they `direct' search for data entries, which are in leaf pages.
Search: Start at root; use key comparisons to go to leaf.
Cost $\log_F N$; F = # entries/index pg, N = # leaf pgs
Insert: Find leaf data entry belongs to and put it there.
Delete: Find and remove from leaf; if empty overflow page, de-allocate.
Static tree structure: inserts/deletes affect only leaf pages.

7) **Present B+ tree index.**
Minimum 50% occupancy (except for root). Each node contains d <= m <= 2d entries. The parameter d is called the order of the tree. Supports equality and range-searches efficiently.

8) **Describe the B+ tree operations insert and delete.**
<u>Insert</u>:
- Find correct leaf L.
- Put data entry onto L.
  - If L has enough space, done!
  - Else, must split L (into L and a new node L2)
    - Redistribute entries evenly, copy up middle key.
    - Insert index entry pointing to L2 into parent of L.
- This can happen recursively.
  - To split index node, redistribute entries evenly, but push up middle key. (Contrast with leaf splits.)
- Splits "grow" tree, root split increases height.
  - Tree growth: gets wider or one level taller at top.

<u>Delete</u>:
- Start at root, find leaf L where entry belongs.
- Remove the entry.
  - If L is at least half-full, done!
  - If L has only d-1 entries,
    - Try to re-distribute, borrowing from sibling (adjacent node with same parent as L).
    - If re-distribution fails, merge L and sibling.
- If merge occurred, must delete entry (pointing to L or sibling) from parent of L.
- Merge could propagate to root, decreasing height.

9) **Describe hash-based indexes. What are the alternatives?**
Hash-based indexes are best for equality selections. Cannot support range searches. Static and dynamic hashing techniques exist.

Lecture 7-8

Query evaluation

1) **What is the access method? What kind of access methods do you know?**
   An access method in computer science refers to the technique used to retrieve or store data. Common types include: sequential access, random access, ISAM (Indexed Sequential Access Method), hashing, B-Tree and B+ Tree, clustered and non- clustered index.
2) **Describe common techniques used for the evaluation of relational operations.**
   Algorithms for evaluating relational operators use some simple ideas extensively:
   - Indexing: Can use WHERE conditions to retrieve small set of tuples (selections, joins)
   - Iteration: Sometimes, it is faster to scan all tuples even if there is an index. (And sometimes, we can scan the data entries in an index instead of the table itself.)
   - Sorting: Many algorithms for the evaluation of relational operations evolved from the external merge sort algorithm: project, join, grouping, etc.
   - Partitioning: By hashing, we can partition the input tuples and replace an expensive operation by similar operations on smaller inputs.
3) **Present the general external merge sort algorithm. What is the complexity of the merge sort?**
   External merge sort typically uses a hybrid sort-merge strategy. In the sorting phase, chunks of data small enough to fit in main memory are read, sorted, and written out to a temporary file. In the merge phase, the sorted subfiles are combined into a single larger file. Time complexity O(nlogn)
4) **How to implement the selection operation?**
   Two Approaches to General Selections
   - First approach: Find the most selective access path (an index or file scan that we estimate will require the fewest page I/Os), retrieve tuples using it, and apply any remaining terms that don't match the index.
   - Second approach: Get sets of rids of data records using each matching index. Then intersect these sets of rids. Retrieve the records and apply any remaining terms.
5) **Present the methods for the implementation of the projection.**
   ```
   SELECT   DISTINCT R.sid, R.bid
   FROM     Reserves R
   ```
   - Sorting Approach: Sort on <sid, bid> and remove duplicates. (Can optimize this by dropping unwanted information while sorting.)
   - Hashing Approach: Hash on <sid, bid> to create partitions. Load partitions into memory one at a time, build in-memory hash structure, and eliminate duplicates.
6) **Describe the nested loops join, the index nested loops join, and the block nested loops join.**
   - Simple nested loops join: For each tuple in the outer relation, we scan the entire inner relation.

- Index nested loops join: Selects one table as the outer loop and uses an index on its join key. The algorithm iterates through each row of the outer table, using the index to efficiently locate matching rows in the inner table, satisfying the join condition. This method is particularly effective when one table is small or has a selective index, minimizing the overall iteration and enhancing the join performance.
- Block nested loops join: Each block of the inner relation is paired with each block of the outer relation.

