

Functional Dependencies

Fundamentals of Databases

Dr. Sandra Sampaio
School of Computer Science
University of Manchester

Acknowledgements

2

These slides are minor adaptations of material
authored and made available to instructors by

R. Ramakrishnan and J. Gehrke

to accompany their textbook

Database Management Systems, 3rd Edition

Copyright remains with them, whom I thank.

Any errors are my responsibility.

The Evils of Redundancy

3

- *Redundancy* is at the root of several problems associated with relational schemas:
 - redundant storage, insert/delete/update anomalies.
- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
- Main refinement technique: *decomposition* (replacing ABCD with, say, AB and BCD, or ACD and ABD).
- Decomposition should be used judiciously:
 - Is there a reason to decompose a relation?
 - What problems (if any) does the decomposition cause?



attributes in a relation

Functional Dependencies (FDs)

4

- A functional dependency (FD) $X \rightarrow Y$ holds over relation R if, for every allowable instance r of R:
 - $t1 \in r, t2 \in r, \pi_X(t1) = \pi_X(t2)$ implies $\pi_Y(t1) = \pi_Y(t2)$
 - i.e., given two tuples ($t1$ and $t2$) in r , if the X values agree, then the Y values must also agree. X and Y are sets of attributes.
- An FD is a statement about **all** allowable 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!
- If K is a **candidate key** for R, then $K \rightarrow R$ (we refer to all attributes of a relation using the relation name).
 - However, $K \rightarrow R$ **does not** require K to be **minimal!**

with the minimal possible number of attributes

Relation Hourly-Emps

5

<u>SSN</u>	Name	Lot	Rating	Hrly-Wage	Hrs-Worked
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

■ *ssn* is the key: $S \rightarrow \text{SNLRWH}$

initials for each attribute

■ *Rating* determines *Hrly_Wages*: $R \rightarrow W$

Example: Constraints on Entity Set - 1

6

- Consider relation Hourly_Emps from the previous slide:
 - Hourly_Emps (ssn, name, lot, rating, hrly_wages, hrs_worked)
- Notation: We will denote this relation schema by listing the attributes: SNLRWH
 - This is really the *set* of attributes {S,N,L,R,W,H}.
 - We will refer to all attributes of a relation by using the relation name. (e.g., Hourly_Emps for SNLRWH)
- Some FDs on Hourly_Emps:
 - *ssn* is the key: $S \rightarrow SNLRWH$
 - *rating* determines *hrly_wages*: $R \rightarrow W$

Example - 2

■ Problems due to $R \rightarrow W$:

- Update anomaly: Can we change W in just the 1st tuple of SNLRWH?
- Insertion anomaly: What if we want to insert an employee and don't know the hourly wage for his rating?
- Deletion anomaly: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

decomposition

Will 2 smaller tables be better?

					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		

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

Reasoning About FDs - 1

8

- Given some FDs, we can usually infer additional FDs:
 - $ssn \rightarrow did, did \rightarrow lot$ implies $ssn \rightarrow lot$
- An FD f is *implied by* a set of FDs F if f holds whenever all FDs in F hold.
 - $F^+ = \text{closure of } F$ is the set of all FDs that are implied by F .
- Armstrong's Axioms (X, Y, Z are sets of attributes):
 - *Reflexivity*: If $X \subseteq Y$, then $Y \rightarrow X$
 - *Augmentation*: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
 - *Transitivity*: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
- These are *sound* and *complete* inference rules for FDs!
 - **Sound**: given a set of FDs, F , specified on relation R , any dependency that we can infer from F holds in every instance of R that satisfies the dependencies in F .
 - **Complete**: using these axioms repeatedly to infer dependencies until no more dependencies can be inferred from F , results in a complete set of all possible dependencies that can be inferred from F .

Reasoning About FDs - 2

9

- Couple of additional rules (that follow from AA):
 - *Union*: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
 - *Decomposition*: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- Example: *Contracts*(*cid,sid,jid,did,pid,qty,value*), and:
 - C is the key: $C \rightarrow CSJDPQV$
 - Project purchases each part using single contract: $JP \rightarrow C$
 - Dept purchases at most one part from a supplier: $SD \rightarrow P$
- $JP \rightarrow C, C \rightarrow CSJDPQV$ imply $JP \rightarrow CSJDPQV$
- $SD \rightarrow P$ implies $SDJ \rightarrow JP$
- $SDJ \rightarrow JP, JP \rightarrow CSJDPQV$ imply $SDJ \rightarrow CSJDPQV$

Reasoning About FDs - 3

10

- 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 $X \rightarrow Y$ is in the closure of a set of FDs F .
 - Example: Does $F = \{A \rightarrow B, B \rightarrow C, C D \rightarrow E\}$ imply $A \rightarrow E$?
 - i.e, is $A \rightarrow E$ in the closure F^+ ?

Normal Forms

11

- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- 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.
- As well as detecting redundancies, the presence of FDs in a relation help us identify in which normal form the relation is and decide whether decomposing the relation is worthwhile.
 - Consider a relation R with 3 attributes, ABC.
 - **No FDs hold:** There is no redundancy here.
 - **Given $A \rightarrow B$:** Several tuples could have the same A value, and if so, they'll all have the same B value!

Normal Forms will be covered in the next lecture!

Summary

12

- Redundancy is at the root of several problems in relational schema design.
- Presence of redundancy can result in update, delete and insert anomalies.
- The detection of functional dependencies in a schema helps identify redundancy problems.
- Decomposition is the main schema refinement technique to overcome/minimize redundancies.
- Armstrong Axioms (AA) helps infer additional functional dependencies (FDs) from a set of FDs.
- The Reflexivity, Augmentation and Transitivity Axioms are sound and complete.
- Normal forms represent quality standards of relational schemas and can be identified by the presence of FDs.

