JACOBS UNIVERSITY

Normal Forms and Physical Database Design

Ramakrishnan & Gehrke, Chapter 17 & 18

Road Map

- Normal Forms
  - Functional Dependencies
  - Normal Forms
  - Decomposition
- Physical database design
  - Indexing
  - Tuning

The Evils of Redundancy

Dept_id	budget	Emp_id	Emp_name	salary
1	100	1	John Williams	60
1	100	2	Phil Coulter	50
2	200	3	Norah Jones	45
3	300	4	Anastacia	40

- Redundancy at the root of several relational schema problems
  - redundant storage, insert/delete/update anomalies
- Integrity constraints identify problems and suggest refinements
  - in particular: functional dependencies

Functional Dependencies

- Let R be relation, X and Y sets of attributes of R
- Functional dependency (FD) X → Y holds over relation R
  if, for every allowable instance r of R:
  - $t1 \in r$, $t2 \in r$:
    $\pi_X(t1) = \pi_X(t2) \Rightarrow \pi_Y(t1) = \pi_Y(t2)$
  - FDs in example?

Dept_id	budget	Emp_id	Emp_name	salary
1	100	1	John Williams	60
1	100	2	Phil Coulter	50
2	200	3	Norah Jones	45
3	300	4	Anastacia	40

- K is a candidate key for R means that K → R
  - K → R does not require K to be minimal!
- FD is a statement about all allowable relation instances
  - Must be identified based on semantics of 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!

Example: Constraints on Entity Set

- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (ssn, name, lot, rating, hrly_wages, hrs_worked)
- Notation: relation schema by listing the attributes: SNLRWH
  - set of attributes {S,N,L,R,W,H}
  - Using equivalently to relation name (e.g., Hourly_Emps for SNLRWH)
- Some FDs on Hourly_Emps:
  - ssn is key: $S \rightarrow SNLRWH$
  - rating determines hrly_wages: $R \rightarrow W$

Example (Contd.)

- Problems due to R → W :
  - Update anomaly:
    change W in just the 1st tuple of SNLRWH?
  - Insertion anomaly:
    insert employee and don't know the hourly wage for his rating?
  - Deletion anomaly:
    delete all employees with rating 5
    ⇒ lose information about the wage for rating 5!

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Wages

R	W
8	10
5	7

Hourly_Emps2

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Will 2 smaller tables be better?

Normal Forms & Functional Dependencies

- normal forms avoid / minimize certain kinds of problems
  - helps to decide on decomposing relation
- Role of FDs in detecting redundancy
  - No FDs hold: no redundancy
  - Given relation R with 3 attributes ABC and FD A → B:
    Several tuples might have the same A value; if so, they all have the same B value

It's all about hidden repeating information across tuples

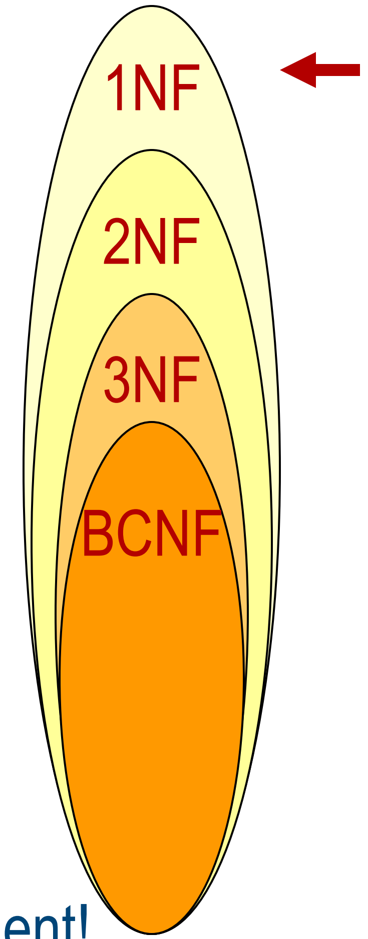
First Normal Form

- First Normal Form (1NF)
  - eliminates attributes containing sets = repeating groups
  - ...by flattening: introduce separate tuples with atomic values
- Ex:

id	name	skillsList
1	Jane	{C,C++,SQL}
2	John	{Java,python,SQL}

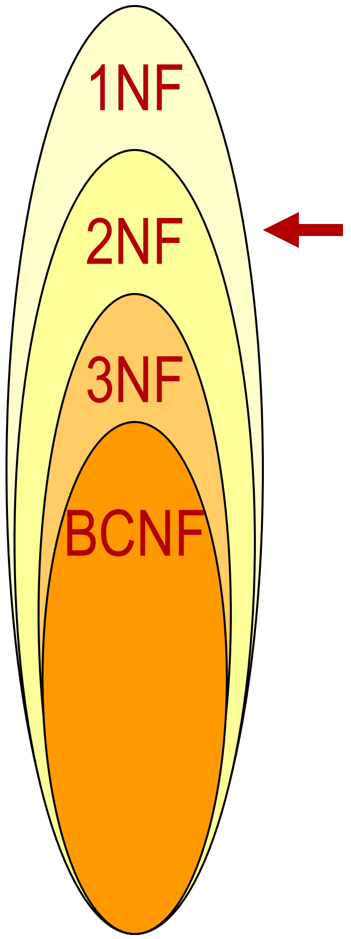
id	name	skill
1	Jane	C
1	Jane	C++
1	Jane	SQL
2	John	Java
2	John	Python
2	John	SQL

  - Skills not f.d. on id, nor name!
- Oops: lost primary key property.
  - Will fix that later.
- Why good? Repeating groups complicate storage management!
  - Experimental DBMSs exist for non-1NF (NFNF, NF^2) tables



