Lecture 1: Introduction

1) What is DBMS?
- Database Management System is a software package designed to maintain, store and manage large collections of data - databases. It is a collection of programs that enables its users to access databases, manipulate data, and help in the representation of data.

2) Files vs. DBMS?
- Large datasets between main memory and secondary storage
- Special code for different queries (efficient query processing)
- Protect data since there are multiple users – protection of inconsistent changes when it's accessed concurrently
- Crash recovery and backup
- Security and access control
- Efficient data access
- Data Integrity and Security
- Data Independence

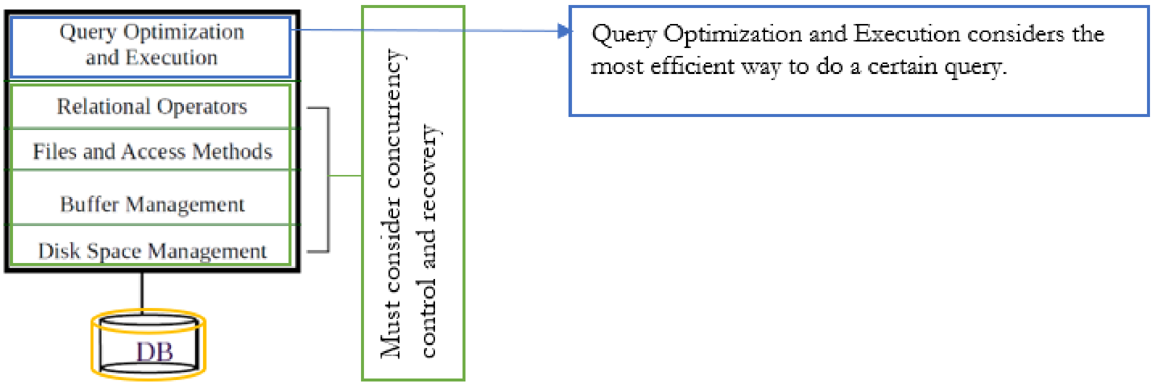
3) What is the data model?
Collection of concepts for describing and organizing data
Gives an idea how the final system would look like after its complete implementation
Defines the data elements and the relations between them
Used to show how data is stored, connected, accessed, and updated in the DBMS
Some data models: Hierarchical, Network, Entity-Relationship, Object-oriented, Object-Relational…
Most widely used: Relational data model (uses relations (tables))

4) What is data independence?
The ability to change one of the schemas (logical/physical) without affecting the other one. It is one of the most important benefits of DBMS. Application programs should not, ideally, be exposed to details of data representation and storage, The DBJVIS provides an abstract view of the data that hides such details.

5) Describe the levels of abstraction in DBMS.
- Physical level/schema – the lowest level of abstraction, describes how the data is actually stored on the physical storage (complex low-level data structures)
- Logical (conceptual) level/schema – next, a higher level of abstraction, describes what data will be stored in the database and the relationships between those data (a small number of relatively simple structures)
- View level – highest level of abstraction, describes how users see the data, full database not available (users don't need it)

6) What is a transaction?
Execution of a DB program. The atomic sequence of DB actions (reads/writes). A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

7) Describe the structure of a DBMS.
The DBMS has a layered architecture (each system has its own variation).


Query Optimization and Execution
Relational Operators
Files and Access Methods
Buffer Management
Disk Space Management
DB
Must consider concurrency control and recovery
Query Optimization and Execution considers the most efficient way to do a certain query.

Buffer Management brings pages from the physical disk to the main memory.

Lecture 2: Relational Data Model

1) What is the relational database model?
Relational database: a set of relations.
Relation: made up of 2 parts:
Instance: a table, with rows and columns.
Schema: specifies the name of the relation, plus the name and type of each column.
Can think of a relation as a set of rows or tuples.

2) What is the integrity constraint?
The condition that must be true for any instance of the database (ex. Domain constraints).

3) Describe the concept of a primary key.
A candidate key has been chosen to be the main key for the relation. If you know the value of the primary key you will be able to uniquely identify a single row within the table.

4) What is the foreign key?
It is used to establish relationships between tables. It is a primary key in the second table.

5) What is referential integrity?
All foreign key constraints are enforced i.e. every value of a foreign key must match a value of an existing primary key.

Lecture 3: Relational Algebra & Relational Calculus

1) Describe the basic operations of relational algebra.
- Selection (σ) - selects a subset of rows from the relation that satisfy the selection condition.
- Projection (Π) - Deletes unwanted columns from relation, eliminate duplicates
- Cross-product (x) - Allows us to combine two relations. It is a relation that contains one tuple for each pair of tuples from the relations R and S. No connection between relations is necessary.
- Set-difference (/) - Tuples in relation 1, but not in relation 2.
- Union (U) - Tuples in relation 1 and in relation 2.

2) Describe the additional operations of RA.
- Intersection – Tuples that are in both relations.
- Join - combine two relations based on matching attributes
- Division A/B - A/B contains all x tuples such that for every y tuple in B, there is an XY tuple in A.
- Renaming – ϱ s(S)(R), R is a relation to be renamed, s is a new relation name, S is a list of new attribute names.

3) Express the additional operations of RA by using the basic operations.
- Intersection – R ∩ S = R – (R – S)
- Join – R ⋈c S = σc (R x S)
- Division - R/S = x ((x (A)B) A)
- Renaming – ρ(R, A ⋈ B) or R = A ⋈ B

4) Present all variants of the operation join.
- Condition/Theta join - $\boldsymbol{\theta}$-join of the relations R and S is a cartesian product where we keep only those tuples that satisfy the condition θ.
- Equijoin of the relations R and S is a θ-join, where condition θ= contains only equalities.
- Natural join of the relations R and S is an equijoin overall common attributes, where we keep only one occurrence of the common attributes (no duplicate attributes).

- Semijoin of relations R and S is equal to natural join where we keep only the attributes of the left relation R.
- 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. Similarly goes for right outer join and full outer join.

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

- R/S = x ((x (A)B) A)

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

Query has the form: {<x1,x2,...,xn> | p <x1,x2,...,xn>} . Answer includes all tuples <x1,x2,...,xn> that make the formula p <x1,x2,...,xn> be true.

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

The use of quantifiers and in a formula is said to bind X. A variable that is not bound is free.

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

The division is easily expressed in the calculus. The calculus query for “Find the names of sailors who have reserved all boats” directly reflects how we might express the query in English: "Find sailors S such that for all (⍰) boats B there is (⍰) a Reserves tuple showing that sailor S has reserved boat B."

9) Explain the concept of a safe query.

It is possible to write syntactically correct calculus queries that have an infinite number of answers! Such queries are called unsafe. It is known that every query that can be expressed in relational algebra can be expressed as a safe query in DRC / TRC; the converse is also true.

10) Explain the relational completeness of a query language.

Query language (e.g., SQL) can express every query that is expressible in relational algebra/calculus.

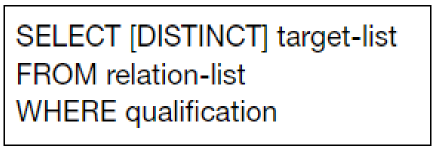
11) Describe the syntax of the relational calculus.

Calculus has variables, constants, comparison ops, logical connectives, and quantifiers.

- Tuple relational calculus (TRC): Variables range over (i.e., get bound to) tuples.
- Domain relational calculus (DRC): Variables range over domain elements (= field values).
- Both TRC and DRC are simple subsets of first-order logic.