Normalization

Fundamentals of Databases

Dr. Sandra Sampaio
School of Computer Science
University of Manchester

Acknowledgements

14

These slides are minor adaptations of material
authored and made available to instructors by

R. Ramakrishnan and J. Gehrke

to accompany their textbook

Database Management Systems, 3rd Edition

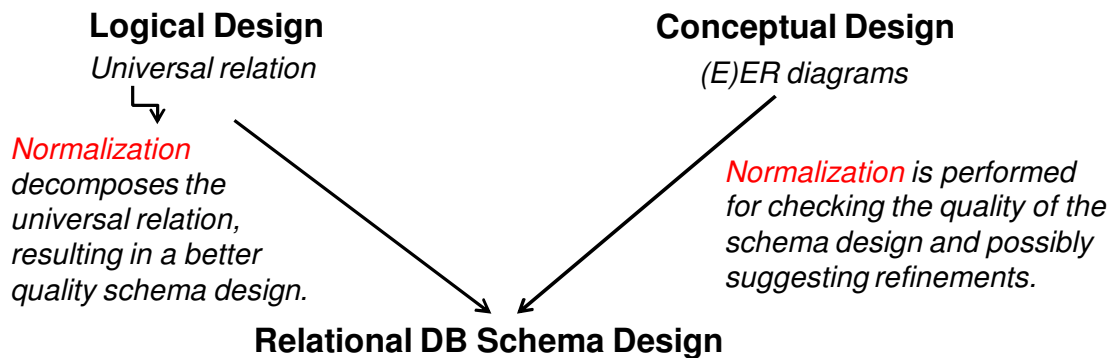
Copyright remains with them, whom I thank.

Any errors are my responsibility.

Normalization of Relations

15

- Normalization is the process of decomposing unsatisfactory (i.e., badly designed) relations by breaking up their attributes into smaller relations.
- A normal form is used to certify whether a relation schema is at a particular quality level in its design, by using the keys and functional dependencies (FDs) of the relation.
- 2NF, 3NF, BCNF normal forms are based on keys and FDs of a relation schema.



Practical Use of Normal Forms

16

- Normalization is carried out in practice so that the resulting schema designs are of high quality and meet the desirable properties.
- The practical utility of these normal forms becomes questionable when the constraints on which they are based are hard to understand or to detect.
- Database designers do not need to normalize to the highest possible normal form (usually up to 3NF or BCNF).
- De-normalization is the process of storing the join of higher normal form relations as a base relation, which is in a lower normal form.

Reviewing Material from Previous Lectures: Definitions of Keys and Prime Attributes

17

- A **superkey** of a relation schema $R = \{A_1, A_2, \dots, A_n\}$ is a set of attributes with the property that no two tuples t_1 and t_2 will have $t_1[S] = t_2[S]$
- A **key** K is a superkey with the additional property that removal of any attribute from K will cause K not to be a superkey any more.
- If a relation schema has more than one key, each is called a **candidate key**. One of the candidate keys is arbitrarily designated to be the primary key, and the others are called secondary keys.
- A **prime attribute** must be a member of some candidate key. A **nonprime attribute** it is not a member of any candidate key.
- A functional dependency $X \rightarrow Y$ is a **full functional dependency** if removal of any attribute A from X means that the dependency does not hold any more.
 - In other words, for any attribute A in X , $\{X - A\}$ does not functionally determine Y .

1st Normal Form (1NF) -Eliminate Multivalued Attributes (1)

18

Disallows attributes whose values for an individual tuple are **non-atomic**.

(a) DEPT

DNAME	<u>DNUMBER</u>	DMGRSSN	DLOCATION
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

1NF -Eliminate Multivalued Attributes(2)

19

(b) DEPT

DNAME	<u>DNUMBER</u>	DMGRSSN	DLOCATION
Research	5	333445555	{Bellaire}
Research	5	333445555	{Sugarland}
Research	5	333445555	{Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

The 1NF is a requirement implicit in the definition of Relational model, and, as such it must always hold. However, newer database systems are relaxing this requirement. But this is out of the scope of this course...

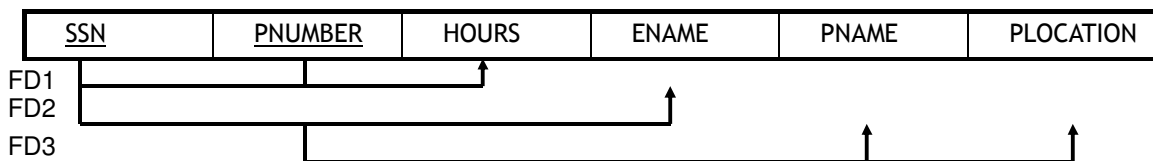
How would you decompose relation DEPT?

2NF -Eliminate Redundant Data

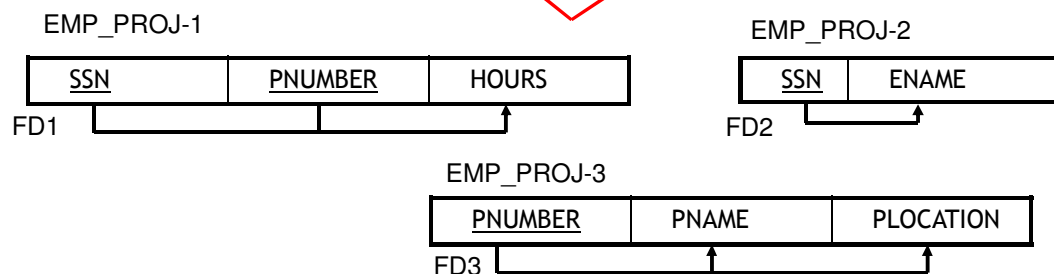
20

A relation schema R is in second normal form (2NF) if it is in 1NF and if every **non-prime attribute** A in R is **fully functionally dependent** on the primary key, where the primary key contains multiple attributes.

EMP_PROJ



2NF NORMALIZATION



3NF -Eliminate Columns Not Dependent On Key

21

A relation schema R is in third normal form (3NF) if it is in 2NF and **no** non-prime attribute A in R is **transitively dependent** on the primary key.

EMP_DEPT

ENAME	<u>SSN</u>	BDATE	ADDRESS	DNUMBER	DNAME	DMGRSSN
↑		↑	↑	↑	↑	↑

3NF NORMALIZATION

EMP_DEPT-1

ENAME	<u>SSN</u>	BDATE	ADDRESS	DNUMBER
↑	↑	↑	↑	↑

EMP_DEPT-2

<u>DNUMBER</u>	DNAME	DMGRSSN
↑	↑	↑

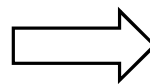
General Definitions of 2NF and 3NF

22

General Definition of 2NF: a relation schema R is in **2NF** if every nonprime attribute A in R is **not partially dependent** on any key of R .

General Definition of 3NF: a relation schema R is in **3NF** if whenever a nontrivial functional dependency $X \rightarrow A$ holds in R , either

- (a) X is a **superkey** of R , **OR**
- (b) A is a **prime** attribute of R .

