Review

June 2, 2023

Relational model

- A database is a collection of relations and each relation is an unordered set of tuples (or rows).
- Each relation has a set of attributes (or columns).
- Each attribute has a name and a domain and each tuple has a value for each attribute of the relation.

Keys: superkey, candidate key, primary key, foreign key

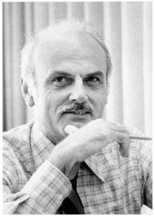


Figure: Edgar Frank Codd

https://en.wikipedia.org/wiki/Edgar_F._Codd

Relational algebra

- Selection $\sigma_p(R)$
- Projection $\Pi_{A_1,\ldots,A_k}(R)$
- Product $R \times S$
- Union $R \cup S$
- Difference $R - S$
- Renaming $\rho_{S(A_1,\ldots,A_k)}(R)$, $\rho_S(R)$
- Join $R \bowtie_\theta S$, $R \bowtie S$

SQL

- SQL DDL
- SFW statement
- Set operations of SQL
- Aggregation and grouping
- Three-valued logic of SQL
- Various joins
- Nested Subqueries
- Integrity constraints
- Update with SQL

Database design theory

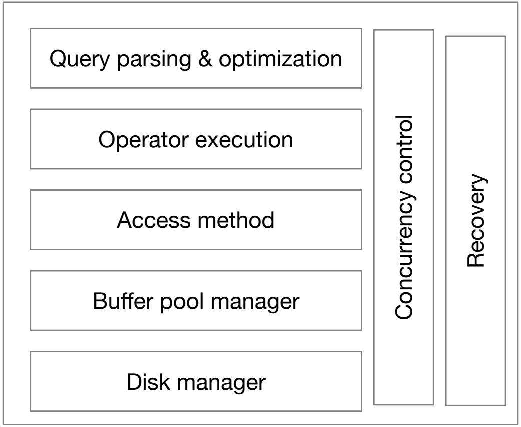
Functional dependency

- FD's are unique-value constraints.
- A FD $X \to Y$ requires the attributes of X functionally determining the attributes Y.
- X is a candidate key of R if (i) $X \to R$, and (ii) $Y \not\to X$ for all proper subset Y of X.
- The attribute closure X_F^+ is the set of attributes determined by X under F.

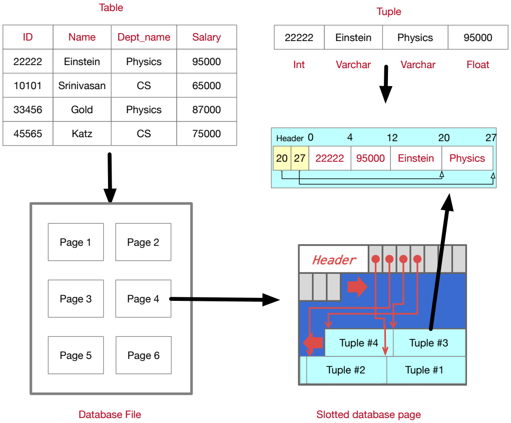
Normal forms and decomposition algorithms

- Insertion, deletion, update anomalies
- Decomposition criteria: lossless join, dependencies preserving, anomalies avoidance
- BCNF & BCNF decomposition algorithm
- 3NF & 3NF synthesis algorithm

Database internals



Storage structure



Table

ID	Name	Dept_name	Salary
22222	Einstein	Physics	95000
10101	Srinivasan	CS	65000
33456	Gold	Physics	87000
45565	Katz	CS	75000

Tuple

22222	Einstein	Physics	95000
Int	Varchar	Varchar	Float

- Tables are stored as database files.
- Each database file consists of a collection of pages.
- Each page is a block of fixed-size that contains a collection of tuples.

