Information Systems

Relational Databases

Temur Kutsia

Research Institute for Symbolic Computation
Johannes Kepler University of Linz, Austria
kutsia@risc.uni-linz.ac.at

Outline

Normalization

Goals:

- Force better database design.
- Eliminate data redundancy.
- Make data retrieval more efficient.

Normalization

- A relation is in a normal form iff it satisfies conditions prescribed to the normal form.
- A relvar is normalized as long as its legal value is a normalized relation.
- In this course we consider four normal forms:
  - 1NF,
  - 2NF,
  - 3NF,
  - BCNF (Boyce-Codd Normal Form).

Normalization

- Normalization procedure: Successive reduction of a collection of relvars to some normal form.
- Normalization procedure is reversible: It is possible to map the output form of the procedure back to the input form.
- Reversibility is important: No information is lost during normalization.
- Normalization process is nonloss or information-preserving.

Nonloss decomposition

- Normalization procedure involves decomposing a given relvar into other relvars.
- Decomposition is required to be reversible.
- The only decompositions we are interested in are nonloss.
- This concept is related with functional dependencies.

Decomposition

Relvar S and two corresponding decompositions:

S

S#	STATUS	CITY
S3	30	Paris
S5	30	Athens

(a) SST

S#	STATUS
S3	30
S5	30

SC

S#	CITY
S3	Paris
S5	Athens

(b) SST

S#	STATUS
S3	30
S5	30

STC

STATUS	CITY
30	Paris
30	Athens

Decomposition

Relvar S and two corresponding decompositions:

S

S#	STATUS	CITY
S3	30	Paris
S5	30	Athens

(a) SST

S#	STATUS
S3	30
S5	30

SC

S#	CITY
S3	Paris
S5	Athens

(b) SST

S#	STATUS
S3	30
S5	30

STC

STATUS	CITY
30	Paris
30	Athens

- In Case (a), no information is lost.
- SST and SC still say that S3 has status 30 and city Paris, and S5 has status and city Athens.
- Nonloss decomposition.

Decomposition

Relvar S and two corresponding decompositions:

S

S#	STATUS	CITY
S3	30	Paris
S5	30	Athens

(a) SST

S#	STATUS
S3	30
S5	30

SC

S#	CITY
S3	Paris
S5	Athens

(b) SST

S#	STATUS
S3	30
S5	30

STC

STATUS	CITY
30	Paris
30	Athens

- In Case (b), information is lost.
- We can still say that both suppliers S3 and S5 have status 30, but cannot tel which supplier has which city.
- Lossy decomposition.

Decomposition

Relvar S and two corresponding decompositions:

S

S#	STATUS	CITY
S3	30	Paris
S5	30	Athens

(a) SST

S#	STATUS
S3	30
S5	30

SC

S#	CITY
S3	Paris
S5	Athens

(b) SST

S#	STATUS
S3	30
S5	30

STC

STATUS	CITY
30	Paris
30	Athens

- The process of decomposition is actually a process of projection.
- SST, SC, and STC are projections of S.
- In Case (a), if we join SST and SC back again, we obtain S.
- In Case (b), joining SST and STC does not give S.

Decomposition

- Reversibility means that the original relvar is equal to the join of its projections.
- Hence, in normalization process:
  - Decomposition is projection
  - Recomposition is join.

Nonloss Decomposition

- Assume:
  - R1 and R2 are both projections of some relvar R
  - R1 and R2 between them include all of the attributes of R
- Question:
  - What conditions must be satisfied to guarantee that joining R1 and R2 back together takes us back to the original R?

Nonloss Decomposition

Relvar S and two corresponding decompositions. The (a) is nonloss:

S

S#	STATUS	CITY
S3	30	Paris
S5	30	Athens

(a) SST

S#	STATUS
S3	30
S5	30

SC

S#	CITY
S3	Paris
S5	Athens

(b) SST

S#	STATUS
S3	30
S5	30

STC

STATUS	CITY
30	Paris
30	Athens

Nonloss Decomposition

Relvar S and two corresponding decompositions. The (a) is nonloss:

S

S#	STATUS	CITY
S3	30	Paris
S5	30	Athens

(a) SST

S#	STATUS
S3	30
S5	30

SC

S#	CITY
S3	Paris
S5	Athens

(b) SST

S#	STATUS
S3	30
S5	30

STC

STATUS	CITY
30	Paris
30	Athens

- Functional dependencies.
- S satisfies the irreducible set of FD's:
  { S# $\rightarrow$ STATUS, S# $\rightarrow$ CITY }.
- It is not a coincidence that S is equal to the join of its projections {S#, STATUS}, {S#,CITY}.

Nonloss Decomposition

Theorem (Heath)

Let R{A,B,C} be a relvar, where A,B,C are sets of attributes. If R satisfies the FD A→B, then R is equal to the join of its projections on {A,B} and {A,C}.