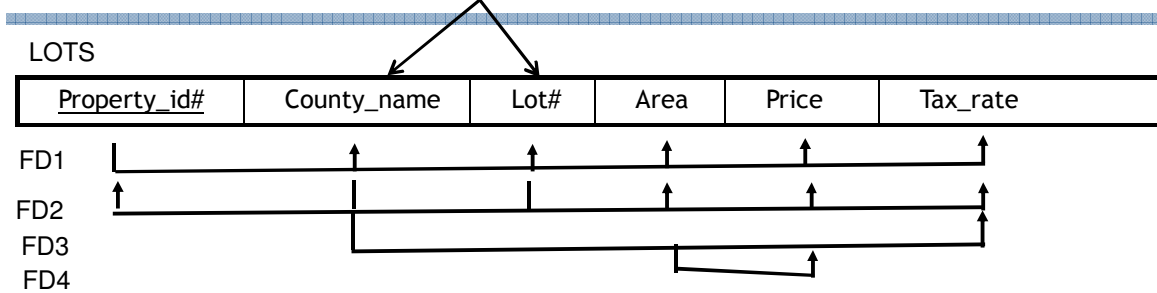


*Every nonprime attribute of R is Fully functional dependent on every key of R **and** it is non transitively dependent on every key of R .*

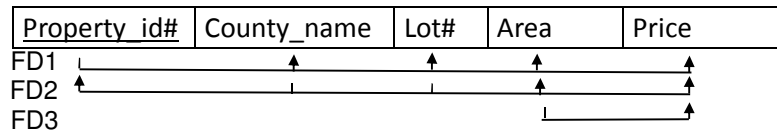
Example

Candidate key

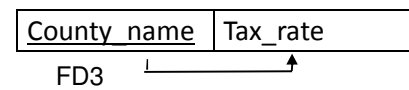
23



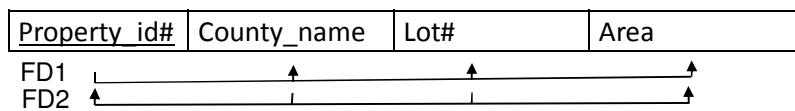
LOTS1



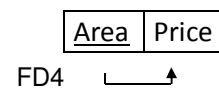
LOTS2



LOTS1A



LOTS1B



BCNF - Boyce-Codd Normal Form (1)

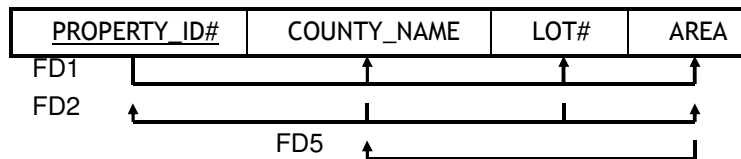
24

- A relation schema R is in Boyce-Codd Normal Form (BCNF) if it is in 3NF and if whenever an FD $X \rightarrow A$ holds in R , then X is a superkey of R .
- Typically, R is NOT in BCNF if:
 - There are multiple candidate keys and
 - The keys are composed of multiple attributes, and there are common attributes between the keys.
- In practice, most relation schemas that are in 3NF are also in BCNF, but not all!

3NF is a compromise, used when BCNF is not achievable (e.g., performance considerations).

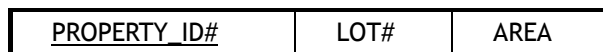
BCNF - Boyce-Codd Normal Form (2)

(a) LOTS1A



BCNF Normalisation

LOTS1AX

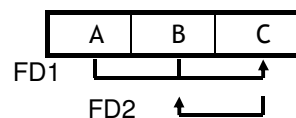


LOTS1AY



(b)

R



*C is not a superkey but B is a prime attribute.
 So, R is in 3NF, but not in BCNF.*

BCNF - Boyce-Codd Normal Form (3)

26

- In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints.
 - No dependency in R that can be predicted using FDs alone.
 - If we are shown two tuples that agree upon the X value, we cannot differ the A value in one tuple from the A value in the other.
 - If example relation is in BCNF, the 2 tuples must be identical (since X is a key).

X	Y	A
x	y1	a
x	y2	?

FD $X \rightarrow A$ would violate BCNF, if X is not a key, or if X does not contain A (as it is the case). Since X is a key (the relation is in BCNF), then in the 2nd tuple, the value of A is *a*, and *y1=y2*. However, since a relation is defined to be a set of tuples, we cannot have two copies of the same tuple, and so, this situation cannot arise.

Summary

27

- Normalization is the process of decomposing unsatisfactory relations by breaking up their attributes into smaller relations.
- A normal form is used to certify whether a relation schema is at a particular quality level in its design, by using the keys and functional dependencies (FDs) of the relation.
- Relations in 1NF disallow attributes whose values for an individual tuple are non-atomic. Since 1NF is a requirement implicit in the definition of the Relational Model, it must hold.
- The 2NF and 3NF still allow presence of redundancy which can be detected via FDs. However, BCNF ensures that no redundancy can be detected via FD information alone. And thus it is the most desirable normal form from the point of view of redundancy.

Advanced SQL

Fundamentals of Databases

Dr. Sandra Sampaio
School of Computer Science
University of Manchester

Acknowledgements

29

These slides are minor adaptations of material
authored and made available to instructors by

T. Connolly and C. Begg

to accompany their textbook

**Database Systems A Practical Approach to Design, Implementation, and
Management, 5th Edition**

Addison-Wesley, 2010, 978-0-321-52306-8

Copyright remains with them, whom I thank.

Any errors are my responsibility.

Outline

30

- How to use the SQL programming language
- How to use SQL cursors
- How to create stored procedures and functions
- How to create triggers
- The advantages and disadvantages of triggers

The SQL Programming Language

- Initial versions of SQL were **not computationally complete**, i.e., they had no programming constructs.
- More recent versions allow SQL to be embedded in high level programming languages, such as C.
- The main problem with this approach is **Impedance Mismatch**.
 - Mixing different programming paradigms.
 - SQL is a declarative language.
 - High-level languages such as C are procedural languages.
 - SQL handles rows of data at a time; high level programming languages, such as C, handle only one row of data at a time.
 - SQL and 3GLs use different models to represent data.
 - For example, SQL has Date and Interval; C does not have these.
 - 30% of programming effort goes into mapping data types from one data type system into the other.

PL/SQL

- Another approach is to extend SQL with programming language (procedural) constructs, which is known as SQL/PSM (Persistent Stored Modules).
- Oracle's procedural extensions to SQL is called **PL/SQL** (Procedural Language/SQL).
- PL/SQL is a limited programming language, compiled for speed.
- While SQL works on sets of rows, PL/SQL treats a table as a flat file accessed one row at a time.