Buffer pool manager

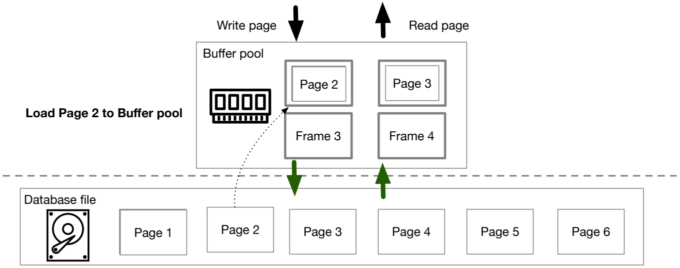


Figure: Buffer pool

Design goal: provide the illusion of operation in memory.

- A buffer pool is a memory region organized as an array of frames.
- When DBMS request a page, an exact copy is placed into one of these frames.
- Buffer pool page replacement policy: LRU & CLOCK.
- The access patterns have big impact on I/O cost.

Index

- Search key: an attribute or a set of attributes used to look up records in a file.
- An index file consists of records (called index entries) of the form

search key	pointer

- An index files is usually much smaller than the original file.

- Ordered index vs. hash index.
- Dense index vs. spare index.
- Clustering index vs. non-clustering index.
- Primary index vs. secondary index.

B^+-tree

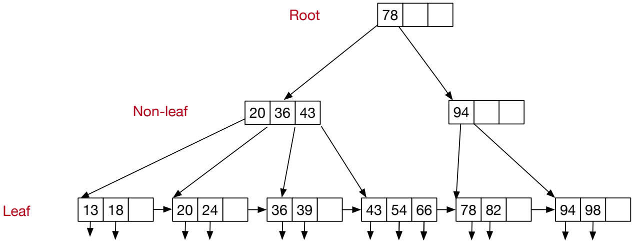


Figure: A sample B^+-tree with max_fanout$= 4$

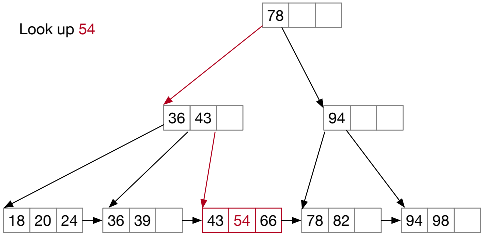
A B^+-tree in a self-balancing search tree with following properties.

- Balance invariant: all leaves are at the same level.
- Occupancy invariant: all nodes (except root) are at least half-full.
- Search, insertions, and deletions in logarithmic time.
- Optimized for disk-based DBMS: one node per block; large fan-out.

Query with B$^+$-tree index

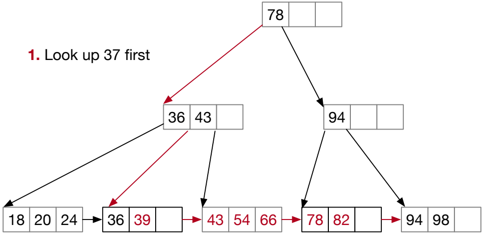
Point query:

```
SELECT * FROM R WHERE K=54;
```


Look up 54

Range query:

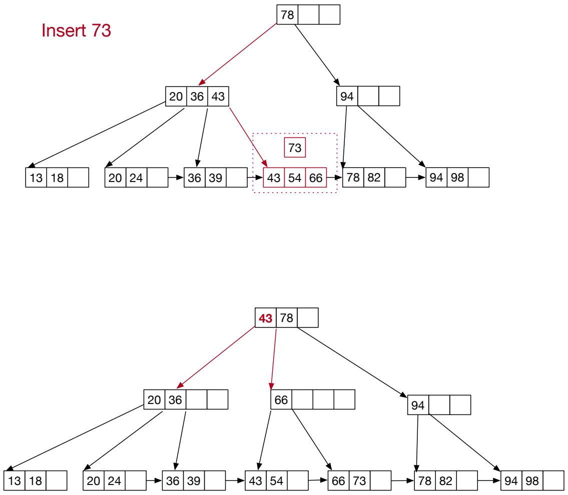
```
SELECT * FROM R
WHERE k >= 37 AND K <= 90;
```