Nonloss Decomposition

- We will say that the decomposition of a relvar R into projections R1,. . . ,Rn is nonloss, if R is equal to the join of R1,. . . ,Rn.
- In practice we would want to impose the additional requirement that R1,. . . ,Rn are all needed to join (to avoid redundancies)

Normal Forms

- First, we introduce an informal definition of 3NF, to give an idea where we are aiming at.
- Then consider the process of reducing of arbitrary relvar to an equivalent collection of 3NF's.

Normal Forms. 3NF. Informal Definition

- Third Normal Form: A relvar is in 3NF iff the nonkey attributes (if any) are both
  - Mutually independent and
  - Irreducibly dependent on the primary key.

Normal Forms. 3NF. Informal Definition

- Third Normal Form: A relvar is in 3NF iff the nonkey attributes (if any) are both
  - Mutually independent and
  - Irreducibly dependent on the primary key.
- A nonkey attribute: Any attribute that does not participate in the primary key.

Normal Forms. 3NF. Informal Definition

- Third Normal Form: A relvar is in 3NF iff the nonkey attributes (if any) are both
  - Mutually independent and
  - Irreducibly dependent on the primary key.
- A nonkey attribute: Any attribute that does not participate in the primary key.
- Attributes are mutually independent if none of them is functionally dependent on any combination of the others. Such independence implies that each attribute can be updated independently of the others.

Normal Forms

Example

- The parts relvar P in the suppliers-and-parts database is in 3NF:
  - The attributes PNAME, COLOR, WEIGHT, and CITY are all independent of one another (it is possible, e.g. to change the color of a part without simultaneously changing its weight)
  - They are all irreducibly dependent on the primary key P#.

Normal Forms. 1NF

- First Normal Form:
  - A relvar is in 1NF iff in every legal value of that relvar, every tuple contains exactly one value for each attribute.

Normal Forms. 1NF. Example

FIRST

S#	STATUS	CITY	P#	QTY
S1	20	London	P1	300
S1	20	London	P2	200
S1	20	London	P3	400
S1	20	London	P4	200
S1	20	London	P5	100
S1	20	London	P6	100
S2	10	Paris	P1	300
S2	10	Paris	P2	400
S3	10	Paris	P2	200
S4	20	London	P2	200
S4	20	London	P4	300
S4	20	London	P5	400

- Assume in the supplier-and-parts database S and SP are not split, but are lumped together in a single relvar (with some values slightly modified):
  FIRST { S#, STATUS, CITY, P#, QTY }
  PRIMARY KEY { S#, P# }

Normal Forms. 1NF. Example

FIRST

S#	STATUS	CITY	P#	QTY
S1	20	London	P1	300
S1	20	London	P2	200
S1	20	London	P3	400
S1	20	London	P4	200
S1	20	London	P5	100
S1	20	London	P6	100
S2	10	Paris	P1	300
S2	10	Paris	P2	400
S3	10	Paris	P2	200
S4	20	London	P2	200
S4	20	London	P4	300
S4	20	London	P5	400

- Let CITY$\rightarrow$STATUS be an additional constraint.

Normal Forms. 1NF. Example

FIRST

S#	STATUS	CITY	P#	QTY
S1	20	London	P1	300
S1	20	London	P2	200
S1	20	London	P3	400
S1	20	London	P4	200
S1	20	London	P5	100
S1	20	London	P6	100
S2	10	Paris	P1	300
S2	10	Paris	P2	400
S3	10	Paris	P2	200
S4	20	London	P2	200
S4	20	London	P4	300
S4	20	London	P5	400

- FIRST is in 1NF, but not in 3NF. Both conditions are violated:
  - The nonkey attributes are not all mutually independent (STATUS depends on CITY).
  - They are not all irreducibly dependent on the primary key (STATUS and CITY each depend on S# alone).

Normal Forms. 1NF. Example

FIRST

S#	STATUS	CITY	P#	QTY
S1	20	London	P1	300
S1	20	London	P2	200
S1	20	London	P3	400
S1	20	London	P4	200
S1	20	London	P5	100
S1	20	London	P6	100
S2	10	Paris	P1	300
S2	10	Paris	P2	400
S3	10	Paris	P2	200
S4	20	London	P2	200
S4	20	London	P4	300
S4	20	London	P5	400

- FIRST contains redundancies:
  - Every tuple for S1 shows city as London.
  - Every tuple for London shows status as 20.
- Too much information bundled together. Bad behavior on delete.