PL/SQL

- PL/SQL has concepts similar to modern programming languages, such as variable and constant declarations, control structures, exception handling, and modularization.
- PL/SQL is a block-structured language: blocks can be entirely separate or nested within one another.
- A PL/SQL program can be built with the following basic units: procedures, function and anonymous (unnamed blocks).
- A PL/SQL block has up to three parts:
 - an optional declaration part, in which variables, constants, cursors, and exceptions are defined and possibly initialized;
 - a mandatory executable part, in which the variables are manipulated;
 - an optional exception part, to handle any exceptions raised during execution.

Anonymous Blocks

34

- Serve as scripts that execute PL/SQL statements (divided into 3 parts).

DECLARE

```
amount NUMBER := 0;  
v_error_code NUMBER;  
v_error_message VARCHAR2(255);
```

BEGIN

```
SELECT COUNT(*) INTO amount  
FROM Student  
WHERE Student.class = 3;  
DBMS-OUTPUT.PUTLINE(amount);
```

EXCEPTION

```
WHEN OTHERS THEN ROLLBACK;  
v_error_code := SQLCODE  
v_error_message := SQLERRM  
INSERT INTO t_errors VALUES ( v_error_code,  
v_error_message);  
END;
```

Please, refer to:
<http://www.ss64.com/oraplsql/exception.html> to find all 20 pre-defined exceptions that Oracle PL/SQL supports.

Declarations

- Variables and constant variables must be declared before they can be referenced.
- Possible to declare a variable as NOT NULL.
- Possible to use %TYPE to declare a variable to be of the same type as an attribute; and to use %ROWTYPE to declare a variable to be of the same type as an entire row. For example:
 - `vStaffNo Staff.staffNo%TYPE;`
 - `vStaffRec Staff%ROWTYPE;`

Assignments

- Variables can be assigned in two ways:
 - Using the normal assignment statement (:=);
 - As a result of an SQL SELECT or a FETCH statement.
- For example:
 - `vStaffNo := 'SG14';`
 - `SELECT COUNT (*) INTO x FROM PropertyForRent WHERE staffNo = vStaffNo;`
 - `FETCH cursor_1 into varRecord_1;`

Control Statements

- Conditional IF statement.
- Conditional CASE statement.
- Iteration statement (LOOP).
- Iteration statement (WHILE and REPEAT).
- Iteration statement (FOR).

Exceptions in PL/SQL

- An exception is an identifier in PL/SQL raised during the execution of a block that terminates its main body of actions.
- Exception handlers in PL/SQL are separate routines that handle raised exceptions.
- User-defined exceptions are defined in the declarative part of a PL/SQL block.

Cursors in PL/SQL

- A cursor has the following properties:
 - Allows the rows of a query result to be accessed one at a time.
 - Must be declared and opened before use.
 - Must be closed to deactivate it after it is no longer required.
 - Can be used to update rows.

Procedures - 1

40

```
CREATE OR REPLACE PROCEDURE testp4 (namein VARCHAR2, statein  
CHAR, deptin NUMBER) AS
```

```
    avg-salary NUMBER := 0; sumsal NUMBER;
```

```
    total-emps-less-Tom NUMBER; new-salary NUMBER;
```

```
CURSOR sal-cursor IS
```

```
    SELECT salary, name FROM Worker;
```

Continues in the next slide...

Procedures - 2

41

```
BEGIN
    sumsal := 0; total-emps-less-Tom := 0;
    FOR row IN sal-cursor LOOP
        IF row.name <> 'Tom' THEN
            sumsal := sumsal + row.salary;
            total-emps-less-Tom := total-emps-less-Tom + 1;
        END IF;
    END LOOP;
    new-salary := sumsal/totalempsless-Tom;
    INSERT INTO Worker VALUES (namein, statein, new-salary,
deptin);
END testp4;
```

Functions

42

- Are procedures which return a value.
- Can be used in SQL statements:

```
SELECT *  
FROM TABLE (flatTypes())  
WHERE branchNo = 'B003';
```

The only differences between a function and a procedure is a **return (result)** in the body of the function and its header, as in the following:

```
CREATE OR REPLACE FUNCTION fname (parameter  
list with types without lengths)  
RETURN datatype
```

Triggers

- A Trigger defines an action that the database should take when some event occurs in the application.
- A Trigger can be programmed to execute on the event of a table being modified.
 - It is 'fired' once for each row modified or once as a set of rows are modified.
- It can be 'fired' BEFORE or AFTER the event.

Trigger Format

```
CREATE TRIGGER TriggerName
  BEFORE | AFTER | INSTEAD OF
  INSERT | DELETE | UPDATE [OF TriggerColumnList]
  ON TableName
  [REFERENCING {OLD | NEW} AS {OldName | NewName}]
  [FOR EACH {ROW | STATEMENT}]
  [WHEN Condition]
  <trigger action>
```

Using a BEFORE Trigger

```
CREATE TRIGGER StaffNotHandlingTooMuch
BEFORE INSERT ON PropertyForRent
REFERENCING NEW AS newrow
FOR EACH ROW
DECLARE
    vpCount    NUMBER;
BEGIN
    SELECT COUNT(*) INTO vpCount
    FROM PropertyForRent
    WHERE staffNo = :newrow.staffNo;
    IF vpCount = 100
        raise_application_error(-20000, ('Member' || :newrow.staffNo || 'already managing 100 properties'));
    END IF;
END;
```

Write this trigger using the Oracle's syntax for Triggers.

Advantages and Disadvantages of Triggers

46

- Elimination of redundant code.
- Simplification of modifications.
- Increased security.
- Improved integrity.
- Improved processing power.
- Good fit with the client-server architecture.
- Performance overhead.
- Cascading effects.
- Cannot be scheduled.
- Less portable.

Trigger Structure

- Oracle trigger syntax:

```
create or replace trigger name  
event  
[when condition]  
[for each row]  
action
```

- In Oracle:

- The *condition* is a boolean expression (that does not access the database).
- The *action* is a PL/SQL block.

Oracle Triggers

- Transition granularity:
 - Row (tuple) triggers - FOR EACH ROW.
 - statement (set) triggers – no FOR EACH ROW.
- Coupling mode:
 - immediate.
- Priorities:
 - unspecified.
- Event types:
 - primitive (DML, DDL and system (e.g. startup/shutdown)).
 - composite (but only OR).