1. Look up 37 first

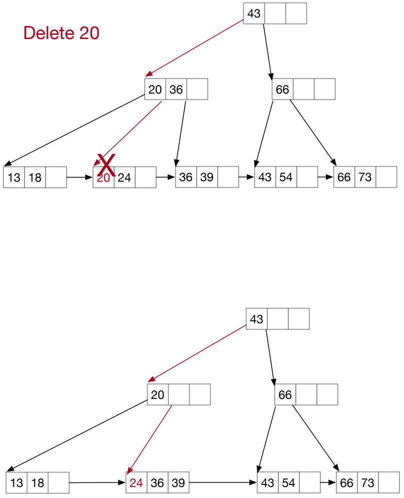
2. Follow the next leaf pointers until hit the upper bound

B$^+$-tree insertion & deletion

Insert 73

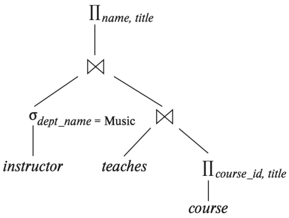


Delete 20

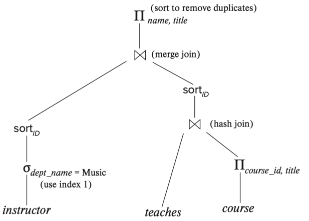


Query processing & optimization

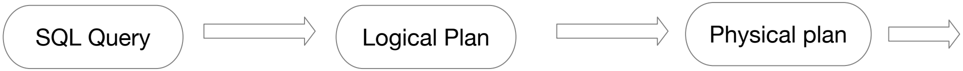
```
SELECT name, title
FROM instructor natural join teaches
     natural join course
WHERE dept_name ='Music';
```



$\Pi_{name,\ title}$
$\bowtie$
$\sigma_{dept\_name\ =\ \text{Music}}$
$\bowtie$
instructor
teaches
$\Pi_{course\_id,\ title}$
course



$\Pi_{name,\ title}$ (sort to remove duplicates)
$\bowtie$ (merge join)
sort_{ID}
sort_{ID}
$\bowtie$ (hash join)
$\sigma_{dept\_name\ =\ \text{Music}}$ (use index 1)
$\Pi_{course\_id,\ title}$
instructor
teaches
course

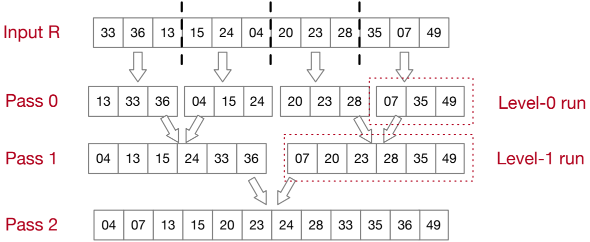


SQL Query → Logical Plan → Physical plan →

- Each node of a logical plan is a relational operator.
- Each node of a physical plan represents an operator algorithm.
- Data flows from the leaves of the physical plan tree up towards the root.

External merge sort

A divide-and-conquer approach to sort a large relation that cannot fit in memory.



Input R: 33 36 13 | 15 24 04 | 20 23 28 | 35 07 49

Pass 0: 13 33 36 | 04 15 24 | 20 23 28 | 07 35 49

Level-0 run

Pass 1: 04 13 15 24 33 36 | 07 20 23 28 35 49

Level-1 run

Pass 2: 04 07 13 15 20 23 24 28 33 35 36 49

- Pass 0: read B pages of R each time, sort them, and write out a level-0 run.
- Pass i: merge $(B-1)$ level-$(i-1)$ runs each time, and write out a level-i run.
- Each pass read the entire relation and write it once.
- Total cost: $2P(R) * \log_{B-1} \lceil P(R)/B \rceil + P(R)$

