Implementing A Database

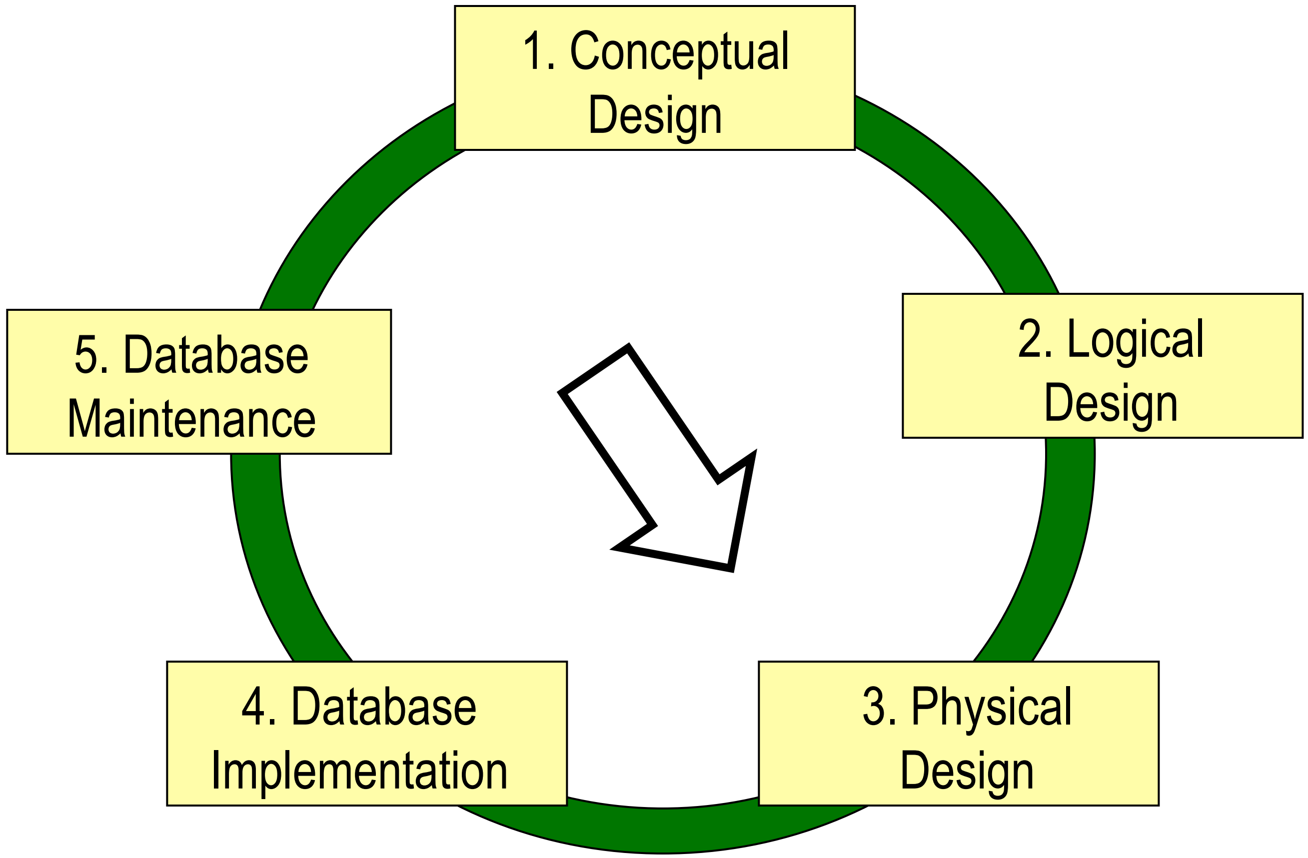
Last Class

- Concepts
- Converting relationships
  - Binary
  - Unary
  - Ternary
- Normalization
- Exercises

Outline

- Data integrity
- Constraining relationships
- Factors affecting DB performance
  - Indexing
  - Views
  - Partitioning
  - Denormalization
- Data Types

Database Life Cycle



Physical Design

- Purpose:
  - Design structure of database and identify technical specifications to optimize database performance
    - Using relations as input

- Output:
  - An implementable database design

Goal

- Create an implementable database design

Project (ID, Name, … Budget)

Project_Employee (ID, SSN, …, Hours_worked)

Employee (SSN, First_name, …, Phone)

Certificate (ID, Name, …, Valid_until)

Department (Number, Name, Floor)

Tables

Project (ID int NOT NULL, Name VARCHAR…)