Normal Forms

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

SECOND

S#	STATUS	CITY
S1	20	London
S2	10	Paris
S3	10	Paris
S4	20	London
S5	30	Athens

- The solution of the problems related to FIRST: Replace FIRST by the two relvars:
  - SP { S#, P#, QTY }
  - SECOND { S#, STATUS, CITY }

Normal Forms

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

SECOND

S#	STATUS	CITY
S1	20	London
S2	10	Paris
S3	10	Paris
S4	20	London
S5	30	Athens

- Decomposition of FIRST into SP and SECOND eliminates redundancies.

Normal Forms. 2NF

The definition assumes only one candidate key, which is the primary key.

- Second Normal Form: A relvar is in 2NF iff
  - it is in 1NF and
  - every nonkey attribute is irreducibly dependent on the primary key.

Normal Forms

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

SECOND

S#	STATUS	CITY
S1	20	London
S2	10	Paris
S3	10	Paris
S4	20	London
S5	30	Athens

- SP and SECOND are both in 2NF. FIRST is not.

Normal Forms

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

SECOND

S#	STATUS	CITY
S1	20	London
S2	10	Paris
S3	10	Paris
S4	20	London
S5	30	Athens

- A relvar that is in 1NF and not in 2NF can always be reduced to an equivalent collection of 2NF relvars.
- Reduction: Replace the 1NF relvar by suitable projections. The obtained collection is equivalent to the original relvar.

Normal Forms

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

SECOND

S#	STATUS	CITY
S1	20	London
S2	10	Paris
S3	10	Paris
S4	20	London
S5	30	Athens

- SP and SECOND are projections of FIRST, and FIRST is the join of SECOND and SP over S#.

Normalization Procedure

- The first step in the normalization procedure: Take projections to eliminate “nonirreducible” functional dependencies.
- Given relvar R as follows
  R { A, B, C, D }
  PRIMARY KEY { A, B }
  /* assume A→D holds */
- Replace R by two projections R1 and R2:
  R1 { A, D }
  PRIMARY KEY { A }
  R2 { A, B, C }
  PRIMARY KEY { A, B }
  FOREIGN KEY { A } REFERENCES R1
- R can be recovered by taking the foreign-to-matching-primary-key join of R2 and R1.

Normal Forms

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

SECOND

S#	STATUS	CITY
S1	20	London
S2	10	Paris
S3	10	Paris
S4	20	London
S5	30	Athens

- SP is satisfactory. It is in 3NF.
- SECOND suffers from a lack of mutual independence among its nonkey attributes

Normal Forms

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

SECOND

S#	STATUS	CITY
S1	20	London
S2	10	Paris
S3	10	Paris
S4	20	London
S5	30	Athens

- Each S# value determines a CITY value, and each CITY value determines the STATUS value.
- STATUS transitively depends on S# via CITY.

Normal Forms

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

SECOND

S#	STATUS	CITY
S1	20	London
S2	10	Paris
S3	10	Paris
S4	20	London
S5	30	Athens

- If we delete the tuple for S5, we lose the information that the status of Athens in 30.
- The problem is “bundling” again. SECOND contains information regarding suppliers **and** regarding cities.
- Solution: “unbundle”.

Normal Forms

SC

S#	City
S1	London
S2	Paris
S3	Paris
S4	London
S5	Athens

CS

City	STATUS
Athens	30
London	20
Paris	10
Rome	50

- Replace SECOND by two projections:
  - SC { S#, CITY }
  - CS { CITY, STATUS}
- The effect is to eliminate the transitive dependence of STATUS on S#.

3NF

The definition assumes only one candidate key, which is the primary key.

- Third Normal Form: A relvar id in 3NF iff
  - it is in 2NF and
  - every nonkey attribute is nontransitively dependent on the primary key.
- No transitive dependencies imply no mutual dependencies.

Normal Forms

SC

S#	City
S1	London
S2	Paris
S3	Paris
S4	London
S5	Athens

CS

City	STATUS
Athens	30
London	20
Paris	10
Rome	50

- SC and CS are both in 3NF, with the primary keys {S#} and {CITY}, respectively.
- A relvar that is in 2NF but not in 3NF can be reduced to an equivalent collection of 3NF relvars.
- The reduction is reversible.

Normalization Procedure

- The second step in the normalization procedure: Take projections to eliminate transitive functional dependencies.
- Given relvar R as follows
  R { A, B, C }
  PRIMARY KEY { A }
  /* assume B→C holds */
- Replace R by two projections R1 and R2:
  R1 { B, C }
  PRIMARY KEY { B }
  R2 { A, B }
  PRIMARY KEY { A }
  FOREIGN KEY { B } REFERENCES R1
- R can be recovered by taking the foreign-to-matching-primary-key join of R2 and R1.

Boyce/Codd Normal Form

- Drop the assumption that every relvar has just one candidate key and consider the general case.
- Terms:
  - Determinant: The left hand side of an FD.
  - Trivial FD: An FD whose right hand side is a subset of its left hand side.
- Boyce/Codd Normal Form: A relvar is in BCNF iff every nontrivial, left-irreducible FD has a candidate key as its determinant.