Lecture 4: Query Language SQL & QBE

1) Present the basic syntax of SQL query language.



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

- Target-list - A list of attributes of relations in relation-list
- Relation-list - A list of relation names (possibly with a range-variable after each name)
- Qualification – Comparisons

- DISTINCT is an optional keyword indicating that the answer should not contain duplicates.

2) Present the basic syntax of QBE.

A user writes queries by creating example tables.

- QBE uses domain variables, as in the DRC, to create example tables.
- The domain of a variable is determined by the column in which it appears, and variable symbols are prefixed with an underscore (_) to distinguish them from constants.
- The fields that should appear in the answer are specified by using the command P., which stands for print.

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

SQL: The FROM clause says to merge data from table1 with that from the table2. The ON says how to figure out which rows in table1 go with which rows in table2 - the id from table1 must match the id from table2.

QBE: Joins accomplished by repeating variables in both tables.

4) Present the nested queries in SQL.

The result of a SELECT statement may be used as a value in another statement. The subquery may return more than one result. If this happens in the WHERE statement, the query above will fail as you are testing one value against more than one value. It is safer to use IN to cope with this possibility. So we can see the output of the subquery as a set.

5) Describe the role of statements [not] exists and [not] unique in SQL.

- UNIQUE checks for duplicate tuples. The UNIQUE predicate evaluates to True only if all the rows that its subquery returns are unique. NOT UNIQUE evaluates to true if there are duplicates.

- EXISTS is another set comparison operator, like IN. The EXISTS predicate evaluates to True only if the subquery returns at least one row. NOT EXISTS evaluates to true if the subquery has nothing to return.

6) How to implement division (RA operation) in SQL and QBE?
SQL: We can do it using EXCEPT
QBE: it can be implemented with aggregates or update operations; using aggregates: G. for grouping the rows we want and using the Conditions box to filter rows where COUNT._B1 = COUNT._B2.

7) Describe the use of aggregation functions in SQL and QBE.
- SQL: (A is a single column)
- COUNT (*) – count all tuples
- COUNT ([DISTINCT] A) – count unique values of the attribute A
- SUM ([DISTINCT] A) – sum unique values of the attribute A
- AVG ([DISTINCT] A) – get the average of the unique values of the attribute A
- MAX (A) / MIN (A) – get max/min value of the unique values of the attribute A
- GROUP BY grouping-list - A group is a set of tuples that have the same value for all attributes in a grouping-list.

QBE: AVG, COUNT, MIN, MAX, SUM (None of these eliminate duplicates, except COUNT). Also, have AVG.UNQ. etc. to force duplicate elimination
We use them with . and the name of the column. Ex. P.AVG._A - print the average value of column A. We have grouping with G.

Lecture 5: Disks, Files, Indexes(Tree, Hash-Based)

1) Name and describe a few external storage devices.
- Hard Disk Drives (HDDs) - stores and retrieves digital data using magnetic storage and one or more rigid rapidly rotating platters coated with magnetic material. The platters are paired with magnetic heads, usually arranged on a moving actuator arm, which read and write data to the platter surfaces.
- Solid State Disks (SSDs) - uses NAND Multi-Level Cell (2-bit/cell) flash memory, no moving parts (no rotate/seek motors), eliminates seek and rotational delay
- RAID – Redundant Array of Independent Disks - arrangement of several disks that gives abstraction of a single, large disk. Increase performance and reliability.
- Disks are the most important external storage devices. They allow us to retrieve any page at a (more or less) fixed cost per page. However, if we read several pages in the order that they are stored physically, the cost can be much less than the cost of reading the same pages in random order.
- Tapes are sequential access devices and force us to read data one page after the other. They are mostly used to archive data that is not needed on a regular basis.

2) What is the file organization? What alternatives we have?
File Organization refers to the logical relationships among various records that constitute the file, particularly with respect to the means of identification and access to any specific record. In simple terms, Storing the files in a certain order is called file Organization. Alternatives:

- Sequential File Organization
- Heap File Organization
- Hash File Organization
- B+ Tree File Organization
- Clustered File Organization

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

- Search key: The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.
- Data entry vs data record: We use the term data entry to refer to the records stored in an index file. A data entry with search key value k, denoted as contains enough information to locate (one or more) data records with search key value k. We can efficiently search an index to find the desired data entries, and then use these to obtain data records (if these are distinct from data entries).
- Index entry: `direct' search for data entry

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

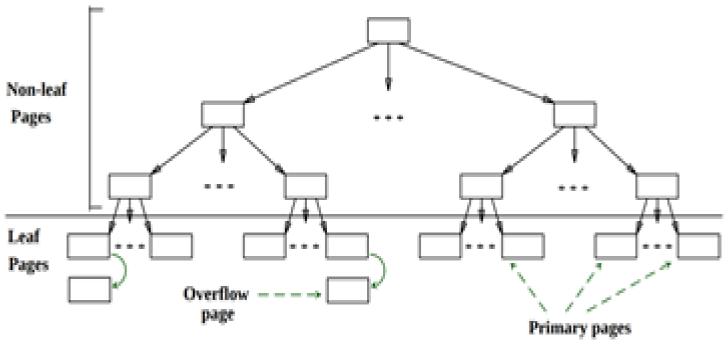
- A data entry *k** is an actual data record (with search key value *k*).
- A data entry is a <*k, rid*> pair, where *rid* is the record id of a data record with search key value k.
- A data entry is a <*k, rid-list*> pair, where *rid-list* is a list of record ids of data records with search key value *k*.

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

- If the index is created on the basis of the primary key of the table, then it is known as primary indexing. All others are secondary. Also, it is known that if the index is using alternative 1 for data entry it is primary, and if it uses alternative 2 or 3, it is secondary.
- When a file is organized so that the ordering of data records is the same as or close to the ordering of data entries in some index, we say that the index is **clustered**; otherwise, it clusters in an **unclustered** index. An index that uses Alternative (1) is clustered, by definition. An index that uses Alternative (2) or (3) can be a clustered

index only if the data records are sorted on the search key field. Otherwise, the order of the data records is random, defined

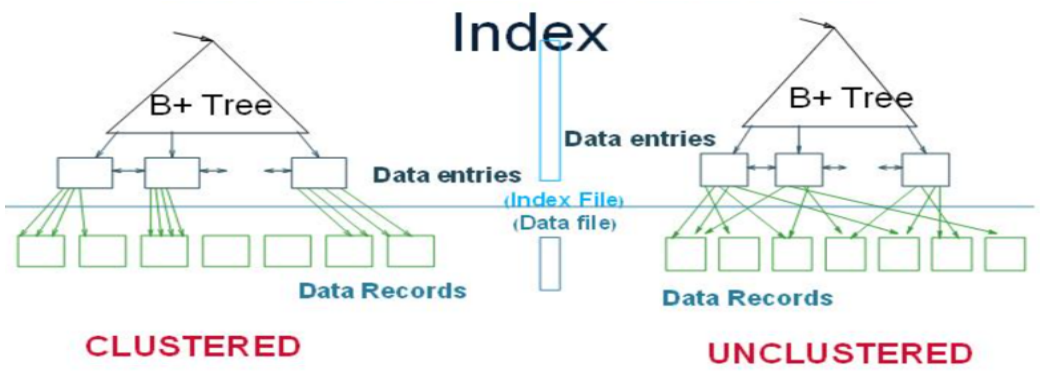
6) Present ISAM indexes.
Static tree structure: inserts/deletes affect only leaf pages.