Project_Employee (ID int NOT NULL, SSN…)

Indexes

Index Project(ID)…

Index Project(Name)…

Views

Employee_List(First_name, Last_Name, Phone)

…

Physical Database Design

- Process of designing the <u>structure</u> of a database and identifying technical specifications to optimize database performance at runtime

- Goals:
  - Ensure data integrity
  - Acceptable performance (speed)
    - Optimize database size
    - Reduce time needed to locate records of interest
    - Minimize joins
    - Consider denormalizing some of the 3NF relations

Ensuring Data Integrity

- Four requirements:
  - Domain integrity – values entered into columns are valid
  - Entity integrity – each row is uniquely identified
  - Referential integrity – references to other tables remain valid
  - Policy integrity – values adhere to business rules

Domain Integrity

- Specify the appropriate data type for each column
- Other controls:
  - Nullability – can a column contain null values?
  - Check constraint – do the column values belong to a set list, range, etc.?
  - Unique constraint – is the column value unique?
  - Default constraint – is there a default value if nothing is entered?

Entity Integrity

- Each entity has a primary key

- Create an artificial primary key
  - No natural primary key
  - Complex composite primary key

  - Artificial key = auto generated number / identity column

Referential Integrity

- The value of a foreign key is "constrained" to the values of a primary key in a different table

EMPLOYEE

Emp_ID	First_Name	Last_Name	Salary
100	Margaret	Simpson	48,000
140	Allen	Beeton	52,000
110	Chris	Lucero	43,000
190	Lorenzo	Davis	55,000
150	Susan	Martin	42,000

COURSE

Course_Num	Course_Name	Emp_ID
4141	Java	190
6217	DB Admin	140
6136	Data Mining	140
6480	eCommerce	110
6321	**ERP**	**999**

Policy Integrity

- Enforce business rules
  - May be enforceable through domain, entity, and/or referential integrity
    - E.g., Each course must have an assigned employee
      - Referential integrity & nullability
    - E.g., An employee may have 0 – 160 hours of vacation leave
      - Check constraint
  - More complicated rules – use triggers or application
    - E.g., Audit trails

Triggers

- Snippets of programs that run (or fire) in response to an event
  - Insert, update, delete data
- More resource intensive than domain, entity, or referential integrity constraints
  - Too many triggers can slow down performance

Integrity Constraints

DATE | TIME | INT | INT | CHAR(2) | CHAR(4)

Domain Integrity

Referential Integrity

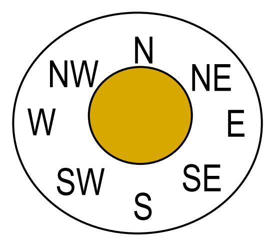
WEATHER

Read_Date	Read_Time	Temp	Wind_Spd	Wind_Dir	Station
2/10/2011	6:00 AM	56	9	SE	S100
2/10/2011	9:00 AM	62	7	S	S100
2/10/2011	12:00 AM	71	4	SE	S101
2/10/2011	3:00 PM	71	6	E	S101
2/10/2011	6:00 PM	66	9	E	S101

Not Null Not Null

Null Constraints

>0



Policy Integrity

Constraining Relationship (FK)

- Specify what should happen to the many side of a 1:M relationship

- Updates and Deletes
  - Restrict
  - Cascade
  - Set-to-null
  - Default

- Can mix and match
  - E.g., Update cascade, Delete restrict

On Delete (Update) – Restrict

- If we try to delete a record on the "one side" of a 1:M relationship, the database will reject the delete if there are any matching FK values on the "many side"

EMPLOYEE

Emp_ID	First_Name	Last_Name	Salary
100	Margaret	Simpson	48,000
~~140~~ 199	~~Allen~~	~~Beeton~~	~~52,000~~
110	Chris	Lucero	43,000
190	Lorenzo	Davis	55,000
150	Susan	Martin	42,000

COURSE

Course_Num	Course_Name	Emp_ID
4141	Java	190
6217	DB Admin	140
6136	Data Mining	140
6480	eCommerce	110

On Delete (Update) – Cascade

- If we try to delete a record on the "one side" of the 1:M relationship, both that record and all matching records on the "many side" value will be deleted

EMPLOYEE

Emp_ID	First_Name	Last_Name	Salary
100	Margaret	Simpson	48,000
~~140~~ 199	~~Allen~~	~~Beeton~~	~~52,000~~
110	Chris	Lucero	43,000
190	Lorenzo	Davis	55,000
150	Susan	Martin	42,000