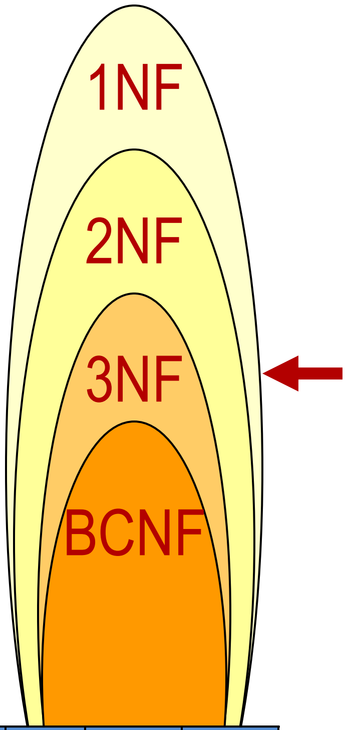
Second Normal Form

- Second Normal Form (2NF):
  - eliminates functional dependencies on a partial key
  - by putting the fields in a separate table
    from those that are dependent on the whole key
- Ex: ABCD with B→C
  becomes: ABD, BC



Third Normal Form (3NF)

- Relation R with FD set F is in 3NF if, for all $X \rightarrow A$ in F^+,
  - Either $A \in X$ (called a *trivial* FD)
  - Or X contains a key for R
  - Or A is part of some key for R
- In plain words:
  - 3NF eliminates functional dependencies on non-key fields by putting them in a separate table
  - = in 3NF, all non-key fields are dependent on
    the key,
    the whole key,
    and nothing but the key
- Ex:



S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Why Is 3NF Good?

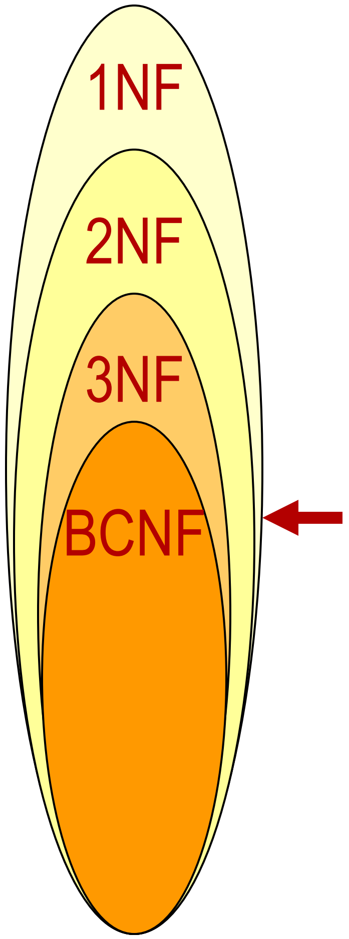
- If 3NF violated by $X \rightarrow A$, one of the following holds:
- X subset of some key K
  - We store (X, A) pairs redundantly
- X not a proper subset of any key
  - Which means: for some key K, there is a chain of FDs $K \rightarrow X \rightarrow A$
  - Which means: we once introduced keys to capture dependencies, but now we have attributes dependent on a non-key attribute!
- …so non-3NF means dangerous updates!

What Does 3NF NOT Achieve?

- Some redundancy possible with 3NF
- Ex: Reserves SBDC, S → C, C → S
  - is in 3NF
  - but S ↔ C means:
    for each reservation of sailor S, same (S, C) pair is stored
- …so we still need to capture "nests" inside the keys

Boyce-Codd Normal Form (BCNF)

- Relation R with FDs *F* is in BCNF if, for all X → A in F^+,
  - Either A∈X (called a *trivial* FD)
  - Or X contains a key for R
  - ~~Or A is part of some key for R~~
- In other words:
  R in BCNF ⇔ only key-to-nonkey constraints FDs left
  - ✓ = No redundancy in R that can be detected using FDs alone
  - ✓ = No FD constraints "hidden in data"



Discussion: 3NF vs. BCNF

- Always possible?
  - 3NF always possible, is “nice” (lossless-join, dependency-preserving)
  - BCNF not always possible
- 3NF compromise used when BCNF not achievable
  - Ex: performance considerations
  - Ex: cannot find ``good’’ decomp (see next)

Decomposition of a Relation Scheme

- Given relation R with attributes A1 ... An
- decomposition of R = replacing R by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of R (and no additional attributes), and
  - Every attribute of R appears as an attribute of one of the new relations
- E.g., decompose SNLRWH into SNLRH and RW

Example Decomposition

- SNLRWH has FDs
  S → SNLRWH, R ↔ W, N → SN
- 2nd FD causes 3NF violation:
  W values repeatedly associated with R values (and vice versa)!

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

- Easiest fix: create relation RW to store assocs w/o dups, remove W from main schema
  = decompose SNLRWH into SNLRH and RW