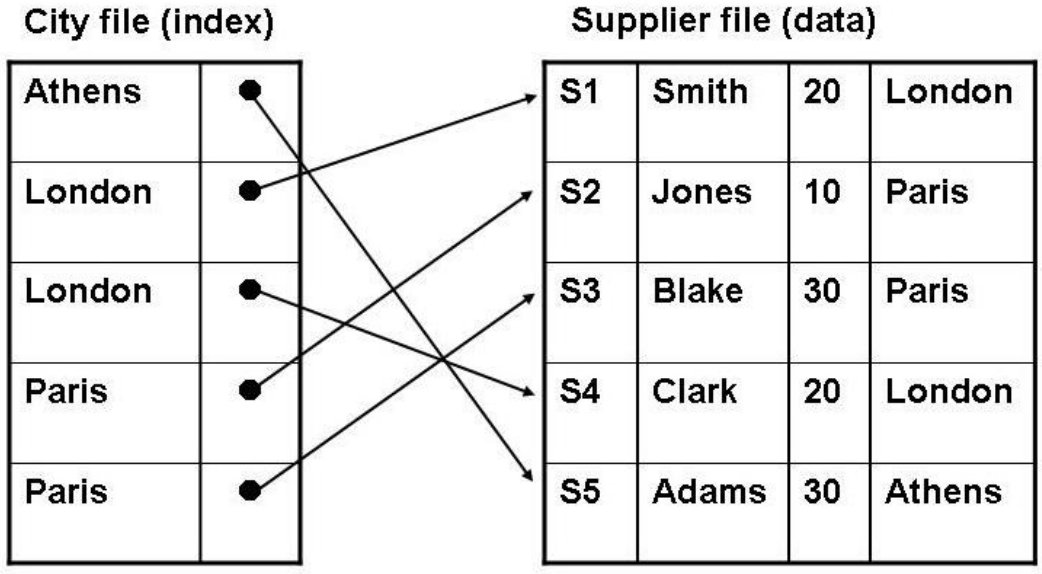
Remark

- There are also higher normal forms denoted by 4NF and 5NF (not treated in this lecture).
- The goal of database design is to reach BCNF, or if it seems to be too much effort for too little gain, to reach 3NF.
- Too much normalization can also decrease system performance, therefore sometimes denormalization is applied after reaching the higher normal forms.

Indexing

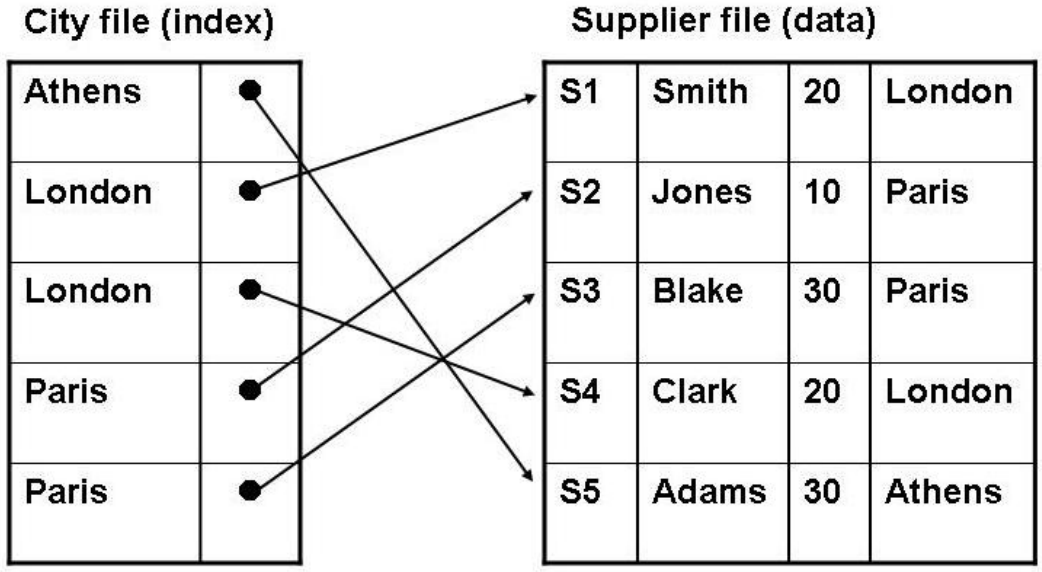
- Consider suppliers once again.
- Suppose the query “Get all suppliers in city c” (where c is a parameter) is an important one: frequently executed and therefore required to perform well.