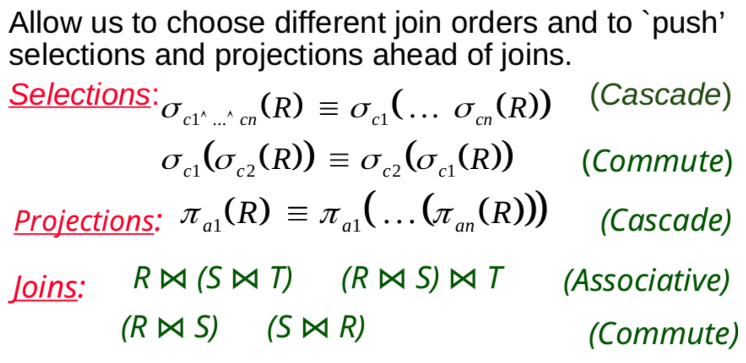
7) **Present the sort-merge join algorithm.**
The Sort-Merge Join algorithm works by first sorting two input datasets based on the join column. Once sorted, it iterates through both datasets in parallel, comparing the values of the join key. Matching rows are then combined, and output result tuples.

8) **Describe the hash-based join algorithm.**
Hash join is used when projections of the joined tables are not already sorted on the join columns. In this case, the optimizer builds an in-memory hash table on the inner table's join column. The optimizer then scans the outer table for matches to the hash table, and joins data from the two tables accordingly.

Query optimization

1) **How to estimate the cost of a query?**
The result is always an estimation. Statistics is used for estimating the cost.
You need to know the size of the data processed by the query, in bytes and the pricing model you are using. Also, consider the cost of CPU and I/O.
Cost estimation function should consider: Number of blocks read from disk, CPU used for computation of RA operators (we ignore CPU cost), Storing intermediate temporary tables, The amount of RAM used in particular algorithm.

2) **Present the relational algebra equivalences?**



Allow us to choose different join orders and to `push' selections and projections ahead of joins.

Selections: $\sigma_{c1 \wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\ldots \sigma_{cn}(R))$ *(Cascade)*

$\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ *(Commute)*

Projections: $\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{an}(R)))$ *(Cascade)*

Joins: $R \bowtie (S \bowtie T)$ $(R \bowtie S) \bowtie T$ *(Associative)*

$(R \bowtie S)$ $(S \bowtie R)$ *(Commute)*

3) **How to obtain all equivalent query expressions for a given query expressed in relational algebra?**
This basically means applying various algebraic rules and properties to manipulate the original expression while maintaining its equivalence. Using rules like ¬(¬A) = A or (A ∪ B) to join (A ⋈ U) when appropriate, (A ∩ B) as set difference (A - (A - B)…

4) **Describe the procedure for the evaluation of the alternative query plans.**
The procedure for evaluating alternative query plans involves estimating the cost of execution, comparing factors like CPU usage, I/O operations, and index utilization, and selecting the plan with the lowest overall cost to optimize query performance.

5) **Describe cost estimation for single-relation query plans.**
The cost estimation for single-relation query plans involves assessing the computational resources required for various scenarios, including those involving indexes and table scans. Here are specific considerations for different situations: index I on primary key matches selection, clustered index I matching one or more selects, non-clustered index I matching one or more selects, sequential scan of file.

6) **Describe left-deep and bushy join trees?**
- Left-Deep Join Tree: In a left-deep join tree, the query execution plan forms a linear or "left-deep" structure. The tables are joined sequentially, and each join operation involves a pair of tables. This structure is characterized by a single path from the leftmost table to the rightmost table in the join sequence. Left-deep join trees are easy to construct and maintain, making them computationally efficient, but they may not exploit parallelism as effectively as other structures.
- Bushy Join Tree: In contrast, a bushy join tree, also known as a "bushy" or "bushy-tailed" join tree, is a more complex structure where tables are joined in a non-linear fashion. Multiple joins can occur simultaneously, creating a branching structure. This allows for parallelism in query execution, as different branches of the tree can be processed concurrently. Bushy join trees are more flexible and can potentially result in more efficient query plans, especially for queries involving multiple joins. However, constructing and optimizing bushy join trees can be computationally expensive compared to left-deep join trees.

7) **Describe the procedure for the enumeration of the left-deep query plans.**
Left-deep plans differ only in the order of relations, the access method for each relation, and the join method for each join.
Enumerated using N passes (if N relations joined):
- Pass 1: Find best 1-relation plan for each relation.
- Pass 2: Find best way to join result of each 1-relation plan (as outer) to another relation. (All 2-relation plans.)
- Pass N: Find best way to join result of a (N-1)-relation plan (as outer) to the N'th relation. (All N-relation plans.)