Wages

R	W
8	10
5	7

Hourly_Emps2

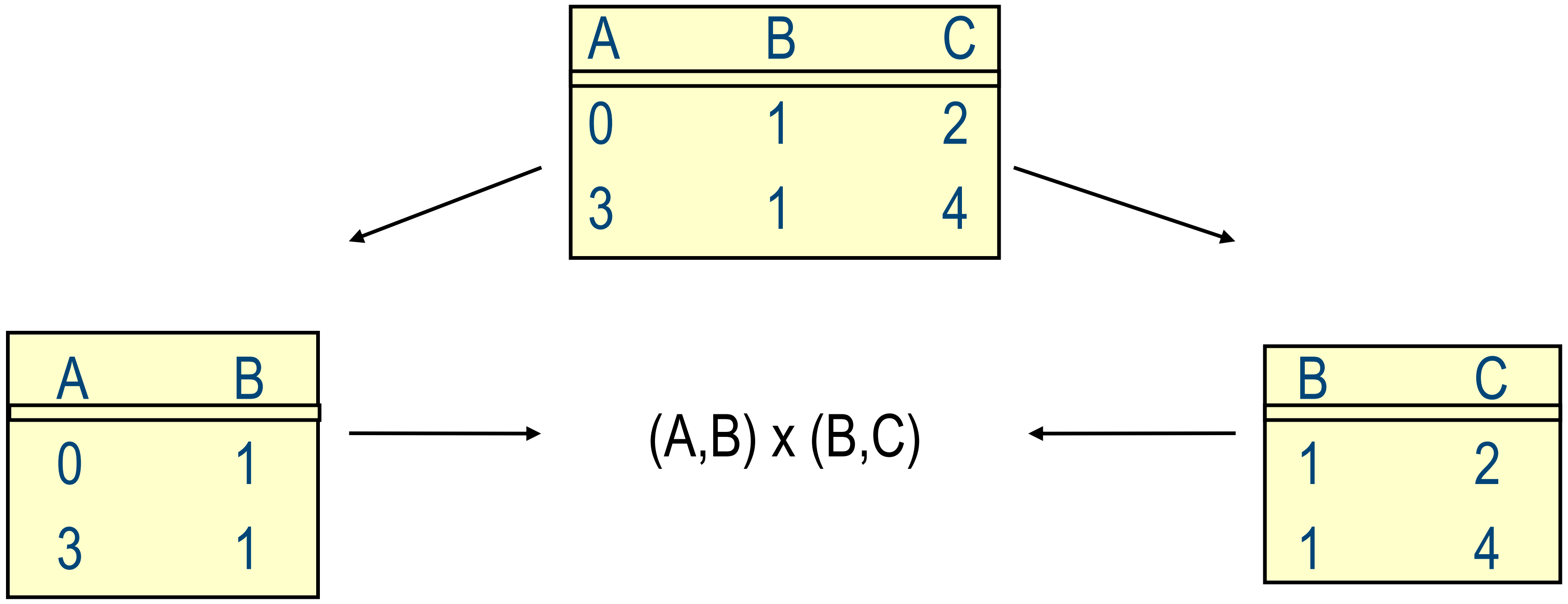
S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

If we just store projections of SNLRWH tuples onto SNLRH and RW, are there any potential problems?

3 Potential Problems with Decomp

- Some queries become more expensive
  - e.g., How much did sailor Joe earn? (salary = W*H)
- may not be able to reconstruct original relation
  - Fortunately, not in the SNLRWH example
  - ⮱

Lossless Join: A Counter Example



A	B	C
0	1	2
3	1	4

A	B
0	1
3	1

B	C
1	2
1	4

(A,B) x (B,C)

What's wrong?

3 Potential Problems with Decomp

- Some queries become more expensive
  - e.g., How much did sailor Joe earn? (salary = W*H)
- may not be able to reconstruct original relation ↳
  - Fortunately, not in the SNLRWH example
- Checking some dependencies may require joining decomposed relations
  - Fortunately, not in the SNLRWH example

- Tradeoff: Must consider these issues vs. redundancy

Summary of Schema Refinement

- BCNF = free of redundancies that can be detected using FDs
  - BCNF good heuristic (consider typical queries!)
    - *Check FDs !*
  - Next best: 3NF
- When not BCNF?
  - not always possible
  - unsuitable, given typical queries - performance requirements
- Use decompositions only when needed!

↳NF pocket guide

Pocket Guide to NFs

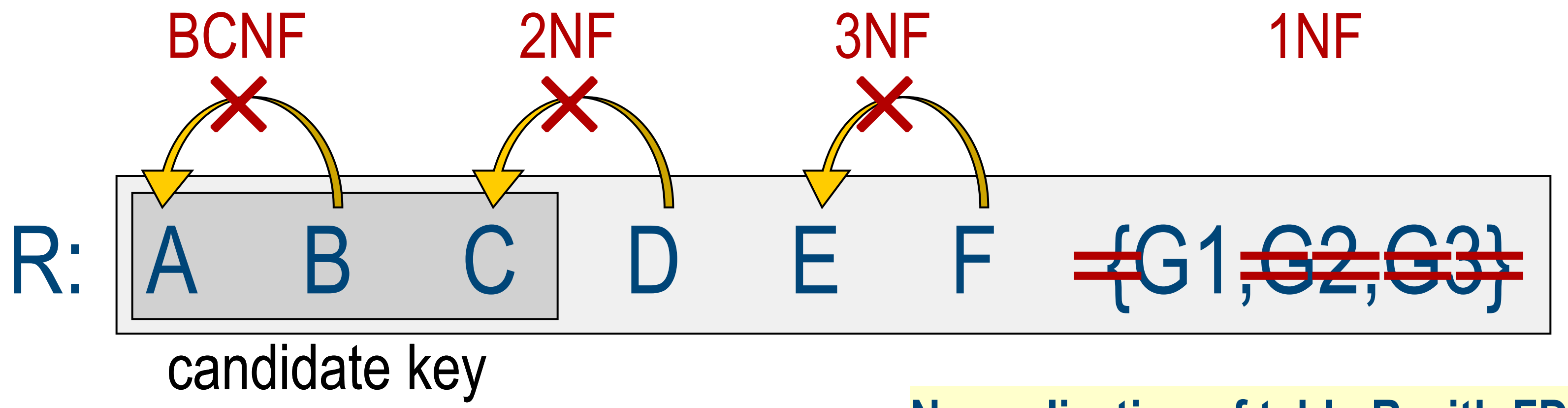
- 1NF = ...
- 2NF = 1NF + ...
- 3NF = 2NF + ...
- BCNF = 3NF + ...