Indexing



- The DBA might choose the stored representation with two files, a supplier file and a city file.
- In the city file CITY is the primary key, and it is ordered in city sequence.
- It includes pointers (RIDs) into the supplier file.

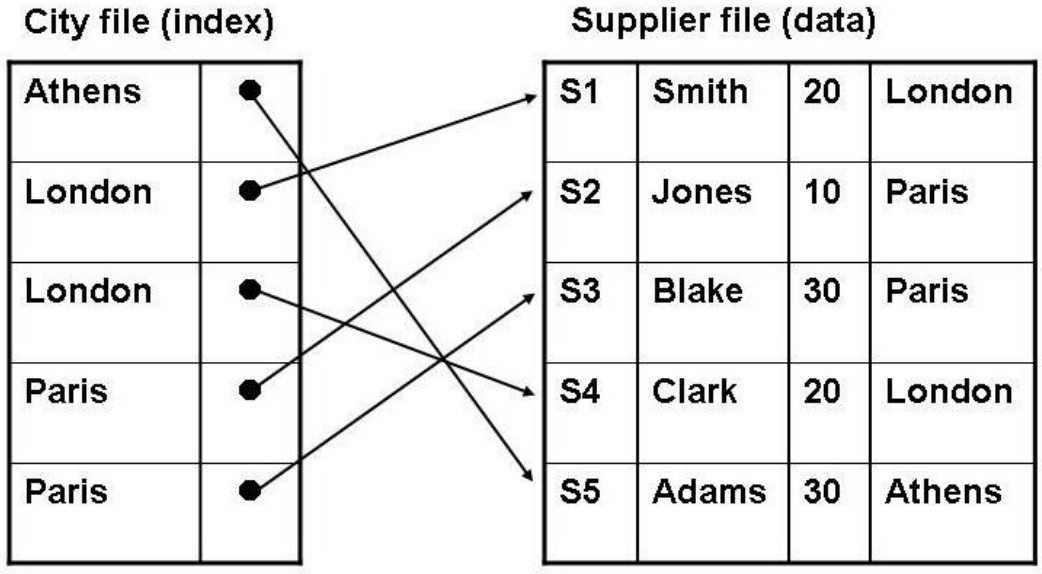
Indexing



DBMS has two possible strategies to get all suppliers in London:

1. Search the entire supplier file, looking for all records with city value equal to London.
2. Search the city file for the London entries, and for each such entry follow the pointer to the corresponding record in the supplier file.

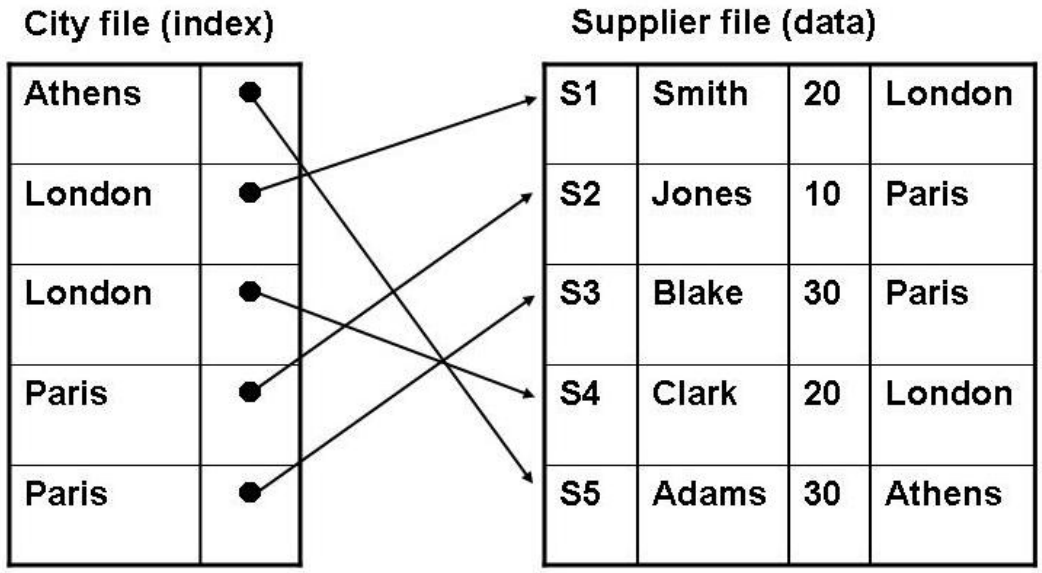
Indexing



If the ratio of London suppliers to others is small, the second strategy is likely to be more efficient, because

1. the DBMS can stop its search of that file as soon as it finds a city that comes later than London in alphabetic order, and
2. even if it did have to search the entire city file, that search would still probably require fewer I/O's overall, because the city file is physically smaller (smaller records).