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>, `direct' search for data entries, which are in leaf pages.

- Search: Start at the root; use key comparisons to go to leaf.
  Cost logF N ; F = # entries/index pg, N = # leaf pgs
- Insert: Find leaf data entry belongs to and put it there.
- Delete: Find and remove from the leaf; if empty overflow page, deallocate.

7) Present B+ tree index.



- Dynamic = The structure is adjusted to the changes in the file. The insert and delete operations keep the tree balanced. Used for ranged search?
- The data entries are arranged in sorted order by search key value and a hierarchical search data structure is maintained that directs searches to the correct page of data entries. Each node is a physical page and retrieving a node involves a disk I/O.
- The lowest level of the tree is called the leaf level and contains the data entries. All searches begin at the root and the contents of pages in a non-leaf level direct searches to the correct leaf page. Non-leaf nodes also contain pointers separated by search key values meaning the node pointer to the left of a key k points to a subtree that contains only data entries less than k and the right one, data entries >= k. All leaf pages are maintained in a doubly-linked-list.
- Condition of the node: $d \le m \le 2d$ (at least 50% full)
  - m … number of records (occupancy) of the node

- d … order of the tree (2d = capacity of the node)
- For the root: 1 ≤ m ≤ 2d

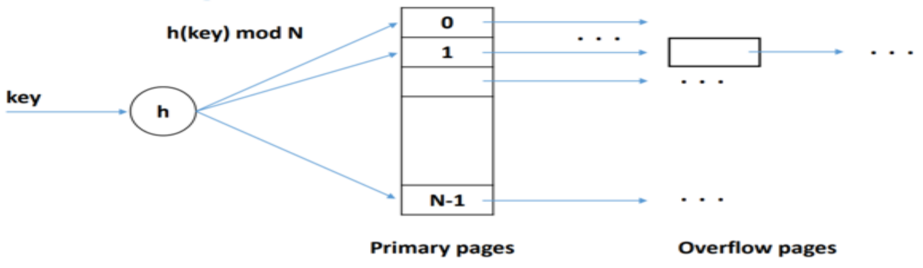
8) Describe the B+ tree operations insert and delete.

- Insert:
  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 the middle key.
  Insert index entry pointing to L2 into the parent of L.
  • This can happen recursively
  To split index node, redistribute entries evenly, but push up the middle key. (Contrast with leaf splits.)
- Delete:
  Start at the 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,

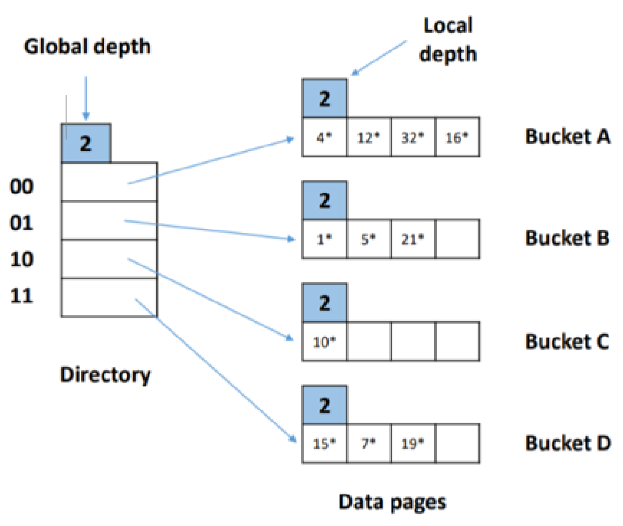
  • Try to re-distribute, borrowing from a sibling (adjacent node with the same parent as L).
  • If re-distribution fails, merge L and sibling.
  • If a merge occurred, we must delete the entry (pointing to L or sibling) from the parent of L.

9) Describe hash-based indexes. What are the alternatives?
As for any index, 3 alternatives for data entries k*:
– Data record with key-value k
– <k, rid of the data record with search key value k>
– <k, list of rids of data records with search key k>
Hash-based indexes are best for equality selections. Cannot support range searches.
-Static indexes can lead to long overflow chains. The pages containing the data can be viewed as a collection of buckets, with one primary page and possibly additional overflow pages per bucket. We apply a hash function h to identify to which bucket the key belongs.



-Dynamic:

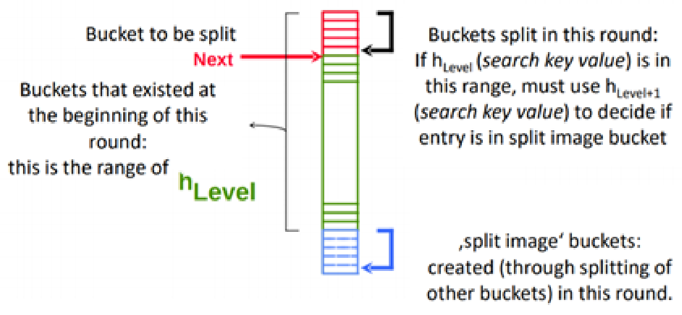
- Extendible Hashing scheme, which doesn‘t use overflow pages.



- Directory with pointers to buckets
  - Number of buckets is doubled by doubling the directory;
  - Split only the the bucket that overflows;
    - Double directory if necessary.
  - Adjust the hash function if needed;
- Most of the times spliting the bucket does not demand doubling the directory (compare local and global depth);

- How do we know to which bucket the entry belongs to?
  - Binary format of new entry;
  - Last two bits tell us the appropriate bucket.

- Linear Hashing scheme, which rarely has more than two overflow pages in a chain.


Bucket to be split
Next
Buckets that existed at the beginning of this round:
this is the range of h_{Level}
Buckets split in this round:
If h_{Level} (search key value) is in this range, must use $h_{Level+1}$ (search key value) to decide if entry is in split image bucket
‚split image‘ buckets: created (through splitting of other buckets) in this round.

- Use family of hash functios h_0, h_1, h_2,...
- Select hash function *h* and number of buckets *N:*
  - $h_i(value)=h(value)mod(2^iN)$
  - d_0 – *number of bits needed to represent N*
  - $d_i=d_0+i$
- Round: Level
  - Use only functions h_{Level} and $h_{Level+1}$
  - Buckets are doubled in every round – one by one.
  - With *Next* we denote the bucket, which will be split next.

Lecture 6: Query Evaluation

1) What is the access method? What kind of access methods do you know?

An access path is a method of retrieving tuples:

- File scan
- Index that matches a selection (in the query):
  - A tree index matches (a conjunction of) terms that involve only attributes in a prefix of the search key
  - A hash index matches (a conjunction of) terms that has a term attribute = value for every attribute in the search key of the index.

2) Describe common techniques used for the evaluation of relational operations.

- Indexing: Can use WHERE conditions to retrieve a small set of tuples (selections, joins)

- Iteration: Sometimes, 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.)
- Partitioning: By using sorting or 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 sorting refers to sorting algorithms that can handle large amounts of data. The general merge sort algorithm sorts a file with N pages using B buffer pages. The sorting produces ceil(N/B) sorted runs of B pages each in Pass 0. In Pass 2 (B-1) runs are merged and sorted.
The number of passes is: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
The final cost is 2N (N = pages) * number of passes

4) How to implement the selection operation?
Selection (σ) selects a subset of rows from relation. Selection conditions are first converted to conjunctive normal form (CNF): ex. (day<8/9/94 OR bid=5 OR sid=3) AND (rname=‘Paul’ OR bid=5 OR sid=3).
- First approach: Find the most selective access path, retrieve tuples using it, and apply any remaining terms that don’t match the index: Most selective access path: An index or file scan that we estimate will require the fewest page I/Os. Terms that match this index reduce the number of tuples retrieved; other terms are used to discard some retrieved tuples, but do not affect the number of tuples/pages fetched.
- The second approach (if we have 2 or more matching indexes that use Alternatives (2) or (3) for data entries): Get sets of rids of data records using each matching index. Then intersect these sets of rids (we’ll discuss intersection soon!) Retrieve the records and apply any remaining terms.