Normalization of table R with given FD set :

- For all FDs F = „X → Y“:
  - Create additional table $R_F(X,Y)$
  - Remove Y from R, but keep X
- Do same for all tables created until all FDs are addressed

$$R:\ \underbrace{\boxed{A\quad B\quad C}}_{\text{candidate key}}\quad D\quad E\quad F\quad \{G1,G2,G3\}$$

Pocket Guide to NFs



- 1NF = no repeating groups
- 2NF = 1NF + no non-key → key
- 3NF = 2NF + no non-key → non-key
- BCNF = 3NF + no key → key

Normalization of table R with FD set :

- For all FDs F = „X → Y“:
  - Create additional table $R_F(X,Y)$
  - Remove Y from R, but keep X
- Drop duplicates arising from „X → Y, Y → X“ cycles
- Crosscheck all new tables created against all FDs for decomposition need

Road Map

JACOBS UNIVERSITY

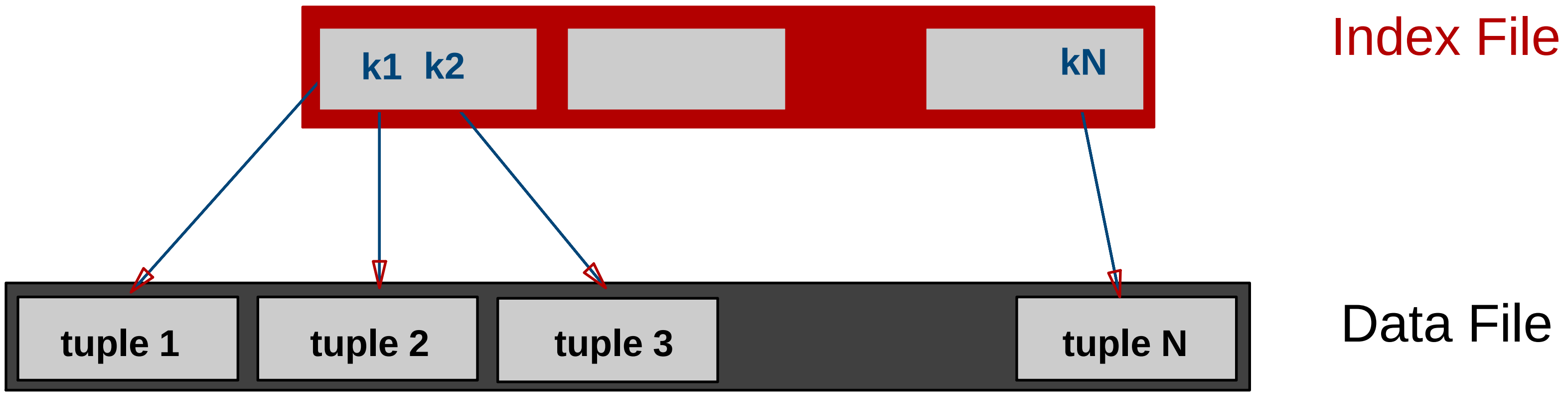
- Normal Forms
  - Functional Dependencies
  - Normal Forms
  - Decomposition
- Physical database design
  - Indexing
  - Tuning

Alternative File Organizations

- Basic storage mapping: Table stored sequentially in a file
  - How to organise for best search performance?
- Many alternatives
  - each ideal for some situations, and not so good in others:
- Heap (random order) files
  - Suitable when typical access is file scan retrieving all records
- Sorted Files
  - Best if records retrieved in some order, or only `range’ of records needed
  - Updates expensive
- Indexes = aux data structures to quickly address records by key
  - Only index search key fields

Range Searches

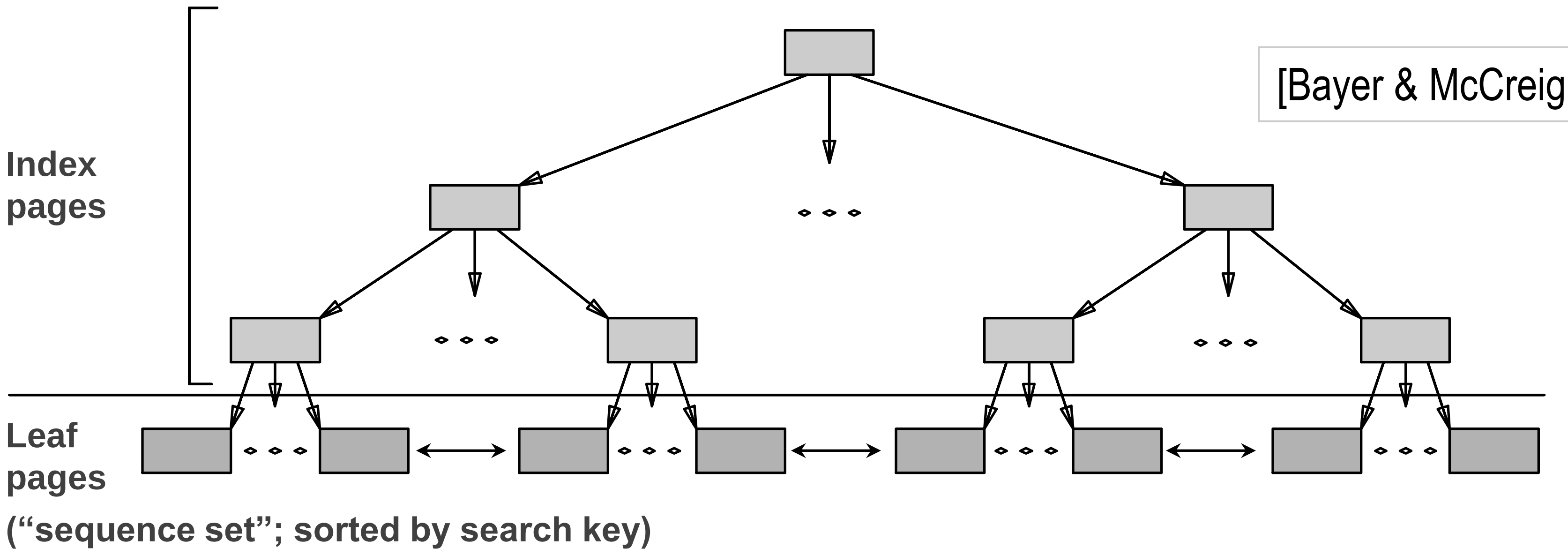
- *``Find all students with gpa > 3.0''*
  - sorted file (by gpa!), fixed-length records:
    binary search to find first student, then scan to find rest
  - Cost of binary search can be quite high
- Simple idea:
  Create an `index' file containing only key values + search values
  - Can do binary search on (smaller) index file!