Join algorithms

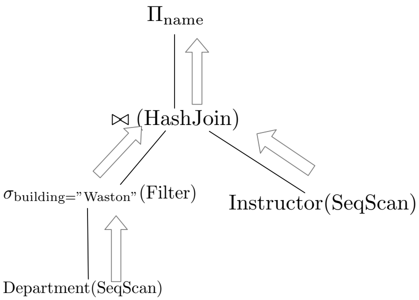
Algorithms	I/O costs
Naive Nested Loop Join	$P(R) + \|R\| * P(S)$
Block Nested Loop Join	$P(R) + P(R) * P(S)$
Indexed Nested Loop Join	$P(R) + \|R\| * C$
Merge Join	$P(R) + P(S)$
In-memory Hash Join	$P(R) + P(S)$
Hash Join	$3 * (P(R) + P(S))$

Table: Algorithms for $R \bowtie S$

- Tables: R, S
- Number of tuples: $|R|$, $|S|$
- Number of pages: $P(R)$, $P(S)$
- Cost metric: number of I/O's

Query processing model

A DBMS's processing model defines how the system executes a physical query plan.

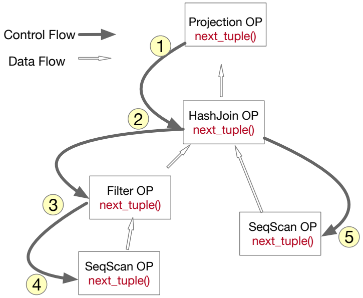


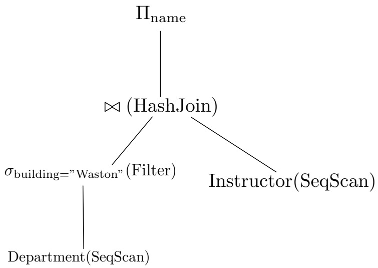
Materialization model

- Evaluate the physical query plan tree bottom-up.
- Children write intermediate results to temporary files.
- Parents read temporary files.
- Good for OLTP queries, not good for OLAP queries with large intermediate results.

Query processing model

A DBMS's processing model defines how the system executes a physical query plan.





Iterator Model (a.k.a. volcano model)

- Do not materialize intermediate results; children pipeline their results to parents.
- Every operator maintains its own execution state and implements a next_tuple method.
- Pull-based execution: (i) Call next_tuple() repeatedly on the root; (ii) Iterators recursively call next_tuple() on the inputs.

Rule-based query optimization

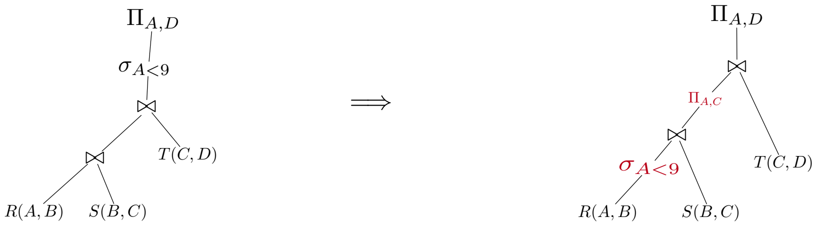
Rewrite query via RA equivalence rules.

- (i) $R \bowtie S = S \bowtie R$. (ii) $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$.
- $\sigma_\theta(R \times S) = R \bowtie_\theta S$. This rule converts a cross product to a theta join.
- $\Pi_{L_1}(\Pi_{L_2}(R)) = \Pi_{L_1}(R)$, where $L_1 \subseteq L_2$.
- $\sigma_{\theta_1}(\sigma_{\theta_2}(R)) = \sigma_{\theta_1 \wedge \theta_2}(R)$.
- Push down selection: $\sigma_{\theta_1 \wedge \theta_2}(R \bowtie_\theta S) = \sigma_{\theta_1}(R) \bowtie_\theta \sigma_{\theta_2}(S)$.