5) Present the methods for the implementation of the projection.
The expensive part is removing duplicates. – SQL systems don’t remove duplicates unless the keyword DISTINCT is specified in a query.
- Sorting Approach: Sort on and remove duplicates. (Can optimize this by dropping unwanted information while sorting.) (Modify Pass 0 of an external sort to eliminate unwanted fields. Modify merging passes to eliminate duplicates.)
- Hashing Approach: Hash on to create partitions. Load partitions into memory one at a time, build the in-memory hash structure, and eliminate duplicates. (Partitioning phase: Read R using one input buffer. For each tuple, discard unwanted fields, apply hash function h1 to choose one of the B-1 output buffers. The result is B-1 partitions (of tuples with no unwanted fields). 2 tuples from different partitions are guaranteed to be distinct. • Duplicate elimination phase: For each partition, read it

and build an in-memory hash table, using hash fn h2 (<> h1) on all fields, while discarding duplicates. If the partition does not fit in memory, can apply hash-based projection algorithm recursively to this partition.)

- The sort-based approach is the standard; better handling of skew and result is sorted.
    - If an index on the relation contains all wanted attributes in its search key can do the index-only scan. Apply projection techniques to data entries (much smaller!)
    - If an ordered (i.e., tree) index contains all wanted attributes as a prefix of the search key, can do even better: Retrieve data entries in order (index-only scan), discard unwanted fields, compare adjacent tuples to check for duplicates.

6) Describe the nested loops join, the index nested loops join, and the block nested loops join.

- Simple nested: For each tuple in the outer relation R, we scan the entire inner relation S. Cost: M + pR * M * N I/Os. • Page-oriented Nested Loops join: For each page of R, get each page of S, and write out matching pairs of tuples, where r is in R-page and S is in Spage. § Cost: M + M*N
- Index: If there is an index on the join column of one relation (say S), can make it the inner and exploit the index. § Cost: M + ((M*pR) * cost of finding matching S tuples) • For each R tuple, the cost of probing S index is about 1.2 for the hash index, 2-4 for B+ tree. § Clustered index: 1 I/O (typical), unclustered: up to 1 I/O per matching S tuple.
- Block: Use one page as an input buffer for scanning the inner S, one page as the output buffer, and use all remaining pages to hold ``block'' of outer R. § For each matching tuple r in R-block, s in S-page, add to result. Then read the next R-block, scan S, etc. - Cost: Scan of outer + #outer blocks * scan of inner where #outer blocks= #pages of outer block / blocksize. With sequential reads considered, analysis changes: may be best to divide buffers evenly between R and S.

Leon: Prm – pages related to M

1. **Simple Nested Loops Join**: M + M*prm*N (M – outer relation, N- inner relation)
   N + N*prn*M (N – outer relation, M- inner relation)
   - For each tuple in the outer relation R1, we scan the entire inner relation R2.

1.1 **Block Nested Loops Join**: M+M*N
- For each page of R1, get each page of R2, and write out matching pairs of tuples
- If we have enough space in the memory for N+2 pages, where N is smaller relation: M+N
- If there is not enough space (lets say B=102):
  M+N*(M/(B-2)) = 1000+500*1000/100=6000

1.2 **Index Nested Loops Join** (if we have hash index on one of the join attributes, we put relation with index on the inner part of the join)
Example for hash index: N+M*prn*1,2 M+N*prm*1,2

7) Present the sort-merge join algorithm.

- Sort R and S on the join column, then scan them to do a ``merge'' (on join col.), and output result tuples. Advance scan of R until current R-tuple >= current S tuple, then advance scan of S until current S-tuple >= current R tuple; do this until current R tuple = current S tuple. At this point, all R tuples with the same value in Ri (current R group) and all S tuples with the same value in Sj (current S group) match; output for all pairs of such tuples. Then resume scanning R and S. • R is scanned once; each S group is scanned once per matching R tuple. (Multiple scans of an S group are likely to find needed pages in buffer.)
- Wikipedia: The basic problem of a join algorithm is to find, for each distinct value of the join attribute, the set of tuples in each relation which display that value. The key idea of the sort-merge algorithm is to first sort the relations by the join attribute so that interleaved linear scans will encounter these sets at the same time.

2. **Sort-Merge Join**: M and N (example: B=100)

$$2*|M|*\,(1+\overset{\sim \log_B(|M|)}{\log_{B-1}(|M|/B)}) + 2*|N|*\,(1+\log_{B-1}(|N|/B)) + |M| + |N| =$$

$$2*1000*(1+1) + 2*500*(1+1) + 1000 + 500 = 7500$$

8) Describe the hash-based join algorithm.

- Partition both relations using hash fn h: R tuples in the partition I will only match S tuples in partition i. Read in a partition of R, hash it using h2 (<> h!). Scan matching partition of S, search for matches.
- #partitions k < B-1 (why?), and B-2 > size of largest partition to be held in memory. If the hash function does not partition uniformly, one or more R partitions may not fit in memory. Can apply hash-join technique recursively to do the join of this R-partition with corresponding S-partition. In the partitioning phase, read+write both relations; 2(M+N). In the matching phase, read both relations; M+N I/Os.

3. **Hash Join** (hash index on both join attributes)
   Both relations with hash index we split to B-1 partitions

   - 3*(M+N), if B > √N (N is smaller relation)
   - If not, we have the example of excesses and each partition of the smaller relationships, that doesn't fit into the memory is recursively divided.

Lecture 7: Query Optimization

1) How to estimate the cost of a query?

Cost estimation is determined by:

- Calculating the cost of each operation
- Number of blocks read from disk
- CPU time (for in-memory operations)
- Sizes of the intermediate results

2) Present the relational algebra equivalences?
Two relational algebra expressions over the same set of input tables are said to be equivalent if they produce the same result on all instances of the input tables. Relational algebra equivalences play a central role in identifying alternative plans. the FROM clause, after applying the selections in the WHERE clause. The use of equivalences enables us to convert this initial representation into equivalent expressions.
In particular:

- Selections and cross-products can be combined into joins.
- Joins can be extensively reordered.
- Selections and projections, which reduce the Size Of the input, can be "pushed" 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)

- A projection commutes with a selection that only uses attributes retained by the projection.
- Selection between attributes of the two arguments of a cross-product converts cross-product to a join. - A selection on just attributes of R commutes with
  $\delta(R \bowtie S) \equiv \delta(R) \bowtie S$
- Similarly, if a projection follows a join, we can `push’ it by retaining only attributes of R (and S) that are needed for the join or are kept by the projection.
  $\Pi(R \bowtie S) \equiv \Pi(R) \bowtie S$

3) How to obtain all equivalent query expressions for a given query expressed in the relational algebra?