DML Events



■ Follow database updates:

- `[BEFORE|AFTER] INSERT ON table.`
- `[BEFORE|AFTER] DELETE ON table.`
- `[BEFORE|AFTER] UPDATE OF table.`
- `[BEFORE|AFTER] UPDATE OF column on table.`

■ Plus disjunction, e.g.:

- `BEFORE INSERT OR UPDATE OF visit.`

Condition

- Row triggers can have conditions that guard the action.
- The condition is a boolean expression (**AND, OR, NOT, >, <, ...**).

- The condition refers to literals and to event properties through correlation variables, e.g.:
 - WHEN **new**.age < 21.

- Correlation variables available depend on event types:

Event	new	old
INSERT	Y	N
DELETE	N	Y
UPDATE	Y	Y

Action

- An action is a PL/SQL block.
- An action:
 - can refer to correlation variables, as `:new`, `:old` (tuple (row) triggers only).
 - can test the type of event being reacted to using `inserting`, `updating`, `deleting`.
 - cannot use transaction control commands directly (but can raise exceptions).

Oracle Trigger Example

```
CREATE TRIGGER upd_check BEFORE UPDATE ON
  account
FOR EACH ROW
BEGIN
  IF NEW.amount < 0 THEN
    SET NEW.amount = 0;
  ELSEIF NEW.amount > 100 THEN
    SET NEW.amount = 100;
  END IF;
END;
```

Summary

53

- Initial versions of SQL were not computationally complete.
- To overcome this problem, newer versions of SQL allow it to be embedded within programming languages, such as C. However, this approach incurs impedance mismatch.
- Another approach is to extend SQL with procedural language features (PL/SQL)
- Procedures, functions and the action part of Triggers can be implemented using PL/SQL.
- Triggers implement reactive behaviour in a DB, and have a number of advantages and disadvantages.

Transactions, Concurrency Control and Recovery

Fundamentals of Databases

Dr. Sandra Sampaio
School of Computer Science
University of Manchester

Acknowledgements

55

These slides are minor adaptations of material
authored and made available to instructors by

M. Kifer, A. Bernstein, P.M. Lewis

to accompany their textbook

Database Systems
An Application-Oriented Approach,
2nd Edition
Pearson Addison-Wesley, 2005

Copyright remains with them, whom I thank.

Any errors are my responsibility.

Transactions

56

- A **transaction** is an action, or a series of actions, carried out by user or application, which reads or updates contents of a database.
- A transaction transforms a database from one consistent state to another. It can have one of the following two outcomes:
 - **Success** – when the transaction **commits** and the database reaches a new consistent state.
 - **Failure** – when the transaction aborts and the database must be restored to the consistent state of before the transaction started. Such a transaction is **rolled back** or **undone**.
- A committed transaction cannot be aborted.
- An aborted transaction that is rolled back can be restarted later.

Properties of Transactions

57

- Four basic (ACID) properties of a transaction are:
 - **Atomicity:** A transaction should either complete or have no effect at all. It is a responsibility of the transaction processing system.
 - **Consistency:** A transaction should correctly transform the database state to reflect the effect of a real world event. It is a responsibility of the transaction designer.
 - **Isolation:** The effect of concurrently executing a set of transactions is the same as if they had executed serially (serializable). It is a responsibility of the transaction processing system.
 - **Durability:** The effect of a transaction on the database state should not be lost once the transaction has committed. It is a responsibility of the transaction processing system.

Isolation and Performance Expectations

58

- Executing a set of (consistent) transactions serially (one at a time) is correct, but the performance obtained from this execution might be inadequate.
- Executing a set of (consistent) transactions concurrently (interleaved) offers performance benefits, but might not be correct (e.g., might leave the database in an inconsistent state).
- The next slide shows an example of a potential problem caused by the use of concurrency in the execution of transactions: **The lost update problem**. Other problems of the same nature are illustrated on the two last slides of this presentation, however, due to time constraints, they may not be discussed in class. These are the uncommitted dependency problem and the inconsistent analysis problem.

Lost Update Problem

59

- In this case, a successfully completed update (T2) is overridden by another user (T1).
- Loss of T2's update can be avoided by preventing T1 from reading bal_x until after update by T2.

Time	T ₁	T ₂	bal_x
t ₁		begin_transaction	100
t ₂	begin_transaction	read(bal_x)	100
t ₃	read(bal_x)	$bal_x = bal_x + 100$	100
t ₄	$bal_x = bal_x - 10$	write(bal_x)	200
t ₅	write(bal_x)	commit	90
t ₆	commit		90

Serializable and Non-Serializable Schedules

60

- The schedule for the execution of transactions is **serializable** in the following cases:
 - If the operations of distinct transactions are on different data items, so these operations can commute.
 - If the operations of distinct transactions are **read** operations on the same data item, so these operations can commute.
- The schedule for the execution of transactions is **non-serializable** in the following case:
 - If the operations of distinct transactions are **read** and **write** operations (or **two write** operations) on the same data item, so that these operations cannot commute.

Concurrency Control Protocols

61

- The objective of a concurrency control protocol is to schedule transactions in such a way as to avoid any interference. This can be achieved via *Serializability*.
- The idea behind the term Serializability is to find **non-serial schedules** that allow transactions to execute concurrently without interfering with one another, and thereby produce a DB state that could be produced by a serial execution.
- In serializability, ordering of read/writes is important:
 - If two transactions only read a data item, they do not conflict and order is not important.
 - If two transactions either read or write completely separate data items, they do not conflict and order is not important.
 - If one transaction writes a data item and another reads or writes same data item, order of execution is important.

Implementing Serializability: The Two Phase Locking Protocol (2PL)

62

- To achieve proper ordering of operations from distinct transactions, **locking** of the data items accessed by the transactions can be done.
- In fact, locking is the most widely used approach to ensure serializability.
 - Generally, a transaction must claim a shared (read) or exclusive (write) lock on a data item before reading or writing that data item.
 - Lock prevents another transaction from modifying an item or even reading it, in the case of a write lock.
- A transaction follows the **Two Phase Locking (2PL) Protocol** if all locking operations precede the first unlock operation in the transaction and release of all locks are left until the end of the transaction.

Lost Update Problem using 2PL

63

Time	T ₁	T ₂	bal _x
t ₁		begin_transaction	100
t ₂	begin_transaction	write_lock(bal_x)	100
t ₃	write_lock(bal_x)	read(bal_x)	100
t ₄	WAIT	bal_x = bal_x + 100	100
t ₅	WAIT	write(bal_x)	200
t ₆	WAIT	commit/unlock(bal_x)	200
t ₇	read(bal_x)		200
t ₈	bal_x = bal_x - 10		200
t ₉	write(bal_x)		190
t ₁₀	commit/unlock(bal_x)		190

The Problem of Deadlock

64

- To make sure that a transaction's rollback following an update will not generate inconsistencies for other transactions, with 2PL, release of all locks are left until end of the transaction.
- An impasse may result when two (or more) transactions are each waiting for locks held by the other to be released. This impasse is called **Deadlock**.
- In case of deadlock:
 - A transaction in the cycle must be aborted by DBMS (since transactions will wait forever).
 - DBMS uses deadlock detection algorithms or timeout to deal with this.