Indexing



1. The city file in an index (the CITY index) to the supplier file.
2. The supplier file is indexed by the city file.

Indexing

- An index is a file in which each entry (i.e. each record) consists of precisely two values, a data value and a pointer (RID).
- The data value is a value for some field of the indexed file.
- The pointer identifies a record of that file that has that value for that field.
- The relevant field of the indexed file is called the indexed field.

Indexing. Pros and Cons

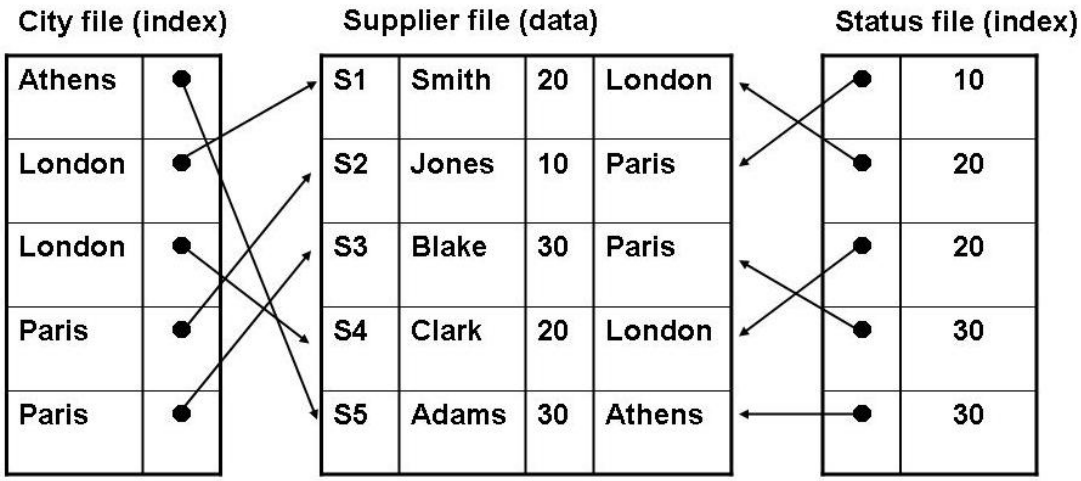
- The fundamental advantage of an index is that it speeds up retrieval.
- Indexing may help to perform existence tests.
- Disadvantage: It slows down updates.
- For instance, every time a new record is added to the indexed file, a new entry will also have to be added to the index.

Retrieval

Access types:

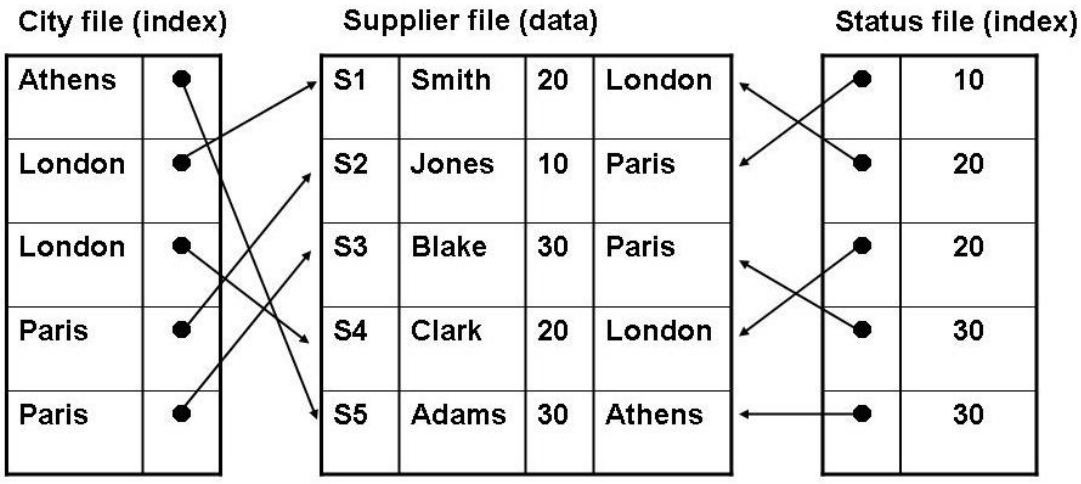
- Sequential access: The index can help with range queries, e.g. “Get suppliers whose city is in some specified alphabetic range”. Two important special cases:
  - “Get all suppliers whose city alphabetically precedes (or follows) some specified value,”
  - “Get all suppliers whose city is alphabetically first (or last).”
- Direct access: The index can help with list queries, e.g. “Get suppliers whose city is in some specified list” (London, Paris, and New York).

Indexing



- A given file can have any number of indexes, e.g. CITY index and a STATUS index.
- Efficient access to supplier records on the basis of given values for either or both of CITY and STATUS.