- A relational query optimizer uses relational algebra equivalences to identify many equivalent expressions for a given query. For each such equivalent version of the query, all available implementation techniques are considered for the relational operators involved, thereby generating several alternative query evaluation plans. The optimizer estimates the cost of each such plan and chooses the one with the lowest estimated cost. (Book)
- Query optimizers use the equivalence rules to generate expressions equivalent to the given expression for our query. They can generate all equivalent expressions such that the application of all applicable equivalence rules on every sub-expression of every equivalent expression that was found and this is repeated until no new equivalent expressions are generated. This is very costly so optimized plans are created that consider all rules. (Internet)

4) Describe the procedure for the evaluation of the alternative query plans.

- The query optimizers enumerate the set of plans and choose the plan with the least estimated cost. The relational algebra equivalences play a central role in identifying alternative query paths. There are two main causes of such plans:
    - Single-relation plans: each access path is considered (with and without indexes) and the one with the least estimated cost is chosen;
    - Multiple-relation plans: with the number of joins increasing, the number of alternative paths grows rapidly so to reduce search space, only left-deep join trees are considered;

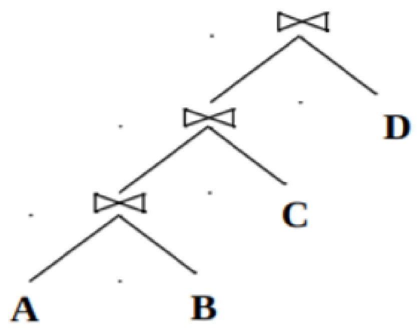
5) Describe cost estimation for single-relation query plans.

For queries over a single relation, queries consist of a combination of selects, projects, and aggregate ops:

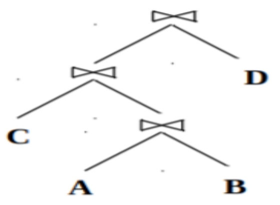
- Each available access path (file scan/index) is considered, and the one with the least estimated cost is chosen.
- The different operations are essentially carried out together (e.g., if an index is used for a selection, projection is done for each retrieved tuple, and the resulting tuples are pipelined into the aggregate computation).

6) Describe left-deep and bushy join trees?

Left-deep trees allow the implementation of the pipeline (fully pipelined plans – pipelined execution), with no need to store intermediate results.



Left-deep: we join A and B first, then join the result with C, then join the result with D.
Bushy Join trees allow parallel execution:



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 the best 1-relation plan for each relation.
- Pass 2: Find the best way to join the result of each 1-relation plan (as outer) to another relation. (All 2-relation plans.)
- Pass N: Find the best way to join the result of an (N-1)-relation plan (as outer) to the Nth 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.

Lecture 8: 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. Users submit transactions and can think of each transaction as executing by itself.

2) Explain possible anomalies of the interleaved execution of transactions.

```
T1:   R(A), W(A),                      R(B), W(B), Abort
T2:                  R(A), W(A), C
```

Reading Uncommitted Data
(Dirty Read, WR Conflict)

Transaction T2 read an object that has been modified by another transaction T1 (T1 has not been committed yet)

```
T1:   R(A),                          R(A), W(A), C
T2:           R(A), W(A), C
```

Unrepeatable Read(RW Conflict)

Transaction T2 changed the value of an object that has been read by another transaction T1 (T1 has not been committed yet)

```
T1:   W(A),                    W(B), C
T2:           W(A), W(B), C
```

Overwriting Uncommitted Data
(WW Conflict)

Transaction T2 overwrote the value of an object which has already been modified by another transaction T1 (T1 has not been committed yet)

3) What is a conflict serializable schedule of a transaction?
Schedule S is conflict serializable if S is conflict equivalent to some serial schedule. A schedule that does not conflict serializable has cycled in the dependency graph. Dependency graph: One node per Xact; edge from Ti to Tj if Ti proceeds and conflicts with some action from Tj. - Theorem: Schedule is conflict serializable if and only if its dependency graph is acyclic.

4) Describe the strict and non-strict two-phase locking protocols.
Strict Two-phase Locking (Strict 2PL) Protocol:
- Each Xact must obtain an S (shared) lock on the object before reading, and an X (exclusive) lock on the object before writing.
- All locks held by a transaction are released when the transaction completes
    - (Non-strict) 2PL Variant: Release locks anytime, but cannot acquire locks after releasing any lock.
- If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.

Strict 2PL allows only serializable schedules.
- Additionally, it simplifies transaction aborts
- (Non-strict) 2PL also allows only serializable schedules but involves more complex abort processing

5) What happens when a transaction is aborted?
- If a transaction Ti is aborted, all its actions have to be undone. Not only that, if Tj reads an object last written by Ti, Tj must be aborted as well! Most systems avoid such cascading aborts by releasing a transaction's locks only at commit time. If Ti writes an object, Tj can read this only after Ti commits.
- In order to undo the actions of an aborted transaction, the DBMS maintains a log in which every write is recorded. This mechanism is also used to recover from system crashes: all active Xacts at the time of the crash are aborted when the system comes back up.

6) Describe the deadlock prevention methods.
Deadlock: Cycle of transactions waiting for locks to be released by each other. Two ways of dealing with deadlocks: Deadlock prevention and Deadlock detection.
Prevention: Assign priorities based on timestamps. Assume Ti wants a lock that Tj holds. Two policies are possible:
- Wait-Die: It 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?
If there is a dense index on the rating field using Alternative (2), T1 should lock the index page containing the data entries with rating = 1.
If there are no records with rating = 1, T1 must lock the index page where such a data entry would be, if it existed! If there is no suitable index, T1 must lock all pages, and lock the file/table to prevent new pages from being added, to ensure that no new records with rating = 1 are added.

8) Describe the optimistic concurrency control.
Disadvantages of locking:
- Lock management overhead.
- Deadlock detection/resolution.
- Lock contention for heavily used objects.
If conflicts are rare, we might be able to gain concurrency by not locking, and instead of checking for conflicts before Xacts commit.
Kung-Robinson Model: Xacts have three phases:
- READ: Xacts read from the database, but make changes to private copies of objects.

- VALIDATE: Check for conflicts. (Test conditions that are sufficient to ensure that no conflict occurred. Each Xact is assigned a numeric id. Just use a timestamp)
- WRITE: Make local copies of changes public.

Multiple transactions can complete without interfering with one another.

Lecture 9: Crash Recovery

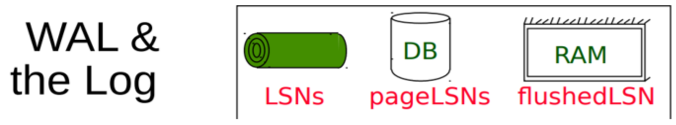
1) Explain the ACID properties of a relational DBMS.
- Atomicity: All actions in the Xact happen, or none happen. (The transaction is either executed or not.)
- Consistency: If each Xact is consistent, and the DB starts consistent, it ends up consistent. (The database has to be consistent before and after the transaction)
- Isolation: Execution of one Xact is isolated from that of other Xacts. (Concurrently executed transactions do not affect each other.)
- Durability: If a Xact commits, its effects persist. (Transactions are durable upon confirmation)

The Recovery Manager guarantees Atomicity & Durability.

2) Explain the trade-offs of "stealing" pages from the buffer pool and "forcing" pages to be stored on a disk.
Suppose a transaction T1 wants to read a data object X, but the working memory is full with all the other transactions' work. So T1 needs to clear some memory, which it does by kicking some other page in working memory to stable storage. This can be dangerous because we can't be sure that what T1 is pushing to stable storage has been committed yet. This is known as stealing.
Forcing means that every time a transaction commits, all the affected pages will be pushed to stable storage. This is inefficient because each page may be written by many transactions and will slow the system down.
So stealing is efficient but not safe and forcing is safe but not efficient.
Iztok:
Force: Poor response time. – But provides durability
Steal: We can’t assure atomicity.