  Here θ_1 (resp. θ_2) involves only attributes of R (resp. S).
- Push down projection
  1. $\Pi_L(\sigma_\theta(R)) = \Pi_L(\sigma_\theta(\Pi_{L \cup L'}(R)))$
     – L' is the set of attributes that referenced by θ and not in L.
  2. $\Pi_L(R \bowtie_\theta S) = \Pi_L(\Pi_{L'}(R) \bowtie_\theta S)$.
     – L' consists of the set of attributes from R that either in L or referenced by θ.
  3. A symmetric version of (2).

Rule-based query optimization

Rewrite query via RA equivalence rules.



- Push down selection: $\sigma_{\theta_1 \wedge \theta_2}(R \bowtie_\theta S) = \sigma_{\theta_1}(R) \bowtie_\theta \sigma_{\theta_2}(S)$.

  Here θ_1 (resp. θ_2) involves only attributes of R (resp. S).

- Push down projection
  1. $\Pi_L(\sigma_\theta(R)) = \Pi_L(\sigma_\theta(\Pi_{L \cup L'}(R)))$
     – L' is the set of attributes that referenced by θ and not in L.
  2. $\Pi_L(R \bowtie_\theta S) = \Pi_L(\Pi_{L'}(R) \bowtie_\theta S)$.
     – L' consists of the set of attributes from R that either in L or referenced by θ.
  3. A symmetric version of (2).

Cost-based query optimization

- Enumerate “all” possible physical plans and pick the one with “lowest” cost.
- In practice, the goal is often not getting the optimal one, but instead avoiding really bad one.
- We have discussed the first cost-based query optimizer.
  - Use Selinger statistics for cost estimation.
  - Consider left-deep joins only for plan enumeration.
  - Generate optimal plans in a bottom-up fashion.

Figure: Patricia Selinger

P. Selinger et al. (1979). Access Path Selection in a Relational Database Management System.

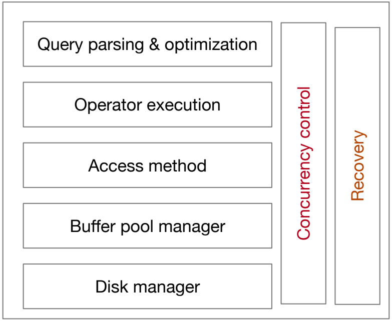
Transaction processing

A transaction ("TXN") is a collection of database operations that servers as a single, indivisible logical unit of work.

- Atomicity: Each TXN is all-or-nothing, i.e., no partial TXN is allowed.
- Consistency: Each TXN should leave the database in a consistent state.
- Isolation: Each TXN is executed as if it were executed in isolation.
- Durability: Effects of a committed TXN are resilient against failures.

Transaction processing

A transaction ("TXN") is a collection of database operations that servers as a single, indivisible logical unit of work.



- Concurrency control: ensure isolation in concurrent database access.
- Recovery: ensure atomicity and durability via logging.

Concurrency control

Goal: to ensure isolation of transactions.

Serializability: a desired property ensuring isolation.

Two-Phase Locking (2PL)

- A pessimistic approach: need to acquire a lock before every shared data access.
- The serializability order of conflicting operations is determined at runtime (according to the lock point).

Timestamp Ordering (T/O)

- An optimistic approach: (i) no locking, (ii) each TXN is assigned a unique timestamp before execution.
- Use the timestamps to determine the serializability order of TXNs.

Recovery

Goal: to ensure atomicity and durability via logging.

Write-ahead logging

- Enable "No-Force + Steal" buffer pool policy for performance.
- Require both REDO and UNDO logging.
- Reduce recovery cost by checkpointing.

Recovery: REDO + UNDO

- REDO: repeat history for durability.
- UNDO: cancel incomplete TXNs for atomicity.
- Write compensation log during UNDO.

Thanks & Good Luck!