Indexing



- Illustration of the “both” case: The query “Get suppliers in Paris with status 30.”
- CITY index give two PIDs: r2 and r3. STATUS index gives r3 and r5.
- From them, the answer is obtained: r3.
- Only then DBMS retrieves the desired record (supplier S3).

Dense vs Sparse Indices

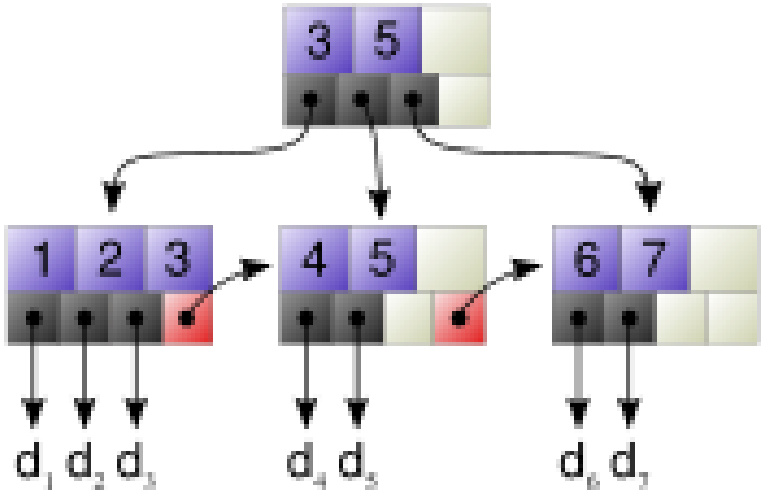
- Sparse index does not contain an entry for every record in the indexed file.
- By contrast, all indexes discussed prior to this point have been dense.
- Sparse index occupies less space than a corresponding dense index.
- As a result, it will probably be quicker to scan also.
- A disadvantage is that it might no longer be possible to perform existence tests on the basis of the index alone.

Multi-Level Indexing

- The reason for providing an index: to remove the need for physical sequential scanning of the indexed file.
- However, physical sequential scanning is still needed in the index.
- If the indexed file is very large, then the index can itself get to be quite sizable.
- Sequentially scanning the index can itself get to be quite time consuming.
- Solution: Build an index for an index.
- each level of the index acts as a sparse index for the level below.

B-Trees

- B-Trees: common and important kind of index.
- The index consists of two parts, the sequence set and the index set.
- The sequence set consists of a single-level index to the actual data. (Lowest level on the figure.)
- The index set, provides fast direct access to the sequence set. (The upper tree on the figure.)

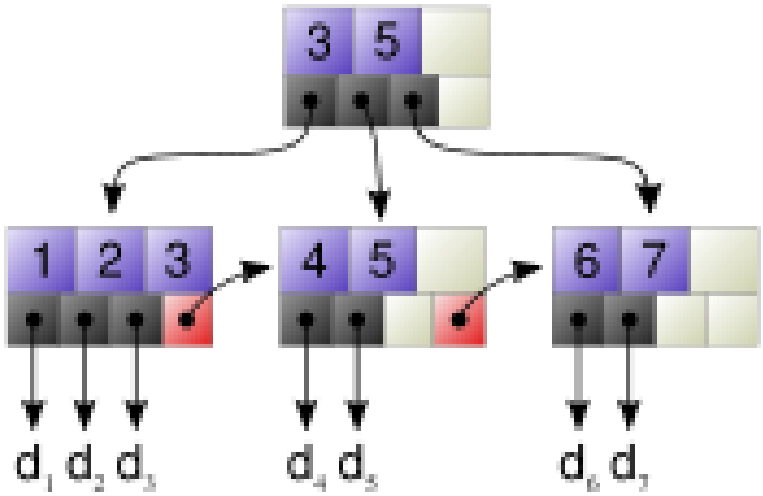


B-Trees

- The most important feature of B-trees is that they are always balanced
- Hence the depth of the tree grows only logarithmically in the growth of the indexed records.

The Rules for B-Trees

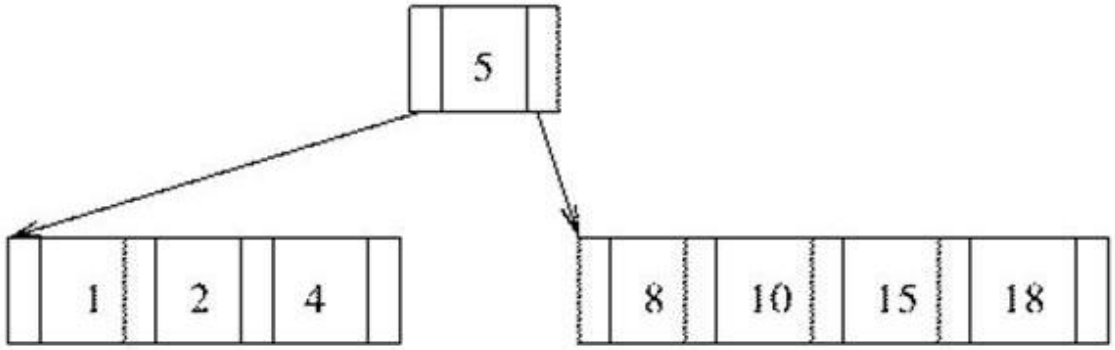
- To construct a B-tree, fix an integer n and require that
  - the root stores minimum 1 and maximum $2n$ key values,
  - each node of the tree, with the exception of the root, stores minimum n and maximum $2n$ key values,
  - leaf nodes are on the same level.
- The subtrees have the property that each value stored in them lie between the two values neighboring the pointer that links the subtree to the node.
- B-tree of order n.