3) Describe the Write-Ahead Logging (WAL) protocol.
The Write-Ahead Logging Protocol:
1. Must force the log record for an update before the corresponding data page gets to disk. **Guarantees Atomicity.**
2. Must write all log records for a Xact before we commit. **Guarantees Durability.**

4) Describe what data is stored and where it is stored for the execution of the WAL protocol.



In the log: Log records - Each log record has a unique Log Sequence Number (LSN).
Disk: Data pages each with a pageLSN and the master record.
RAM: Xact Table lastLSN status, Dirty Page Table recLSN, flushedLSN

5) Present the analysis, redo and undo phases of the crash recovery.
Analysis:

- Reconstruct state at the checkpoint. – via end_checkpoint record.
- Scan log forward from the checkpoint.
  - End record: Remove Xact from Xact table.
  - Other records: Add Xact to Xact table, set lastLSN=LSN, change Xact status on commit.
  - Update record: If P not in Dirty Page Table, • Add P to D.P.T., set its recLSN=LSN.

Lecture 10: Logical Database Design

1) Why can redundancy appear in relational databases?
Relational databases can store the same information redundantly (in more than one place within the database) which leads to redundant storage and insert/delete/update anomalies. Redundancy can occur when inconsistent duplicates of the same entry are found in the database.

2) What is a functional dependency?
An FD is a statement about all allowable relations.

- Must be identified based on the semantics of the application.
- Given some allowable instance r1 of R, we can check if it violates some FD f, but we cannot tell if f holds over R!

Leon: Functional dependencies represent the relationship between the attributes of the relational schema. They limit the permissible values of the attributes in the relationship tuples. Let X and Y be nonempty sets of attributes in relation to schema R: X,Y ⊆ R. The attributes X functionally determine attributes Y (denoted by X→Y) if there do not exist two tuples in any relation to R schema that would match the values of the X attributes and would not match the values of the Y attributes.

3) How can we reason about the functional dependencies?
Closure: An FD f is implied by a set of FDs F if f holds whenever all FDs in F hold.

- F+ = closure of F is the set of all FDs that are implied by F.

Armstrong's Axioms: (X, Y, Z are sets of attributes):

- Reflexivity: If X ⊆ Y, then Y → X
- Augmentation: If X → Y, then XZ → YZ for any Z
- Transitivity: If X → Y and Y → Z, then X → Z

Union: If X → Y and X → Z, then X → YZ

Decomposition: If X → YZ, then X → Y and X → Z

Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in # attrs!). Typically, we just want to check if a given FD X -> Y is in the closure of a set of FDs F. An efficient check:

Compute attribute closure of X (denoted) wrt F:

- Set of all attributes A such that X A is in
- There is a linear-time algorithm to compute this.

4) What is the purpose of the normalization of a relation? What is the normal form?

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.

- Role of FDs in detecting redundancy: Consider a relation R with 3 attributes, ABC.
  - No FDs hold: There is no redundancy here. • Given A -> B: Several tuples could have the same A value, and if so, they'll all have the same B value!

Leon: Normalization is the process of transforming relational schemes or corresponding relations into a form in which we cannot get anomalies. Normalization uses a bottom-up approach. Planning a database thus starts with a single relationship that contains all the important attributes. The relationship is then gradually transformed into the desired normal form. • 1NF: first normal form • 2NF: second normal form • 3NF: third normal form • BCNF: Boyce-Codd normal form • 4(B)NF: fourth (business) normal form • 5(B)NF: fifth (business) normal form

5) Present the Boyce-Codd normal form of a relation.

Relation R with FDs F is in BCNF if, for all X A in

- A X (called a trivial FD), or
- X contains a key for R.

In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints. In the case of arbitrary R with a dependency X→A, then A must be the same for all tuples having the same X. If we do not have dep X→A then no value A can be inferred. In case R is in BCNF then X must contain a key and the value of A can be inferred.

Internet: For a table to satisfy the Boyce-Codd Normal Form, it should satisfy the following two conditions:

1. It should be in the Third Normal Form.

2. And, for any dependency A B, A should be a super key.

The second point sounds a bit tricky, right? In simple words, it means that for a dependency A —y B, A cannot be a non-prime attribute, if B is a prime attribute.

6) Present the 3rd normal form (3NF) of a relation.

Relation R with FDs F is in 3NF if, for all X A in

- A ∈ X (called a trivial FD), or
- X contains a key for R, or
- A is part of some key for R.

• Minimality of a key is crucial in the third condition above!
• If R is in BCNF, obviously in 3NF.
• If R is in 3NF, some redundancy is possible.

If 3NF violated by X->A, one of the following holds:

- X is a subset of some key K
- X is not a proper subset of any key.
- There is a chain of FDs K X A, which means that we cannot associate an X value with a K value unless we also associate an A value with an X value.

But: even if the relation is in 3NF, these problems could arise.

Leon: The relation is in third normal form 3NF, if:

- It is in second normal form 2NF and
- It doesn‘t have transitive dependencies.

Transitive dependence is a functional dependency between attributes that are not part of the key. We eliminate transitive dependencies by breaking the relationship into several new relations. The relation is automatically in 3NF if it is in 2NF and its key consists of all or all but one schema attribute.

7) What is a lossless-join decomposition of a relation?

Let R be a relation schema and let F be a set of FDs over R. A decomposition of R into two schemas with attribute sets, X and Yis said to be a lossless-join decomposition with respect to F if, for every instance r of R that satisfies the dependencies in F, $\Pi x\ (r) \bowtie \Pi y\ (r) = r$. In other words, we can recover the original relation from the decomposed relations.

It is essential that all decompositions used to deal with redundancy be lossless!

The decomposition of R into X and Y is lossless-join wrt F if and only if the closure of F contains:

- $X \cap Y \rightarrow X$, or
- $X \cap Y \rightarrow Y$

8) What is dependency preserving decomposition of a relation?

Dependency preserving decomposition (Intuitive):

If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, on Y and on Z, then all FDs that were given to hold on R must also hold. Projection of set of FDs F: If R is decomposed into X, ... a projection of F onto X (denoted FX) is the set of FDs U → V in F+ (closure of F) such that U, V are in X. Decomposition of R into X and Y is dependency preserving if (FX union FY) + = F + i.e., if we consider only dependencies in the closure F + that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in F +.
Internet: The dependency preservation decomposition is another property of decomposed relational database schema D in which each functional dependency X -> Y specified in F either appeared directly in one of the relation schemas Ri in the decomposed D or could be inferred from the dependencies that appear in some Ri. Decomposition D = { R1 , R2, R3,,.., ,Rm} of R is said to be dependency-preserving with respect to F if the union of the projections of F on each Ri , in D is equivalent to F. In other words, R ⊂ join of R1, R1 over X. The dependencies are preserved because each dependency in F represents a constraint on the database. If decomposition is not dependency-preserving, some dependency is lost in the decomposition.

10) Present the algorithm for the decomposition of a relation into the BCNF.
Obviously, the algorithm for lossless join decomposition into BCNF can be used to obtain a lossless join decomposition into 3NF (typically, can stop earlier).
- To ensure dependency preservation, one idea:
  If X Y is not preserved, add-relation XY.
  Problem is that XY may violate 3NF!

Refinement: Instead of the given set of FDs F, use a minimal cover for F.