Atomicity and Durability in Case of Failure

65

- Atomicity must be enforced even in the event of failure. For example:
 - When a user aborts a transaction (e.g., cancel button);
 - When the system aborts a transaction (e.g., deadlock);
 - When a transaction aborts itself (e.g., unexpected db state);
 - When the system crashes, etc.
- Durability must also be enforced in the event of failure. For example:
 - Media failure.
- The process of recovering the database back to a consistent state in the event of a failure enforces both properties.
- A mechanism that aids in **recovering** from failures is the **log**.

The Log - 1

66

- The log is an append-only sequence of records used to restore a database to a consistent state following a failure. It contains begin, update, abort and commit records from a number of transactions running at a particular time.
- It is stored on non-volatile storage device, distinct from the mass storage device that contains database. Therefore, it survives processor crash and media failure.

The Log - 2

67

- When a transaction performs an update: a description of the update, containing a before image of the updated item, is appended to the log.
- When a transaction aborts: the log is scanned backwards; the before image of updated items are restored; the scan is terminated when a *Begin Record* is found.
- A transaction commits only after its *Commit Record* has been added to the log.
- When the system crashes : the log is scanned backwards; if the first record encountered for T is an update record, T was active at time of crash and must be rolled back.

Recovery (transaction-based)

68

- It is the process of restoring a database to a correct state in the event of a failure.
- If a failure occurs between the commit time of a transaction and the database buffers being flushed to secondary storage, then, to ensure durability, the Recovery Manager has to redo (**rollforward**) transaction's updates, which have been recorded in the log.

Summary

69

- A transaction is an action, or a series of actions, carried out by user or application, which reads or updates contents of database.
- Transactions have the ACID properties.
- While the property of Isolation needs to be enforced for correctness, performance issues make the use of concurrency in the execution of transactions mandatory.
- Concurrency control protocols apply Serializability to find non-serial schedules that allow transactions to execute concurrently without interfering with one another, thereby producing a DB state that could be produced by a serial execution.
- For recovering from failures, the log file is used to restore the database to a previous consistent state before the failure.

Uncommitted Dependency Problem

70

- Occurs when a transaction can see intermediate results of another transaction before it has committed.
- This problem can be avoided by preventing T3 from reading bal_x until after T4 commits or aborts.

Time	T ₃	T ₄	bal_x
t ₁		begin_transaction	100
t ₂		read(bal_x)	100
t ₃		$bal_x = bal_x + 100$	100
t ₄	begin_transaction	write(bal_x)	200
t ₅	read(bal_x)	:	200
t ₆	$bal_x = bal_x - 10$	rollback	100
t ₇	write(bal_x)		190
t ₈	commit		190

Inconsistent Analysis Problem

71

- Occurs when a first transaction reads several values but a second transaction updates some of them during the execution of the first.
- This problem can be avoided by preventing T6 from reading bal_x and bal_z until after T5 has completed its updates.

Time	T ₅	T ₆	bal _x	bal _y	bal _z	sum
t ₁		begin_transaction	100	50	25	
t ₂	begin_transaction	sum = 0	100	50	25	0
t ₃	read(bal _x)	read(bal _x)	100	50	25	0
t ₄	bal _x = bal _x - 10	sum = sum + bal _x	100	50	25	100
t ₅	write(bal _x)	read(bal _y)	90	50	25	100
t ₆	read(bal _z)	sum = sum + bal _y	90	50	25	150
t ₇	bal _z = bal _z + 10		90	50	25	150
t ₈	write(bal _z)		90	50	35	150
t ₉	commit	read(bal _z)	90	50	35	150
t ₁₀		sum = sum + bal _z	90	50	35	185
t ₁₁		commit	90	50	35	185

File Organisation and Indexes

Fundamentals of Databases

Dr. Sandra Sampaio
School of Computer Science
University of Manchester

Acknowledgements

73

These slides are minor adaptations of material
authored and made available to instructors by

T. Connolly, C. Begg

to accompany their textbook

Database Systems

A Practical Approach to Design, Implementation, and Management

4th Edition

Pearson Addison-Wesley

Copyright remains with them, whom I thank.

Any errors are my responsibility.

Unordered Files (Heap / Pile)

74

- New records are inserted at the end of the file.
- Record **insertion** is quite **efficient**.
- **Reading** requires reading and searching half the file records on average, and is hence quite **expensive**. Therefore, reading in order of a particular field **requires sorting** the file records.

Ordered Files (Sequential)

75

- File records are kept sorted by the values of an ordering field.
 - **Insertion is expensive**: records must be inserted in the correct order. A separate **unordered overflow file** for new records may be used to improve efficiency, and is periodically merged with the main ordered file.
 - A **binary search** is used to search. It represents an **improvement over linear search**.
 - **Reading** the records **in order** of the ordering field is quite **efficient**.
-

Hashed Files (External Hashing)

76

- The file records are divided into **M equal-sized buckets**.
- One of the file fields is designated to be the **hash key** of the file.
- The record with hash key value K is stored in bucket i , where $i=h(K)$, and h is the **hash function**.
- **Search** is very **efficient** on the **hash key**.
- **Collisions** occur when a new record hashes to a bucket that is already full.