For each subset of relations, retain only cheapest plan overall, plus cheapest plan for each interesting order of the tuples.

Lectures 9-11

Concurrency control

1) **What is the transaction?**
   A transaction is the DBMS's abstract view of a user program: a sequence of reads and writes.
2) **Explain possible anomalies of the interleaved execution of transactions.**
   Reading Uncommitted Data (WR Conflicts, "dirty reads")
   Unrepeatable Reads (RW Conflicts)
   Overwriting Uncommitted Data (WW Conflicts)
3) **What is a conflict serializable schedule of a transaction?**
   A schedule is conflict serializable if it can be transformed into a serial schedule by swapping the order of non-conflicting operations.
4) **Describe the strict and non-strict two-phase locking protocols.**
   The strict two-phase locking protocol requires transactions to hold all their locks until the transaction commits. No locks can be released before this point. In contrast, the non-strict two-phase locking protocol allows a transaction to release its locks before reaching the commit point, but the transaction can not request additional locks once it releases any locks.
5) **What happens when a transaction is aborted?**
   If transaction Ti is aborted, then all the actions must be undone. (Not only this but if Tj reads objects written by Ti, then Tj must also be aborted!). Most systems avoid cascade aborts by releasing locks just before the commit.
6) **Describe the deadlock prevention methods.**
   Assign priorities based on timestamps. Assume Ti wants a lock that Tj holds. Two policies are possible:
   - Wait-Die: If Ti has higher priority; Ti waits for Tj; otherwise, Ti aborts.
   - Wound wait: If Ti has higher priority, Tj aborts; otherwise, Ti waits.

   If a transaction re-starts, make sure it has its original timestamp.
7) **What is index locking? How is it implemented?**
   Index locking refers to the mechanism in a database management system where locks are applied specifically to index structures, such as B-tree or hash indexes, to control access and modifications to the index. Index locking is implemented by acquiring and releasing locks on the index nodes or pages, preventing multiple transactions from conflicting access to the same part of the index simultaneously.
8) **Describe optimistic concurrency control.**
   Optimistic concurrency control is a strategy in database management systems where multiple transactions are allowed to proceed concurrently without acquiring locks during the read phase. Instead, each transaction reads the data it needs and performs its

operations without locking resources. However, during the write or commit phase, the system checks if any conflicts have occurred by comparing the values read and the current values in the database. If conflicts are detected, the system may abort and rollback some transactions. Optimistic concurrency control is based on the optimistic assumption that conflicts will be infrequent, aiming to maximize concurrency and minimize the need for locking.

Crash recovery

1) **Explain the ACID properties of a relational DBMS.**
   Atomicity. The transaction is either executed or not.
   Consistency. The database must be consistent before and after the transaction.
   Isolation. Concurrently executed transactions do not affect each other.
   Durability. Committed transaction changes are stored persistently, ensuring they survive system failures and can be recovered even after a crash.
2) **Explain the trade-offs of "stealing" pages from the buffer pool and "forcing" pages to be stored to a disk.**
   "Stealing" pages allows for efficient memory use by replacing pages in the buffer pool even if modified, but it may result in increased write operations during memory pressure. On the other hand, "forcing" pages promptly writes modifications to disk, ensuring data durability but potentially leading to suboptimal memory utilization. The decision between these approaches depends on system priorities, with a focus on performance optimization favoring "stealing" for better memory use or a preference for data durability favoring "forcing" to ensure timely writes to disk and mitigate the risk of data loss in case of system failure.
3) **Describe the Write-Ahead Logging (WAL) protocol.**
   Write-ahead logging (WAL) is used to undo the actions of aborted transactions and to restore the system to a consistent state after a crash.
   Must force the log record for an update before the corresponding data page gets to disk. Must write all log records for a Xact before commit. #1 guarantees Atomicity. #2 guarantees Durability.
   The Write-Ahead Logging (WAL) protocol ensures database transaction durability by writing log records to a separate file on disk before modifying actual database pages, allowing for recovery in case of system failures.
4) **Describe what data is stored and where it is stored for the execution of the WAL protocol.**
   For the execution of the Write-Ahead Logging (WAL) protocol, log records containing transactional changes, such as operations and before-and-after values, are stored in a separate log file on disk, ensuring durability and supporting recovery in case of system failures.