11) Present the algorithm for the decomposition of a relation into the 3NF.
Consider relation R with FDs F. If X Y violates BCNF, decompose R into R - Y and XY. Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition and guaranteed to terminate. In general, several dependencies may cause violation of BCNF. The order in which we ``deal with'' them could lead to very different sets of relations! In general, there may not be a dependency preserving decomposition into BCNF.

Lecture 11: Physical Database Design

1) Describe the workload of the application. What information is included?
We must begin by understanding the workload:
- The most important queries and how often they arise.
- The most important updates and how often they arise.
- The desired performance for these queries and updates.

Workload includes all queries generated by an application.

- General: Study UPDATE/INSERT/DELETE/UPDATE queries and related attributes. Frequency of queries. Upper bounds for queries.
- For each read query Which relations|attributes are accessed? How often? In one query? Which attributes are involved in joins? The selectivity of conditions?
- For each update Which attributes are in selection condition? The same for joins... How often? ...

2) Describe index selection for the efficient implementation of a select operation.
Our choice of indexes is guided by the plan(s) that we expect an optimizer to consider for a query. Attributes mentioned in a WHERE clause are candidates for indexing, ideally a hash index. A range selection condition suggests that we consider a B+ - tree index on the selected attributes. As a rule of thumb, equality selections are more selective than range selections. A B+ - tree index is usually preferable because it supports range queries as well as equality queries.

3) Describe index selection for the efficient implementation of a join operation.
Considering a join condition, a hash index on the inner is very good for Index Nested Loops. It should also be clustered if the join column is not key for inner, and inner tuples need to be retrieved. A Clustered B+ tree on join column(s) would be good for Sort-Merge.

4) Explain why and how the alternative decomposition of a relation should be chosen.
There are two ways to decompose a relation – lossless and dependency preserving decomposition. If the information is not lost from the relation that is decomposed, then the decomposition will be lossless. This guarantees that the join of relations will result in the same relation as it was decomposed.
In the dependency preservation, at least one decomposed table must satisfy every dependency. So, if a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.

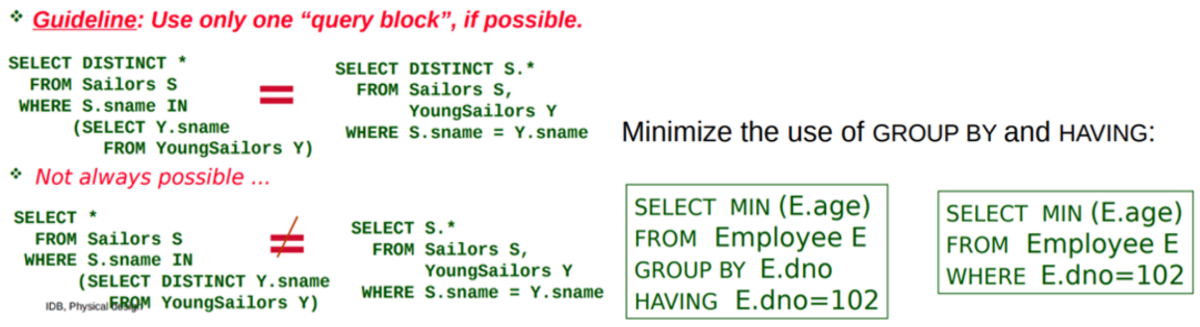
5) Explain when it makes sense to denormalize a relation.
When we want to ``undo'' some decomposition steps and settle for a lower normal form. To speed up the query. If the query is sufficiently important, and we cannot obtain adequate performance otherwise (i.e., by adding indexes or by choosing an alternative 3NF schema.)

6) What is a horizontal decomposition? Why is it used?
Sometimes, we might want to replace a relation with a collection of relations that are selections.

- Each new relation has the same schema as the original, but a subset of the rows.
- Collectively, new relations contain all rows of the original. Typically, the new relations are disjoint.

One way to deal with this is to build a clustered B+ tree index on the field.
A second approach is to replace contracts with two new relations with the same attributes.

7) Present some examples of queries that need to be tuned or rewritten to gain efficiency.



❖ ***Guideline**: Use only one "query block", if possible.*

```
SELECT DISTINCT *
  FROM Sailors S
 WHERE S.sname IN
      (SELECT Y.sname
         FROM YoungSailors Y)
```

=

```
SELECT DISTINCT S.*
  FROM Sailors S,
       YoungSailors Y
 WHERE S.sname = Y.sname
```

❖ *Not always possible ...*

```
SELECT *
  FROM Sailors S
 WHERE S.sname IN
      (SELECT DISTINCT Y.sname
         FROM YoungSailors Y)
```

≠

```
SELECT S.*
  FROM Sailors S,
       YoungSailors Y
 WHERE S.sname = Y.sname
```

IDB, Physical Design

Minimize the use of GROUP BY and HAVING:

```
SELECT  MIN (E.age)
FROM  Employee E
GROUP BY  E.dno
HAVING  E.dno=102
```

```
SELECT  MIN (E.age)
FROM  Employee E
WHERE  E.dno=102
```

Lecture 12: ER Data Model

1) Describe the data model ER.
Internet: ER Model stands for Entity-Relationship Model, is a high-level conceptual data model diagram. ER model helps to systematically analyze data requirements to produce a well-designed database.
The ER Model represents real-world entities and the relationships between them. Creating an ER Model in DBMS is considered the best practice before implementing your database. Main components: entities, attributes, relationships.

2) Describe when to use an entity, a relationship, and an attribute in the conceptual design of a relational DBMS.
We use entities when we want to represent objects from the real world that can be separated from other objects. When we want to represent the properties of that object we use attributes and we say that they are the identifier of that entity. There exists a primary identifier and the rest of the attributes are description attributes. Complex and multi-valued attributes allow for an abstract representation of a property. A relationship defines the link between two or more entities that can belong to different entity sets.

3) Present the concept of the cardinality of a relationship.
Let E1, E2... En be entity sets connected with a relationship R. Cardinality is defined for each particular entity set taking part in the relationship R.
The cardinality of an entity set Ei in relationship R describes how many different relationships an entity from R can participate.

Cardinality of the entity set Ei in the relationship R is a function: card(Ei ,R) = (min, max)

- min is minimal cardinality of Ei in relationship R
- max is maximal cardinality Ei in R

Possible values of minimal and maximal cardinality:

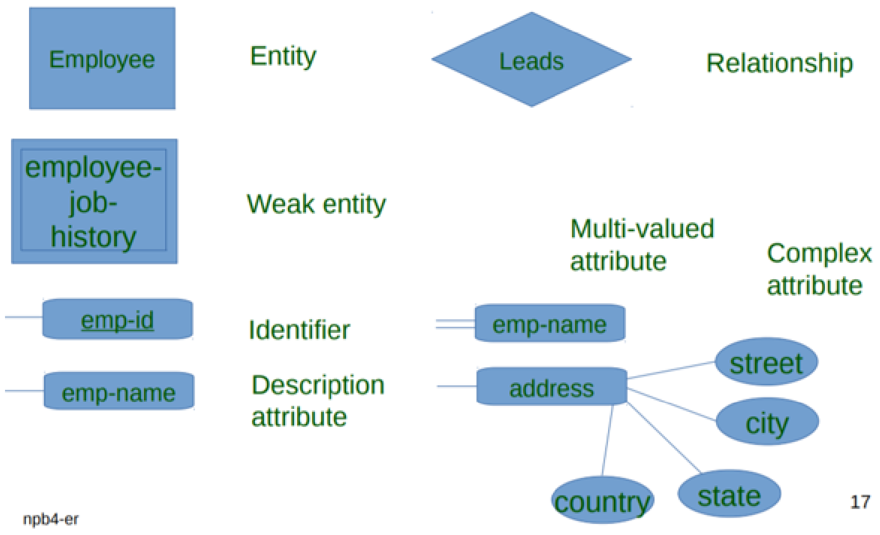
- “0” (zero), “1” (one) “N” (“N” reads “many”; in general, “more than one”).
- min-card(E,R) and max-card(E,R)

4) What is a weak entity?