Indexes

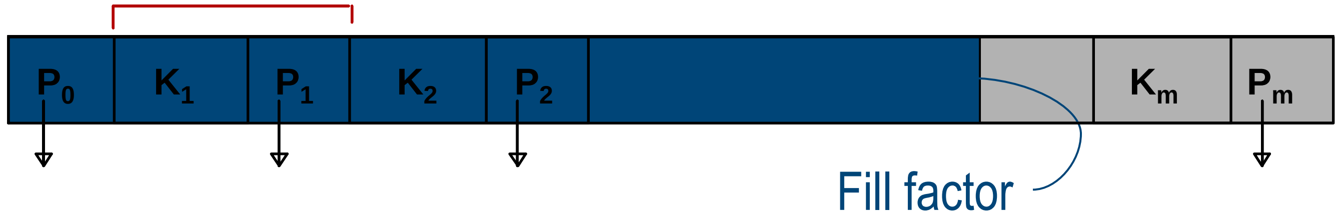
- speeds up selections on predefined search key fields
  - Index always on one relation (~file)
  - Any attribute can be search key for an index on the relation
- contains collection of data entries,
  supports efficient retrieval of all data entries k* with a given key value k
  - Ideally, in at most one disk I/O (details soon …)

B+ Tree Indexes

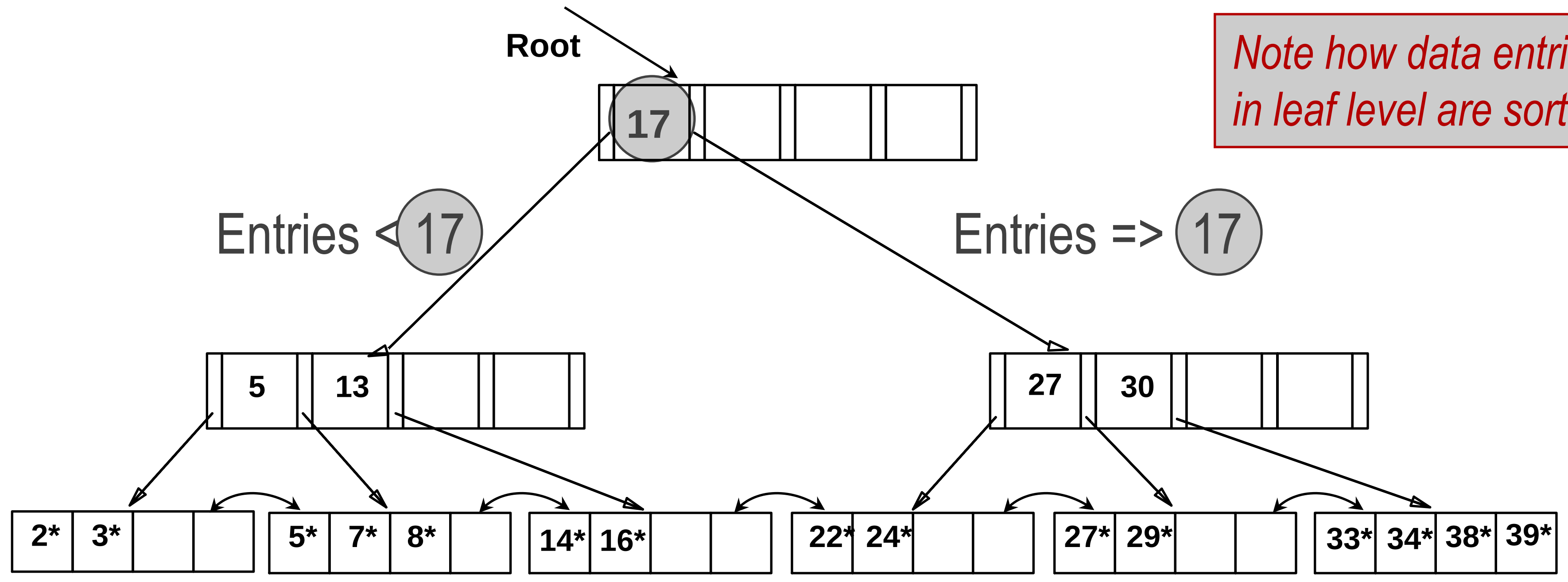


[Bayer & McCreight, 1972]

- Ordered Tree; Leaf pages contain data entries, are chained (prev & next)
- Non-leaf pages have index entries; only used to direct searches:



Example B+ Tree



Note how data entries in leaf level are sorted

- Find 28*? 29*? All > 15* and < 30*?
- Insert/delete: Find data entry in leaf, change it; adjust parent if needed
  - change sometimes bubbles up the tree
  - O($\log_F N$) where F = fan-out, N = # leaf pages

B+ Trees in Practice

- Typical fill-factor: 67%
- Average fanout: 133
- Typical capacities:
  - Height 3: 133^3 = 2,352,637 records
  - Height 4: 133^4 = 312,900,700 records
- Can often hold top levels in buffer pool:
  - Level 1 = 1 page = 8 Kbytes
  - Level 2 = 133 pages = 1 Mbyte
  - Level 3 = 17,689 pages = 133 MBytes

Hash-Based Indexes

- Goal: *compute* address without disk access, i.e., in O(1)
- Idea: distribute data evenly into fixed number of “buckets”
  - Compute location from key via Hashing function h: key $\rightarrow$ bucket
  - Example hashing function: h(int r) = r**a* mod *b*   with *b* prime relative to *a*
  - If keys match same address: overflow pages
- Hash index = collection of buckets + hashing function
  - Bucket = primary page plus zero or more overflow pages
  - Buckets contain data entries
- Good for equality, no support for range queries