COURSE

Course_Num	Course_Name	Emp_ID
4141	Java	190
~~6217~~	~~DB Admin~~	~~140~~ 199
~~6136~~	~~Data Mining~~	~~140~~ 199
6480	eCommerce	110

On Delete (Update) – Set-to-Null

- If a record on the "one side" of the 1:M relationship is deleted, that record will be deleted and the matching FK on the "many side" will be changed to null

EMPLOYEE

Emp_ID	First_Name	Last_Name	Salary
100	Margaret	Simpson	48,000
~~140~~ 199	~~Allen~~	~~Beeton~~	~~52,000~~
110	Chris	Lucero	43,000
190	Lorenzo	Davis	55,000
150	Susan	Martin	42,000

COURSE

Course_Num	Course_Name	Emp_ID
4141	Java	190
6217	DB Admin	~~140~~ NULL
6136	Data Mining	~~140~~ NULL
6480	eCommerce	110

On Delete (Update) – Default

- If a record on the "one side" of the 1:M relationship is deleted, that record will be deleted and the matching FK on the "many side" will be changed to a predefined default value

EMPLOYEE

Emp_ID	First_Name	Last_Name	Salary
100	Margaret	Simpson	48,000
~~140~~ **199**	~~Allen~~	~~Beeton~~	~~52,000~~
110	Chris	Lucero	43,000
190	Lorenzo	Davis	55,000
150	Susan	Martin	42,000

COURSE

Course_Num	Course_Name	Emp_ID
4141	Java	190
6217	DB Admin	~~140~~ **100**
6136	Data Mining	~~140~~ **100**
6480	eCommerce	110

Factors Affecting DB Performance

- Data-related factors:
  - Large data volumes: How many records, how large is each record
- Database structure factors:
  - Searching on attributes without direct access
  - Composite keys
- Data storage factors:
  - Data dispersed all over the disk (multiple tables): Slows data access
- Application Factors:
  - Need for joins: Consumes substantial time and resources
  - Need to calculate totals: Time-consuming for large tables
- Business environment factors:
  - Too many data access operations: How frequently are tables accessed
  - Overly liberal data access: Who access these tables, for what purpose (security)

Physical Design Techniques

Don't Change Logical Design

- Adding external features:
  - Adding indexes
  - Adding views
- Splitting one table into multiple files:
  - Horizontal partitioning
  - Vertical partitioning
  - Splitting large text attributes

Change Logical Design

- Changing table attributes:
  - Creating new primary keys
  - Storing derived data
- Combining tables:
  - Combine tables in one-to-one relationships
  - Alternative for repeating groups
  - Denormalization

External Feature: Indexes

- Indexes are mechanisms for providing direct access to specific fields (or combination of fields) in a table
  - Improves data retrieval speed
  - Useful for search and join operations
- Which field(s) to index:
  - Primary keys
  - Foreign keys used for joins
  - Search attributes
  - Attributes used for grouping

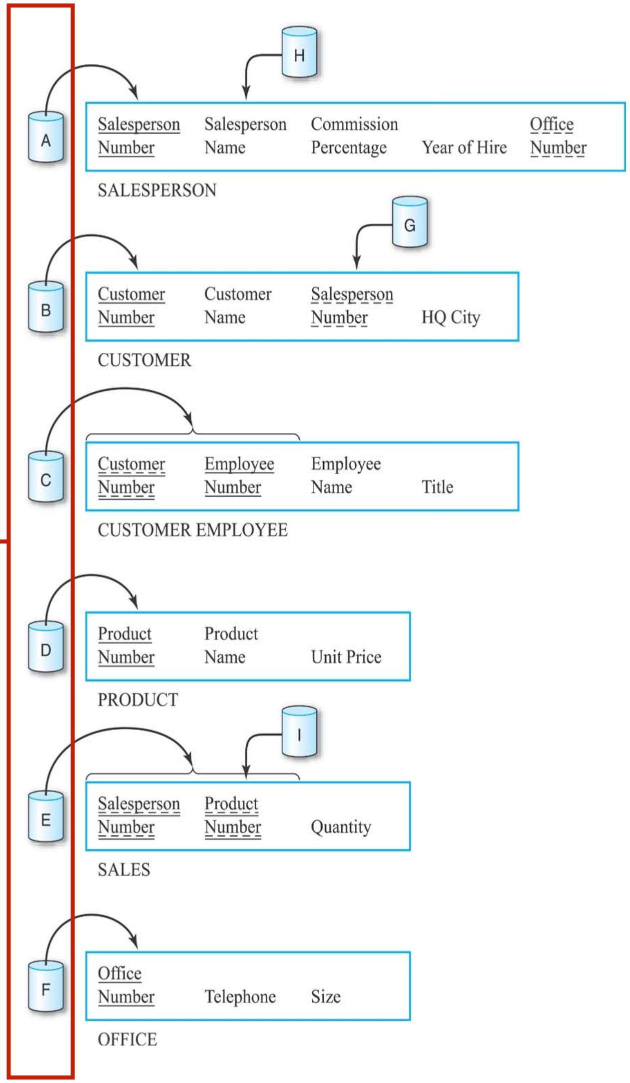
- Having many indexes may take more space than actual data
- Too many indexes may slow down performance: each update causes DBMS to update related indexes
- Small tables may not need indexes (read entire table into memory)