- A weak entity can be identified uniquely only by considering the primary key of another (owner) entity.
- Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
- Weak entity set must have total participation in this identifying relationship set.

Strong Entity: A strong entity is not dependent on any other entity in the schema. A strong entity will always have a primary key. Strong entities are represented by a single rectangle. The relationship of two strong entities is represented by a single diamond. Various strong entities, when combined together, create a strong entity set.
Weak Entity: A weak entity is dependent on a strong entity to ensure its existence. Unlike a strong entity, a weak entity does not have any primary key. It instead has a partial discriminator key. A weak entity is represented by a double rectangle. The relation between one strong and one weak entity is represented by a double diamond.
Leon: • The existence of a weak entity is conditioned with the existence of some other entity. • presented with an arrow in the direction of the dependence • it is identified by using a key of another entity • example: every bread has its own baker (entity bread is dependent on entity baker)

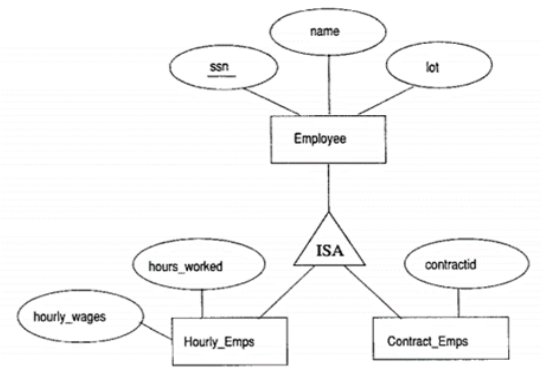
5) Present the Chen notation of the ER model.



The relationship can connect one or multiple tuples from one entity to one or multiple tuples of another entity. You can use these cardinalities for the relationships:

- One-to-one (1:1)
- One-to-many (1:N)
- Many-to-one (N:1)
- Many-to-many (M:N)

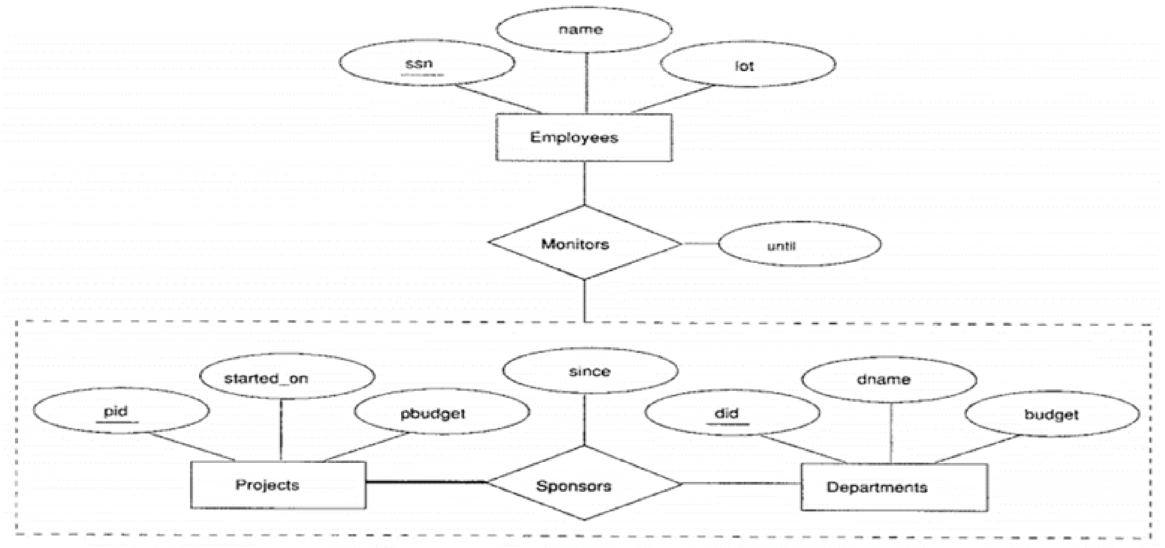
6) Present the constructs for modeling the generalization/specialization hierarchy of entities.



A class hierarchy can be viewed in one of two ways:

- Employees are specialized in subclasses. Specialization is the process of identifying subsets of an entity set (the superclass) that share some distinguishing characteristic. Typically, the superclass is defined first, the subclasses are defined next, and subclass-specific attributes and relationship sets are then added.
- Hourly-Emps and Contract-Emps are generalized by Employees. As another example, two entity sets Motorboats and Cars may be generalized into an entity set Motor _ Vehicles. Generalization consists of identifying some common characteristics Of a collection Of entity sets and creating a new entity set that contains entities possessing these common characteristics. Typically, the subclasses are defined first, the superclass is defined next, and any relationship sets that involve the superclass are then defined.
-

7) How to model the aggregation of entities?



To define a relationship set such as Monitors, we introduce a new feature of the ER model, called aggregation. Aggregation allows us to indicate that a relationship set (identified through a dashed box) participates in another relationship set. This is illustrated in Figure 2.13, with a dashed box around Sponsors (and its participating entity sets) used to denote aggregation. This effectively allows us to treat Sponsors as an entity set for purposes of defining the Monitors relationship set.

- Used when we have to model a relationship involving (entity sets and) a relationship set.
- Aggregation allows us to treat a relationship set as an entity set for purposes of participation in (other) relationships.

8) Present the rules for translating the ER model into the relational database model.
Each entity Set maps to a new table, each attribute to a new table column, and each relationship set to either new table columns or to a new table.
Translation of entities - SQL table with the same information contents as the original entity from which it was developed.
The translation is used for entities with the binary relationship:

  - N-N
  - 1-N on „one“ side
  - 1-1 on both sides

- ● Entities that are linked with the binary recursive relationship of type N-N
- ● Entities with ternary or, in general, n-ary relationship, or, with the generalization hierarchy

Translation of relationships - SQL table derived from a relationship contains foreign keys of all entities in the relationship.
This translation is used for:

- Binary relationships of type N-N – always defined with a table that contains foreign keys of all entities in the relationship.

Relationships that are recursive and N-N

- Relationships that are ternary or higher-order

Recursive relationship - Binary relationship defined on a single entity
Two possible translations into SQL:

- Set of pairs is stored in a separate SQL table
- Set of pairs can be represented by an additional column of SQL table derived from the given entity

Ternary relationships: The constraints are important for the translation of the ternary relationship:

- Some can be expressed: participation constraints
- Some can not be expressed with SQL CREATE statement
- Keys in a ternary relationship are always NOT NULL
- Keys must be updated and deleted

Generalization - Translation of a generalization hierarchy composed of a root supertype and subtypes can result in more than one SQL tables

- A table derived from the root supertype contains
  - The key of the supertype entity, and
  - All additional attributes defined for the root supertype entity
- A table derived from the subtype entity contains
  - A key of the supertype
  - Only the attributes from the specific subtype entity