Index Selection Guidelines

- Understand workload:
  - Queries vs. update
  - What relations (sizes!), attributes, conditions, joins (selectivity!), …?
- Attributes in WHERE clause are candidates for index keys
  - Exact match condition suggests hash index, range query suggests tree index
  - Consider multi-attribute search keys for several WHERE clause conditions
    - *Order of attributes important for range queries*
- Choose indexes that benefit as many queries as possible
  - impact on updates: Indexes make queries faster, updates slower
  - require disk space
- *understand how DBMS evaluates queries & creates query evaluation plans*

Summary

- Many alternative file organizations, each appropriate in some situation
- Index = collection of data entries
  plus a way to quickly find entries with given key values
- If selection queries are frequent, sort file or build an index
  - Hash indexes only good for equality search
  - Sorted files and tree indexes best for range search; also good for equality search
  - Files rarely kept sorted in practice; B+ tree index is better
- Understand workload and DBMS query plans

Road Map

- Normal Forms
  - Functional Dependencies
  - Normal Forms
  - Decomposition
- Physical database design
  - Indexing
  - Tuning

Decisions to Make

- What indexes?
  - Which relations? What field(s) search key? Several indexes?
  - For each index, what kind of an index should it be?
- Change conceptual schema?
  guided by workload, in addition to redundancy issues
  - Consider alternative normalized schemas? (many choices!)
  - "undo" some decompositions, settle for a lower normal form, such as 3NF? (denormalization)
  - Horizontal partitioning, replication, views ...see manuals
- If made after a database is in use, called schema evolution

Example Schemas

Contracts (<u>Cid</u>, Sid, Jid, Did, Pid, Qty, Val)
Depts (<u>Did</u>, Budget, Report)
Suppliers (<u>Sid</u>, Address)
Parts (<u>Pid</u>, Cost)
Projects (<u>Jid</u>, Mgr)

- Contracts = CSJDPQV; ICs: $JP \rightarrow C$, $SD \rightarrow P$; C is primary key
  - candidate keys for CSJDPQV?
  - What normal form is this relation schema in?

Denormalization

Contracts (<u>Cid</u>, Sid, Jid, Did, Pid, Qty, Val)

- Suppose following query is important:
  - *"Is the value of a contract less than the budget of the department?"*
- To speed up, add field budget B (from Departments) to Contracts
  - New FD for Dept./Budget: $D \rightarrow B$
  - Contracts no longer in 3NF
- might choose to modify Contracts if query is sufficiently important, and cannot obtain good performance otherwise
  - i.e., by indexes, choosing alternative 3NF schema

Masking Conceptual Schema Changes

```
CREATE VIEW  Contracts(cid, sid, jid, did, pid, qty, val)
          AS
              SELECT  *  FROM  LargeContracts
            UNION
              SELECT  *  FROM  SmallContracts
```

- replacement of Contracts by LargeContracts and SmallContracts can be masked by view
- However, queries with the condition val>10000 must be asked wrt LargeContracts for efficient execution:
  so users concerned with performance have to be aware of the change!

Tuning Queries and Views

- If a query runs slower than expected,
  check if index needs to be re-built or statistics too old
- Sometimes, DBMS may not be executing the plan you had in mind.
  Common areas of weakness:
  - Selections involving null values
  - Selections involving arithmetic or string expressions
  - Selections involving OR conditions
  - Lack of evaluation features like index-only strategies or certain join methods or poor size estimation
- Check plan used, adjust choice of indexes or rewrite query/view
  - Avoid nested queries, temporary relations, complex conditions, and operations like DISTINCT and GROUP BY

Outlook: Spatial Data Management

Points(X number, Y number, ptType: integer)

- Spatial data
  = multi-dimensional data
  - Objects regions have location
  - [+ spatial extent, ie, boundary]
- 2 fundamentally distinct categories:
  - Vectorial: point, line, region data in n-dimensional space
  - Raster: n-D “images” = arrays
- Not only spatio-temporal data:
  Also feature vectors extracted from text/images = non-spatial data!
  - Usually *very* high-dimensional, 1000s

Types of Multidimensional Queries

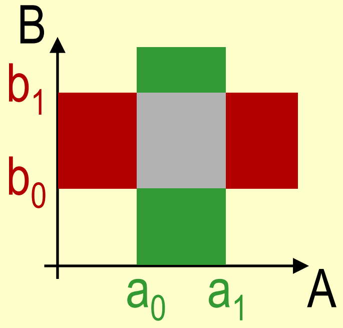
- Point Queries
  - *"Show Bremen"*
- Spatial Range Queries
  - *"Find all cities within 50 km of Bremen"*
  - Query has associated region (location, boundary)
- Nearest-Neighbor Queries
  - *"Find the 10 cities nearest to Bremen"*
  - Results must be ordered by proximity
- Spatial Join Queries
  - *"Find all cities near a lake"*
  - Expensive; join condition involves regions and proximity!
- Similarity queries
  - content-based retrieval
  - "Given a face, find the five most similar faces"
- *…plus aggregation, and several more*

Multiple B+ Trees?

- Query example:

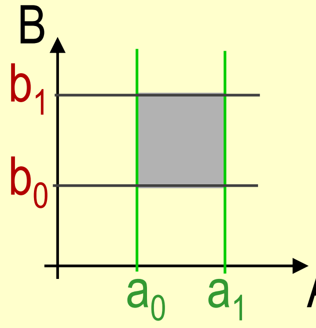
```
select * from R where a0 < A < a1 and b0 < B < b1
```

Several conventional indexes:



- read tuple with $a_0<A<a_1$
- read tuple with $b_0<B<b_1$
- intersect

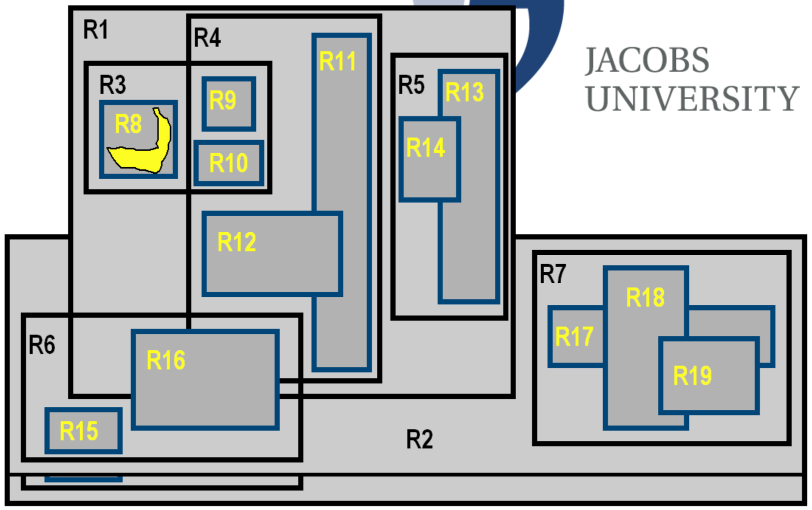
wanted:



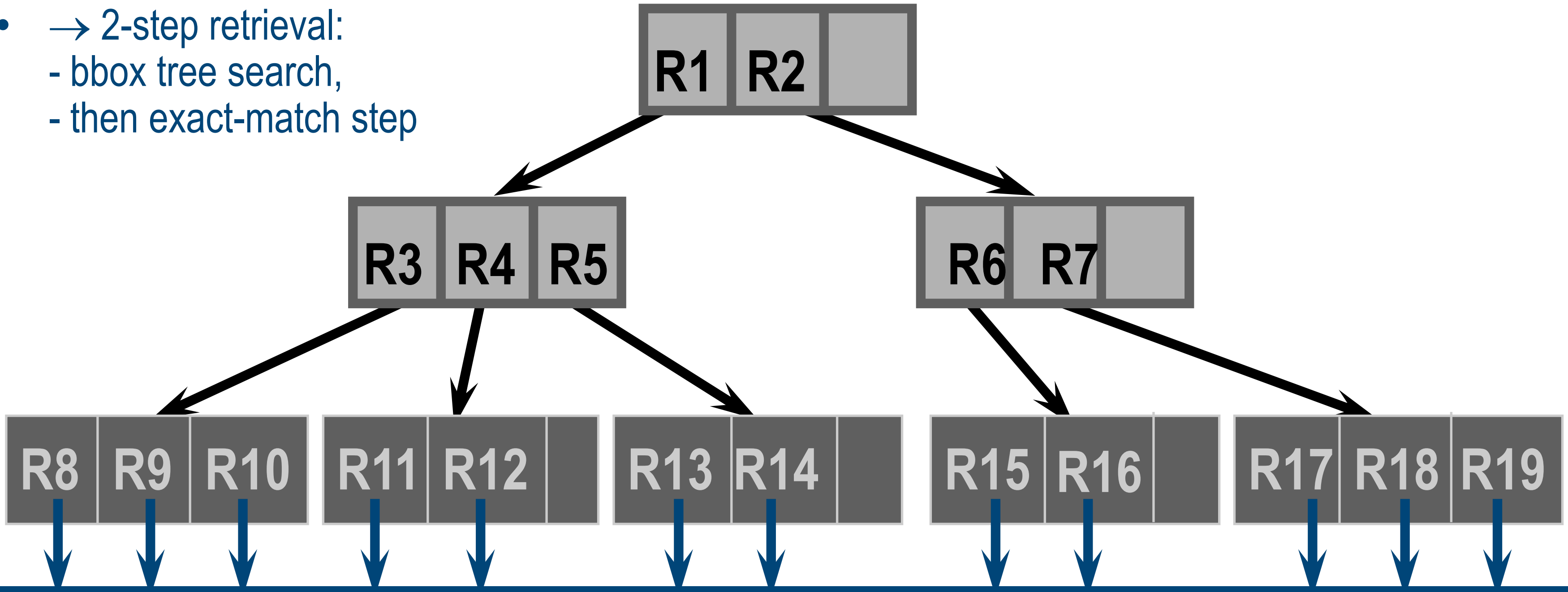
read only tuples with $a_0<A<a_1$ <u>and</u> $b_0<B<b_1$

- Specific family of n-D ("spatial") indexing techniques
  - R-tree = balanced tree; widely used in GIS
  - Grid Files, Quad trees, “space-filling” curves, …

R-Tree



- tree-structured n-D index [Guttman 1984]
- Index value = bounding box
  - Node's box covers its subtree
  - we do not search exact object boundaries, but their bounding boxes
  - → 2-step retrieval:
    - bbox tree search,
    - then exact-match step