Working without Indexes

- Querying a table without an index:
  - Perform a table scan
    - Read each record into memory
    - Find records meeting the condition
    - Return records containing required data
- Adding/updating/deleting a record:
  - Add/update/delete the record
- Tradeoffs:
  - Slower queries, increase disk accesses
    - Except for very small tables
  - Faster maintenance: only need to add / delete / update records

Indexes

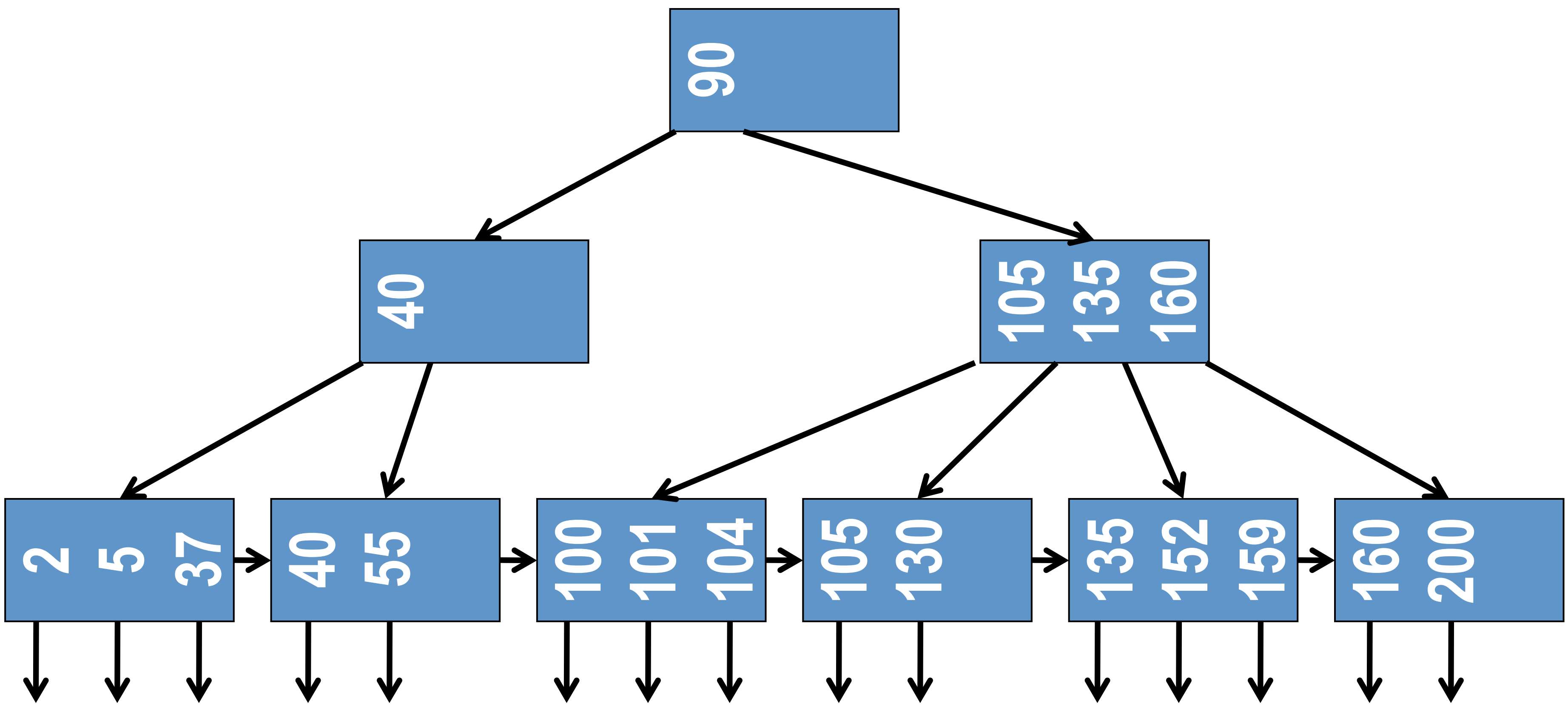
Indexes created automatically by the DBMS