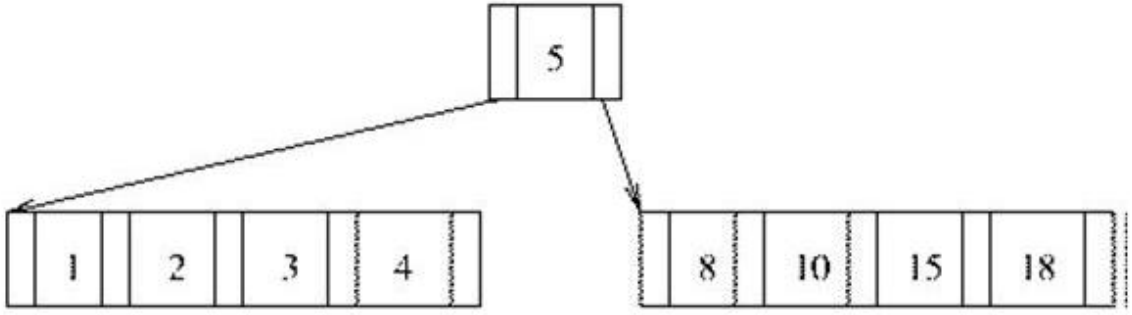
Insertion in a B-Tree

Example

B-tree of order 2:



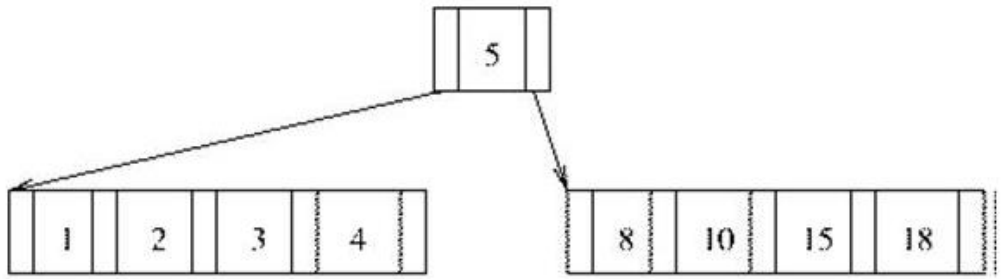
Insert 3:



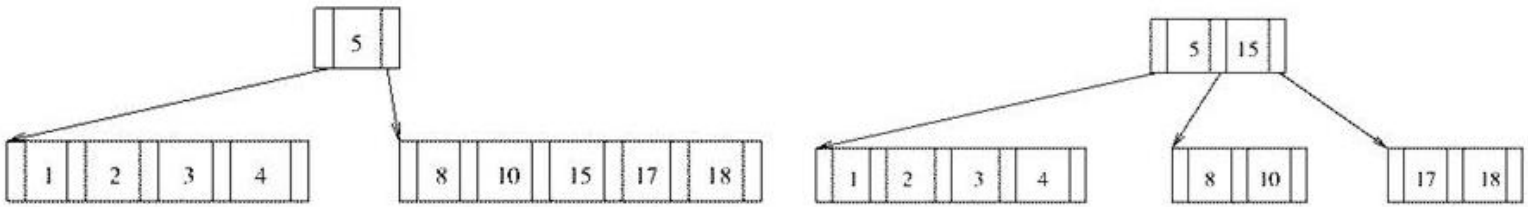
Insertion in a B-Tree

Example

B-tree of order 2:



Insert 17:



Summary

- First, second, third, and Boyce-Codd normal forms have been discussed.
- A given relvar can always be brought to BCNF.
- The normalization process consists of replacing a given relvar by certain projections in such a way that joining those projections back together again gives us back the original relvar.
- It means that the process is reversible.
- Functional dependencies play a crucial role in the process.
- Heath's theorem says that if a certain FD is satisfied, then a certain decomposition is nonloss.
- Moreover, indexes (in particular B-trees) and their use for both sequential and direct access have been discussed.