Hashed Files (Collision Resolution)

77

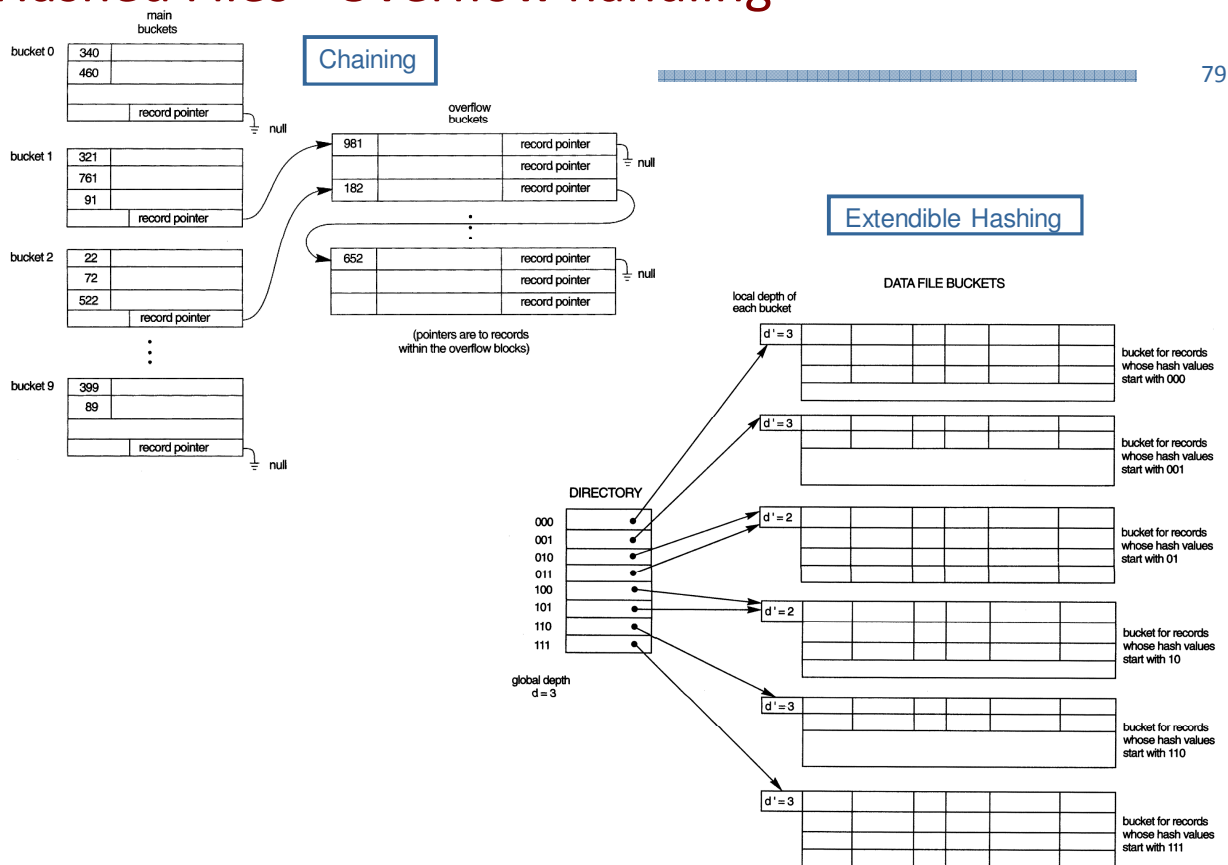
- **Open addressing:** Proceeding from the occupied position specified by the hash address, checks for subsequent empty positions.
- **Chaining:** Overflow locations are kept by extending the bucket with a number of overflow positions. A pointer field is added to each record location. A collision is resolved by placing the new record in an unused overflow location and setting the pointer of the occupied hash address location to the address of that overflow location.
- **Multiple Hashing:** The program applies a second hash function if the first results in a collision. If another collision results, the program uses open addressing or applies a third hash function and then uses open addressing if necessary.

Hashed Files (cont.)

78

- **Ordered access** on the hash key is quite **inefficient** (requires sorting the records).
- The hash function should distribute the records uniformly among the buckets.
- Main **disadvantage** of static hashing:
 - **Fixed** number of **buckets**, M , is a problem if the number of records in the file grows or shrinks (files can grow or shrink over time).
- Techniques are adapted to allow the dynamic growth / shrinking.
 - **Dynamic hashing**: there is a directory which is a binary tree.
 - **Extendible hashing**: there is a directory which is an array of size 2^d , where d is called the global depth.

Hashed Files - Overflow handling



Indexes as Access Paths

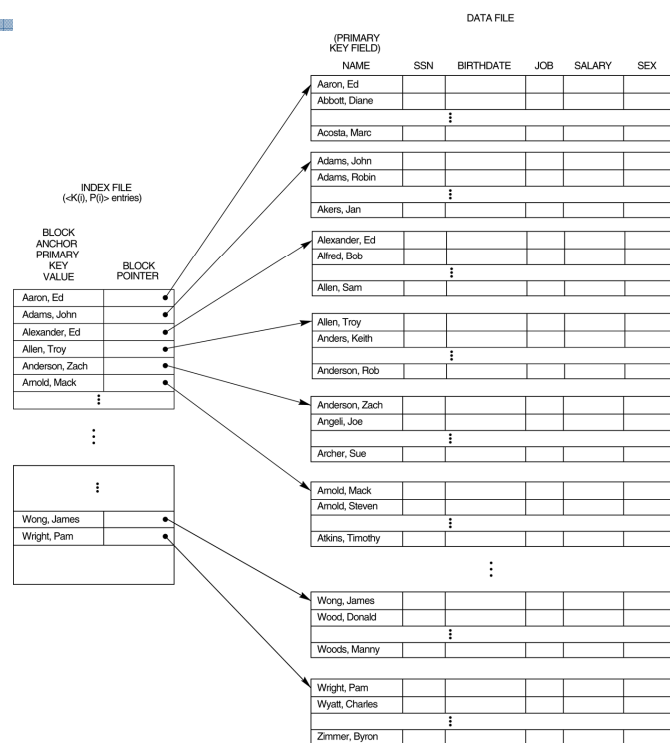
80

- A single-level index is an **auxiliary file** that makes it more efficient to search for a record in the data file.
 - The index is usually specified on **one field of the file**.
 - One form of an index is a **file of entries** <field value, pointer to record>, which is **ordered by field value**.
 - The index is called an access path on the field, characterized as follows.
 - A **dense** index has an index entry for every search key value (and hence every record) in the data file.
 - A **sparse** (or nondense) index, on the other hand, has index entries for only some of the search values
-

Types of Single-Level Indexes(1)

■ Primary Index

- Defined on an ordered data file
- The data file is **ordered** on a **key field**
- A primary index is a non-dense (**sparse**) index, since it includes an entry for each disk block of the data file and the key of its anchor record rather than for every search value.