Working with Indexes

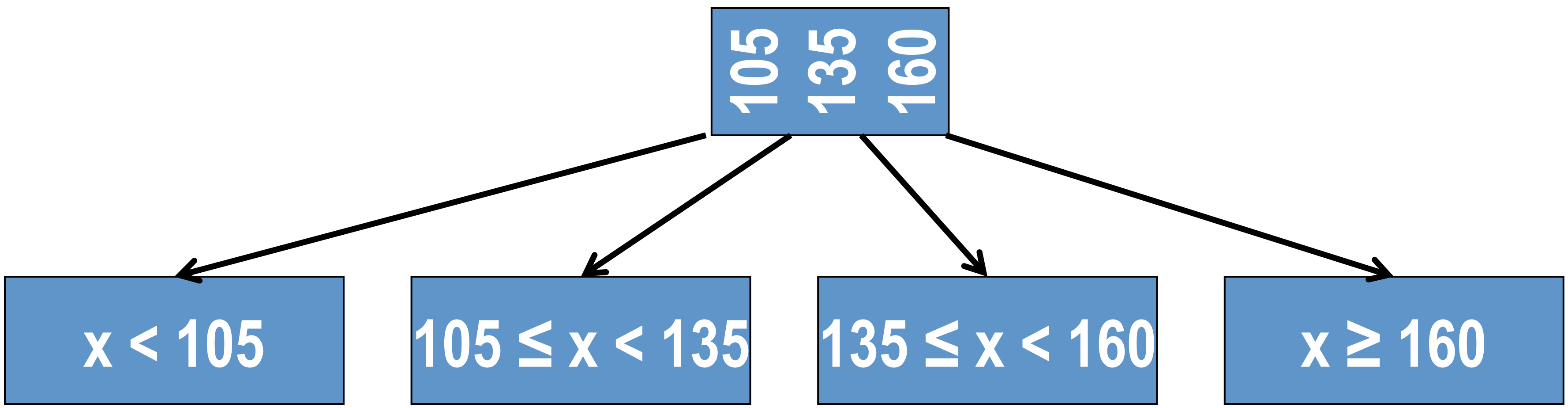
- Querying a table with an index:
  - Read index into memory
  - Search index to find records meeting the condition
  - Access only those records containing required data
- Adding/updating/deleting a record:
  - Add/update/delete the record
  - Update the index
- Tradeoffs:
  - Faster queries, since disk accesses are reduced
  - Slower maintenance: Require at least two accesses for adding/deleting/ updating records
  - Static databases benefit more overall