5) **Present the analysis, redo and undo phases of the crash recovery.**
   - Analysis: Scan the log forward (from the most recent checkpoint) to identify all Xacts that were active, and all dirty pages in the buffer pool at the time of the crash.
   - Redo: Redoes all updates to dirty pages in the buffer pool, as needed, to ensure that all logged updates are in fact carried out and written to disk.
   - Undo: The writes of all Xacts that were active at the crash are undone (by restoring the before value of the update, which is in the log record for the update), working backwards in the log. (Some care must be taken to handle the case of a crash occurring during the recovery process!)

Lecture 12

1) **Why can redundancy appear in relational databases?**
   Redundancy can appear in relational databases due to data duplication, denormalization practices, and inadequate database design, impacting storage efficiency and data consistency.
2) **What is a functional dependency?**
   A functional dependency in a relational database indicates that the values of one set of attributes uniquely determine the values of another set.
3) **How can we reason about the functional dependencies?**
   We reason about functional dependencies in a relational database by understanding how the values of certain attributes uniquely determine the values of other attributes, helping to ensure data integrity and reduce redundancy.
4) **What is the purpose of the normalization of a relation? What is the normal form?**
   Normalization is the process of transforming relational schemes or corresponding relations into a form in which we cannot get anomalies.
   The purpose of normalizing a relation is to reduce data redundancy and dependency by organizing it efficiently, ensuring data integrity. If a relation is in a certain normal form (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.
5) **Present the Boyce-Codd normal form of a relation.**
   The Boyce-Codd Normal Form (BCNF) ensures that every non-trivial functional dependency within a relation is determined by a superkey, eliminating certain anomalies in a database.
6) **Present the 3rd normal form (3NF) of a relation.**
   The Third Normal Form (3NF) ensures that data is free from transitive dependencies (when the value of one non-prime attribute uniquely determines the value of another non-prime attribute) by eliminating non-prime attributes that depend on other non-prime attributes within a relation.

7) **What is a lossless-join decomposition of a relation?**
   Lossless join decomposition ensures that you can perform joins between the smaller tables to reconstruct the original table without any data loss. This is typically achieved through primary keys and foreign keys that maintain referential integrity.
8) **What is dependency preserving decomposition of a relation?**
   Dependency Preserving Decomposition is a technique used to decompose a relation into smaller relations while preserving the functional dependencies between the attributes.
9) **Present the algorithm for the decomposition of a relation into the BCNF.**
   Check notebook.
10) **Present the algorithm for the decomposition of a relation into the 3NF.**
   Check notebook.

Questions from exams

1) **Describe the internal representation of a relational table in main memory and on disk.**
   Table is a heap file composed of a set of pages.
   Possible representations of a heap file is double-linked list or directory.
   Table records are stored in pages.
   Pages are always read in a buffer pool.
   Records are accessed from the pages in the buffer pool.
2) **Describe the bulk loading algorithm for B+ trees. What are the differences to using a simple B+ tree add operation?**
   First sort the records.
   Allocate contiguous disk space to store the leaf level.
   Link a new root node to first leaf page.
   Insert index entries (in order) from the leaf pages to
   the rightmost index pages above the leaf level.
   Simple add is updating different parts of B+ tree.
3) **Describe the cardinality of a relationship in the data model ER.**
   Min-max notation:
   card(E,R) = (min,max), where min is minimal number of relationship R instances where an entity E instance is involved, and max is the maximal number of relationship R instances where an entity E instance is involved.
4) **Present the meaning of integrity constraint. Describe each type of integrity constraint in one sentence.**
   Integrity constraints specify the conditions that have to be fulfilled for each row of a relation.
   Primery key: unique value for each table row.
   Foreign key: reference to the primary key in other table.

Value constraints: attribute value is constrained.

NULL constrains: allow NULL/require NOT NULL.

5) **Describe the alternative implementations of the relational operation PROJECT.**

The expensive part is removing duplicates.

Modify external sorting to eliminate duplicates; faster than sorting.

Use hash function to partition a relation.

Read partitions in a buffer and eliminate duplicates.

Write out unique values. Read next partition.

6) **Present the main ideas of optimistic concurrency control.**

Locking is not used.

Xact speculates no other Xact will interfere on commit.

Xact execute reads. Writes are done on copies of pages.

Validation phase checks if no other Xact interferes and then executes writes.

Each Xact has lists of object that it reads and updates. Used for validation.