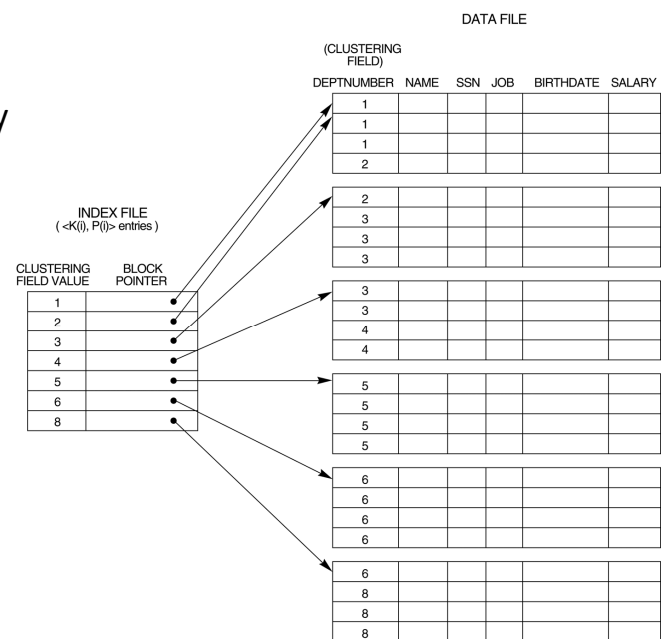


Types of Single-Level Indexes(2)

82

■ Clustering Index

- The data file is **ordered** on a **non-key field** (also called *clustering field*), unlike primary index (which requires that the ordering field of the data file have a distinct value for each record).
- Includes one index entry for each distinct value of the field; the index entry points to the first data block that contains records with that field value.
- It is another example of **non-dense (sparse) index**.



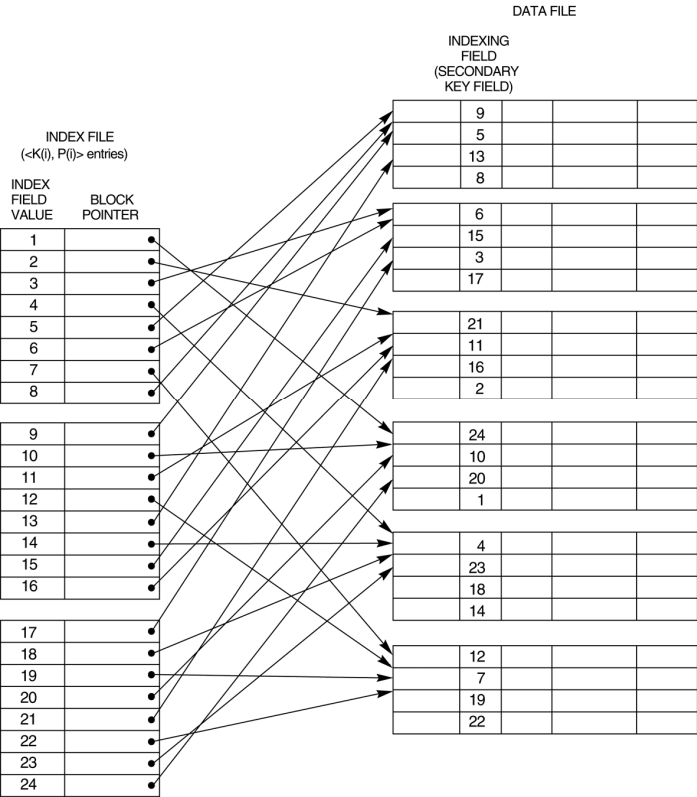
Types of Single-Level Indexes(3)

83

■ Secondary Index

- A secondary index provides a **secondary means of accessing a file** for which some primary access already exists.
- The secondary index may be on a field which is a **candidate key** and has a unique value in every record, **or a non-key** with duplicate values.
- The index is an ordered file with two fields.
 - The first field is of the same data type as some **non-ordering field** of the data file that **is an indexing field**.
 - The second field is either a **block pointer** or a **record pointer**. **There can be many secondary indexes (and hence, indexing fields) for the same file.**
- Includes one entry for each record in the data file; hence, it is a **dense index**.

A Dense Secondary Index



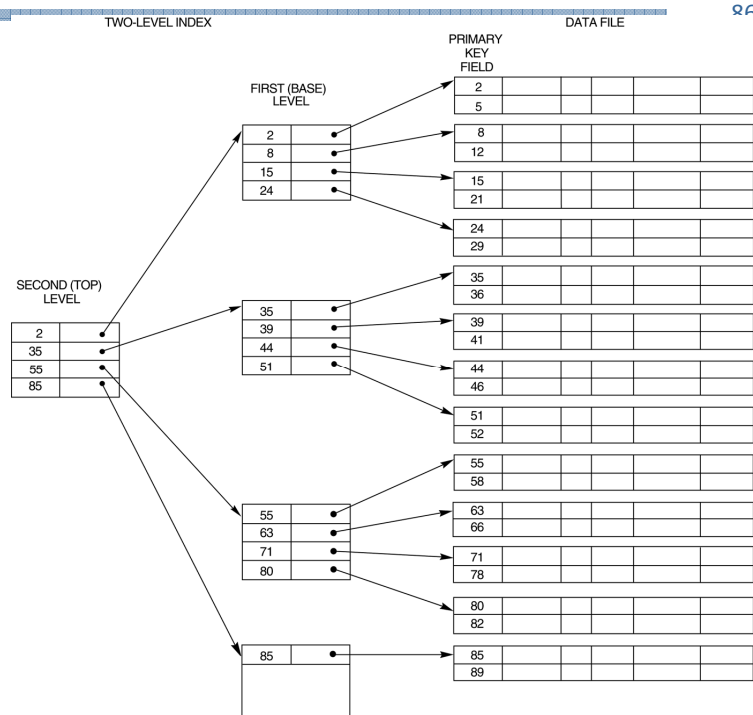
Multi-Level Indexes

85

- Because a single-level index is an ordered file, we can create a [primary index to the index itself](#); in this case, the original index file is called the first-level index and the index to the index is called the second-level index.
 - We can repeat the process, creating a third, fourth, ..., top level until all entries of the top level fit in one disk block.
 - A multi-level index [can be created for any type of first-level index](#) (primary, secondary, clustering) as long as the first-level index consists of more than one disk block.
-

A Two-Level Primary Index

- Such a multi-level index is a form of search tree; however, insertion and deletion of new index entries is a severe problem because every level of the index is an ordered file.
- One Method - Dynamic Multilevel Indexes using B-Trees and B+-Tree.



Summary

87

- There are 3 main types of file organisation:
 - Heap or unordered files: records are placed on disk in no particular order.
 - Sequential or ordered files: records are ordered by the value of a specified field.
 - Hash files: records are placed on disk according to a hash function.
- Indexes represent a technique for making the retrieval of data more efficient/faster.
- Indexes can be sparse or dense, can have one or more levels, and can be primary secondary or clustering, etc.
- File organisation as well as indexes have a significant impact on the performance with which data is retrieved from and written to disk.