B+-Tree Index



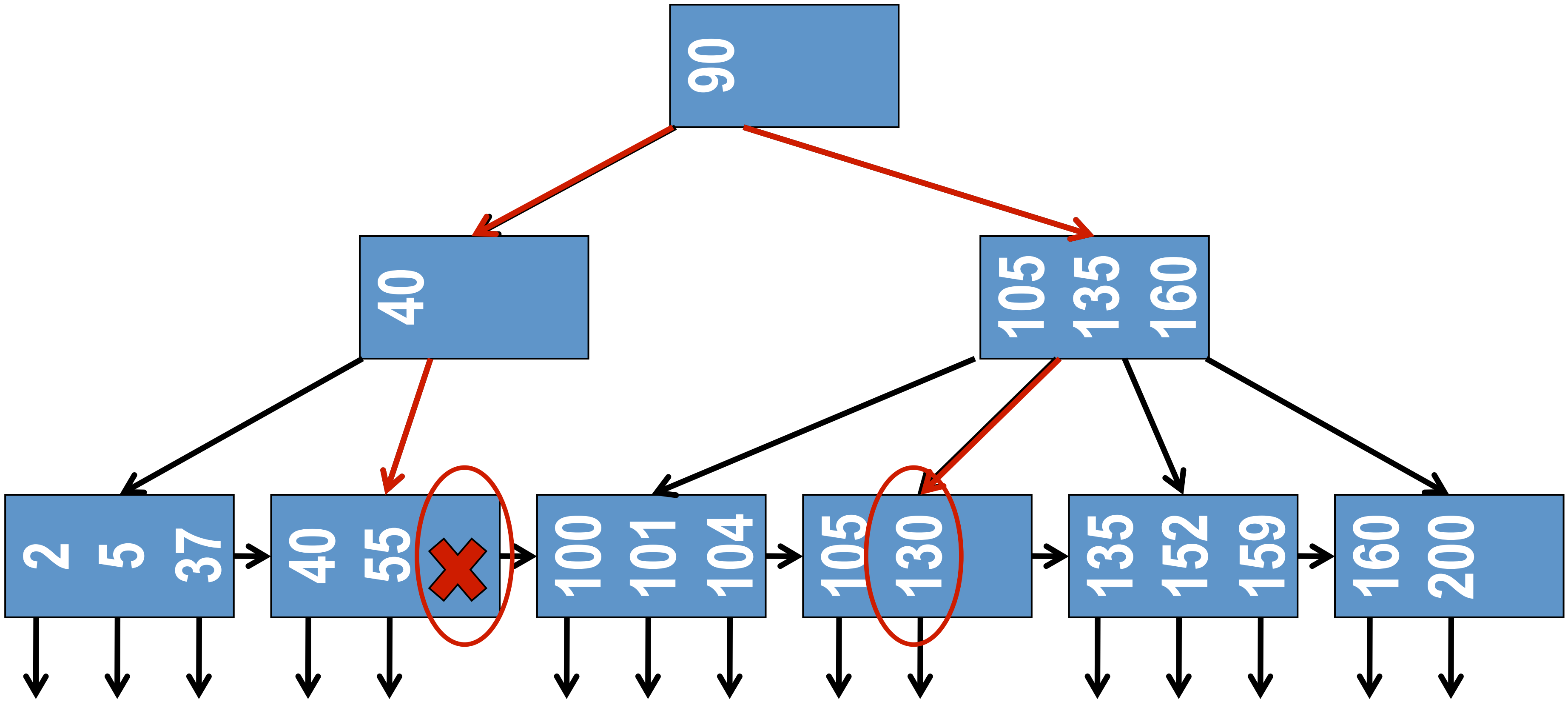
90
40
105 135 160
2 5 37
40 55
100 101 104
105 130
135 152 159
160 200