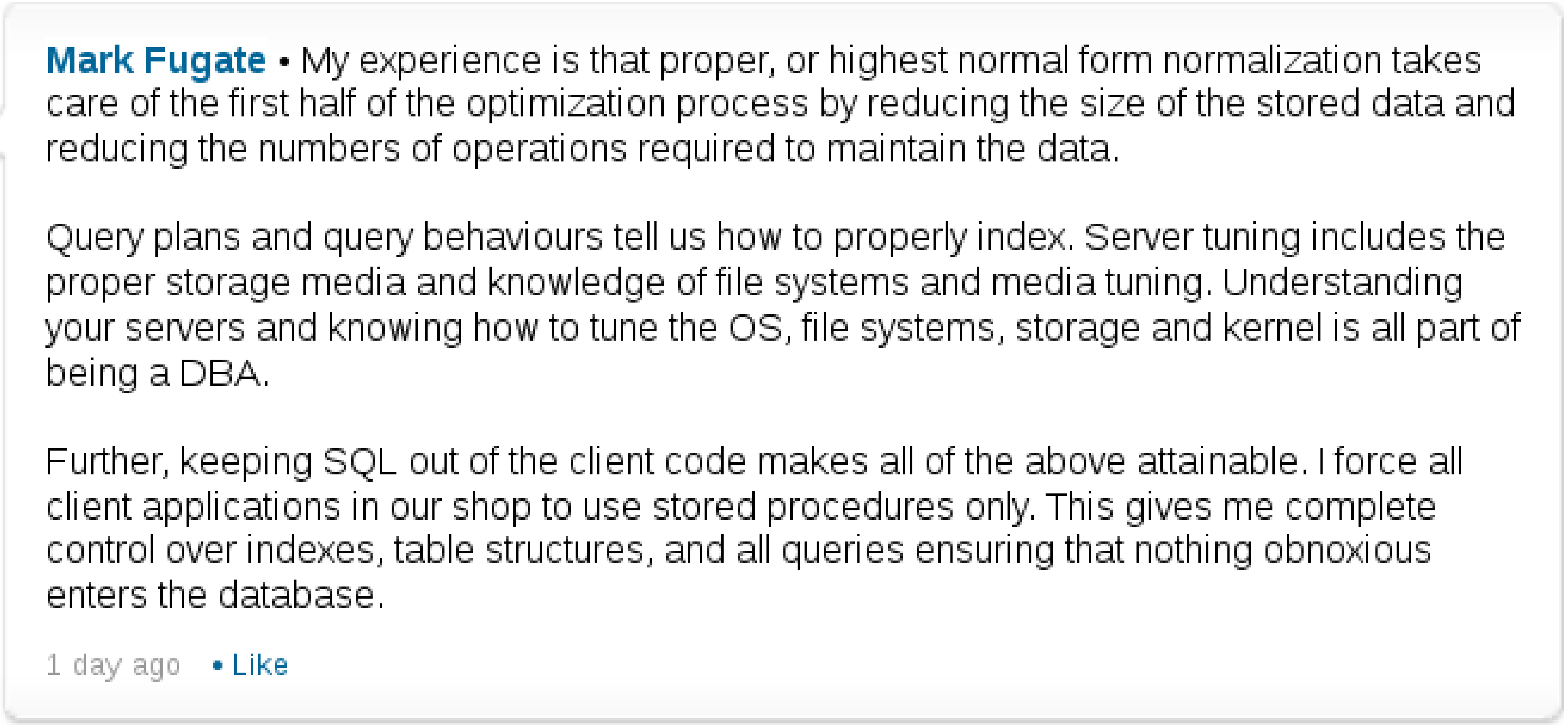
Applications of Multidimensional Data

- Geographic Information Systems (GIS)
  - Geospatial information; service standards by Open GeoSpatial Consortium (OGC)
  - Vendors: ESRI, Intergraph, SmallWorld, …, Oracle, …; open-source: Grass, PostGIS, …
  - All classes of spatial queries and data are common
- Computer-Aided Design / Manufacturing
  - spatial objects, ex: surface of airplane fuselage
  - Range queries and spatial join queries are common
- Multimedia Databases
  - Images, video, text, etc. stored and retrieved by content
  - First converted to *feature vector* form; high dimensionality
  - Nearest-neighbor queries are the most common

PS: A Moderately Complex Query

```
SELECT stadtbezirk, stadtteil, name, stadtteilchar, 'touche' AS entstehung, the_geom FROM
    (SELECT foo3.stadtbezirk, foo3.stadtteil, foo3.name, foo3.stadtteilchar, foo3.the_geom FROM
        (SELECT foo.gid, max(foo.laengste) AS laengste FROM
            (SELECT a.gid, b.stadtbezirk, b.stadtteil, b.name, b.stadtteilchar,
                (ST_Length(ST_Intersection(a.the_geom, ST_Union(b.the_geom)))) AS laengste
                FROM symdif a, dump b
                GROUP BY a.gid, a.the_geom, b.stadtbezirk, b.stadtteil, b.name, b.stadtteilchar
                HAVING ST_Touches(a.the_geom, ST_Union(b.the_geom))
                ORDER BY a.gid) AS foo
        GROUP BY foo.gid) AS foo2
        ,
        (SELECT a.gid, b.stadtbezirk, b.stadtteil, b.name, b.stadtteilchar, a.the_geom AS the_geom,
            (ST_Length(ST_Intersection(a.the_geom, ST_Union(b.the_geom)))) AS laengste
            FROM symdif a, dump b
            GROUP BY a.gid, a.the_geom, b.stadtbezirk, b.stadtteil, b.name, b.stadtteilchar
            HAVING ST_Touches(a.the_geom, ST_Union(b.the_geom))) AS foo3
    WHERE (foo2.gid = foo3.gid AND foo2.laengste = foo3.laengste)
    GROUP BY foo2.gid, foo3.stadtbezirk, foo3.stadtteil, foo3.name, foo3.stadtteilchar,
        foo3.laengste, foo2.laengste, foo3.the_geom) AS foo4
;
```

Key Performance Factors


Mark Fugate • My experience is that proper, or highest normal form normalization takes care of the first half of the optimization process by reducing the size of the stored data and reducing the numbers of operations required to maintain the data.

Query plans and query behaviours tell us how to properly index. Server tuning includes the proper storage media and knowledge of file systems and media tuning. Understanding your servers and knowing how to tune the OS, file systems, storage and kernel is all part of being a DBA.

Further, keeping SQL out of the client code makes all of the above attainable. I force all client applications in our shop to use stored procedures only. This gives me complete control over indexes, table structures, and all queries ensuring that nothing obnoxious enters the database.

1 day ago • Like

- Ref: discussion "what are the key points to improve the query performance" on the LinkedIn Database list, 2012-07-20

Summary

- Database design consists of several tasks:
  - requirements analysis,
  - conceptual design,
  - schema refinement,
  - physical design and tuning
- In general, have to go back & forth to refine database design; decisions in one task can influence the choices in another task
  - May choose 3NF or lower normal form over BCNF
  - May choose among alternative decompositions into BCNF (or 3NF) based upon the workload
  - *…and many techniques more*
- System may still not find a good plan – may have to rewrite query/view