B+-Tree Index



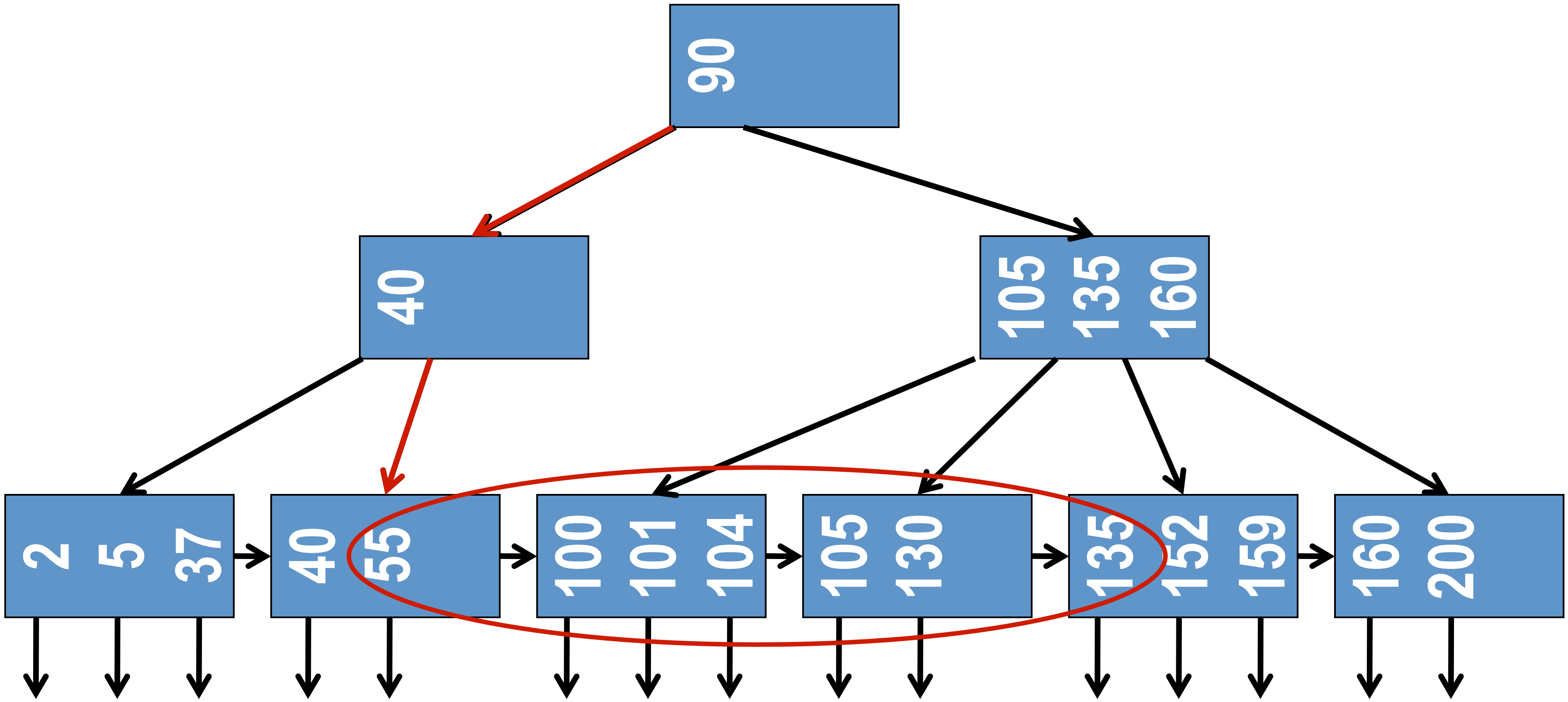
B+-Tree Index

- Lookup: x = 130; x = 57



B+-Tree Index

- Range: $41 \leq x \leq 137$



External Feature: Views

- Views are logical combinations of a subset of records and attributes in one or more tables
- Often implemented via table joins
- No data is physically duplicated – hence no redundancy
- Can protect data security and privacy, by restricting the data provided to users

CUSTOMER (<u>CustID</u>, Name, Address, City, State, Zip, Phone)
SALE (<u>TransID</u>, CustID, StoreID, Date, Amount)

View:
CUST_SALE (CustID, Name, City, Zip, TransID, Date, Amount)

Splitting Tables: Horizontal Partitioning

- Records of a table are split into two or more separate files
  - Each partition has the same fields but different records
  - Records that are used together should stay together
- Why split:
  - Works well if applications do not need to access all records at the same time
  - Easier to retrieve data from smaller tables
  - Improves performance

Splitting Tables: Horizontal Partitioning

Northeastern Customers

Cust_ID	Name	City
112	Alice Smith	New York
221	Mike Jordan	Albany

Western Customers

Cust_ID	Name	City
411	John Doe	Los Angeles
522	Sue Jones	Stanford

CUSTOMER table partitioned based on customer's location

Splitting Tables: Vertical Partitioning

- Fields in a table are distributed across two or more files/ partitions
  - Each partition have different fields but must store the primary key
  - Fields that are used together should stay together
- Works well if different fields are needed for different applications
- Splitting large text attributes:
  - Splitting large text attributes into a separate vertical partition with its own copy of the primary key

Splitting Tables: Vertical Partitioning

Marketing Department

Cust_ID	Name	City	Phone
112	Alice Smith	New York	(212) 223-2222
221	Mike Jordan	Albany	(721) 243-3564

Billing Department

Cust_ID	Amt_Due	Date_Due
112	$1,332.90	2/5/2011
221	$223.90	1/31/2011

CUSTOMER table partitioned based on departmental relevance

Adding Attributes: Creating New PK

- Changes the logical design
- Replace composite primary key with a new single-attribute key
  - May be ideal for association relations
- Advantages:
  - Single attribute PK are faster to index than composite keys
  - Single attribute PK are faster to join than composite keys

Original table: **SALE** (<u>CustID</u>, <u>StoreID</u>, <u>Date</u>, <u>Time</u>, Amount)

New table: **SALE** (<u>TransID</u>, CustID, StoreID, Date, Time, Amount)

New PK to replace composite PK in original table

Adding Attributes: Storing Derived Data

- If the same values have to be calculated over and over again, compute them and store as new field
- Disadvantage:
  - Introduces transitive and/or partial dependency: Table no longer in 3NF/2NF
  - Data redundancy introduced: Derived data must be updated every time values are added or updated in the referenced fields
- Rule: Use sparingly and with caution

SALE (Trans_ID, Cust_ID, Store_ID, Date, Time, Amount)

CUSTOMER (Cust_ID, Name, SP_Num, Annual_Purchase)

Derived data calculated as Sum (Amount) for that Cust_ID in SALE table

Combining Tables: 1:1 Relationships

- Advantage:
  - Avoids the need for joins
  - Makes joint retrieval of data from both tables faster
- Disadvantages:
  - Tables no longer logically or physically independent
  - May introduce anomalies (e.g., can't add Office_Num without SP_Num)
  - Makes retrieval of data from each table slower
- Rule of thumb:
  - Somewhat safe for 1:1 relationships; don't try for 1:M or M:N relationships

Original tables:

SALESPERSON (<u>SP_Num</u>, Name, Commission, Year_Of_Hire, Office_Num)

OFFICE (<u>Office_Num</u>, Location, Phone)

Why is Office_Num no longer a foreign key?

New table:

SALESPERSON (<u>SP_Num</u>, Name, Commission, Year_Of_Hire, Office_Num, Location, Phone)

Combining Tables: Repeating Groups

- If repeating groups are well controlled, they can be folded into one table
- Rule of thumb:
  - "Well controlled" means no more than two repeating groups
  - If more than two of, if in doubt, don't combine tables

CUSTOMER (<u>Cust_ID</u>, Name, Address, City, State, Zip, Home_Phone, Work_Phone)

Where NOT to use this strategy:

SALESPERSON (<u>SP_Num</u>, Name, Commission, Year_Of_Hire, Office_Num)

SALESPERSON_CUSTOMER (<u>Cust_Num</u>, Name, City, Phone, SP_Num)

Combining Tables: Denormalization

- The process of combining normalized relations into larger unnormalized relations to improve database performance
- Normalized (3NF) relations:

  BOOK (Book_Num, Book_Name, Pub_Year, Pub_Name)

  PUBLISHER (Pub_Name, Pub_City, Pub_Phone)

- Denormalized relation:

  BOOK (Book_Num, Book_Name, Pub_Year,
  Pub_Name, Pub_City, Pub_Phone)

- Tradeoffs:
  - Less joins implies faster disk access and greater speed/ performance.
  - SQL statements much simpler.
  - Introduction of anomalies (and thus data redundancies) which will require more efforts on data maintenance.

Data Types

- Select the appropriate data type for columns
  - Try minimizing storage requirements

- General data types:
  - String: fixed or variable length character
  - Numeric: integer, floating point, etc.
  - Date/Time
  - Binary: files such as photos, etc.
  - Large Object (LOB): large blocks of text or binary data
- Oracle Data Types

Common Oracle Data Types

- Character string:
  - CHAR(size) Fixed length
  - VARCHAR(size) Variable length

- Example:
  - Country_code CHAR(2)
    - E.g., US, CA, UK, A, B
  - Country_code VARCHAR(3)
    - E.g., US, CA, UK, USA, CAN, A, B

Common Oracle Data Types

- Numeric: (choice depends on storage requirements)
  - INT — Integer (no decimal point)
  - FLOAT — (up to 23 digits – including decimal)
  - DOUBLE — (up to 53 digits – including decimal)
  - DECIMAL (M,N) — (M digits in total, N decimals)
  - UNSIGNED option (only allows positive values)
- Example:
  - Age INT
    - E.g., 3; 6; 100; 1,000,000 (no commas)
  - Weight DECIMAL(5, 2)
    - Any value from -999.99 to 999.99
    - Default of DECIMAL is 10 digits, no decimal places

Common Oracle Data Types

- Date/Time:
  - DATE yyyy-mm-dd
  - TIME hh:mm:ss (24-hour format)
  - DATETIME yyyy-mm-dd hh:mm:ss

- Examples:
  - Birth_date DATE
    - E.g., 1980-01-25; 1990-02-28
  - Birth_time TIME
    - E.g., 13:12:02; 11:45:00
  - Birth_datetime DATETIME
    - E.g., 1980-01-25 13:12:02

Common MySQL Data Types

- Others:
  - BLOB — Binary large objects
  - TEXT — Text (larger than VARCHAR)
  - ENUM — Values from a pre-specified group
    - Lookup tables commonly used
  - [Oracle Datatypes]
- Example:
  - Image — BLOB
  - Blog — TEXT
  - Weekend — ENUM(‘Sat’, ‘Sun’)
    - E.g., ‘Sat’ ; ‘Sun’ ; ‘Sat, Sun’