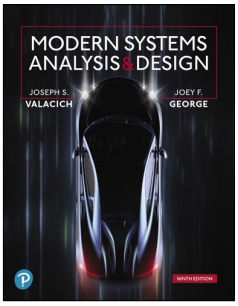
Modern Systems Analysis and Design

Ninth Edition



Chapter 9

Designing Databases

Pearson

1

Learning Objectives

9.1 Describe the database design process, its outcomes, and the relational database model

9.2 Describe normalization and the rules for second and third normal form

9.3 Transform an entity-relationship (E-R) diagram into an equivalent set of well-structured (normalized) relations

9.4 Merge normalized relations from separate user views into a consolidated set of well-structured relations

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

Pearson

2

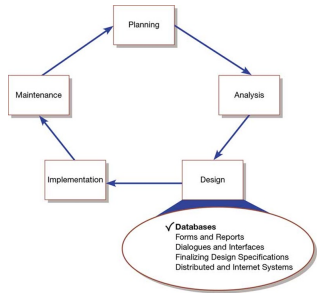
Introduction

- Database design has five purposes as follows:
  1. Structure the data in stable structures (normalize)
  2. Develop a logical database design that reflects the actual data requirements that exist in the forms
  3. Develop a logical database design as a basis for physical database design
  4. Translate a relational database model into a technical file and database design that balances several performance factors
  5. Choose data storage technologies that will efficiently, accurately, and securely process database activities

Pearson

3

Figure 9-1: Systems Development Life Cycle with Design Phase Highlighted



Pearson

4

Database Design

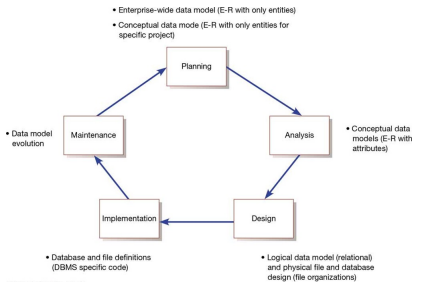
9.1 Describe the database design process, its outcomes, and the relational database model

- File and database design occurs in two steps:
  1. Develop a logical database model, which describes data using notation that corresponds to a data organization used by a database management system
     - Relational database model
  2. Prescribe the technical specifications for computer files and databases in which to store the data
     - Physical database design provides specifications
- Logical and physical database design in parallel with other system design steps

Pearson

5

Figure 9-2: Relationship Between Data Modeling and the SDLC



Pearson

6

Process of Database Design

9.1 Describe the database design process, its outcomes, and the relational database model

- Four key steps in logical database modeling and design:
  1. Develop a logical data model for each known user interface for the application using normalization principles
  2. Combine normalized data requirements from all user interfaces into one consolidated logical database model (view integration)
  3. Translate the conceptual E-R data model for the application into normalized data requirements
  4. Compare the consolidated logical database design with the translated E-R model and produce one final logical database model for the application

Pearson

7

Physical Database Design

9.1 Describe the database design process, its outcomes, and the relational database model

- Key physical database design decisions:
  - Choosing a storage format (data type) for each attribute from the logical database model
  - Grouping attributes from the logical database model into physical records
  - Arranging related records in secondary memory (hard disks and magnetic tapes) so that records can be stored, retrieved and updated rapidly
  - Selecting media and structures for storing data to make access more efficient

Pearson

8

Deliverables and Outcomes

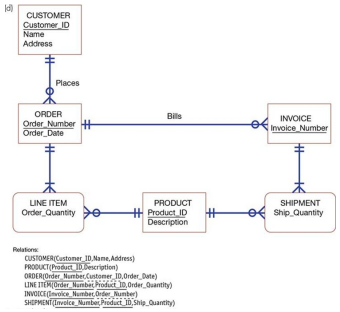
9.1 Describe the database design process, its outcomes, and the relational database model

- During logical database design you must account for every data element on a system input or output
  - Normalized relations are the primary deliverable
- **Primary key** – attribute (or combination of attributes) whose value is unique across all occurrences of a relation
- Physical database design converts relations into database tables
  - Programmers and database analysts code the definitions of the database using Structured Query Language (SQL)

Pearson

9

Figure 9-3(d): Conceptual Data Model and Transformed Relations



Pearson

10

The Relational Database Model (1 of 2)

9.1 Describe the database design process, its outcomes, and the relational database model

- **Relational database model** – data represented as a set of related tables or relations
- **Relation** – named, two-dimensional table of data. Each relation consists of a set of named columns and an arbitrary number of unnamed rows.

Pearson

11

The Relational Database Model (2 of 2)

9.1 Describe the database design process, its outcomes, and the relational database model

- Relations have several properties that distinguish them from nonrelational tables:
  - Entries in cells are simple
  - Entries in columns are from the same set of values
  - Each row is unique
  - The sequence of columns can be interchanged without changing the meaning or use of the relation
  - The rows may be interchanged or stored in any sequence

Pearson

12

Well-Structured Relations

9.1 Describe the database design process, its outcomes, and the relational database model

- **Well-structured relation** – relation that contains a minimum amount of redundancy and that allows users to insert, modify, and delete the rows without error or inconsistencies; also known as a table

Pearson

13

Normalization (1 of 2)

9.2 Describe normalization and the rules for second and third normal form

- **Normalization** – process of converting complex data structures into simple, stable data structures
- The result of normalization is that every nonprimary key attribute depends upon the whole primary key and nothing but the primary key

Pearson

14

Normalization (2 of 2)

9.2 Describe normalization and the rules for second and third normal form

- **First Normal Form (1NF):**
  - Has no multivalued attributes, unique rows, and all relations are in 1NF
- **Second Normal Form (2NF):**
  - Each nonprimary key attribute is identified by the whole key (referred to as a full functional dependency)
- **Third Normal Form (3NF):**
  - Nonprimary key attributes do not depend on each other (referred to as a transitive dependency)

Pearson

15

Functional Dependency (1 of 2)

9.2 Describe normalization and the rules for second and third normal form

- **Functional dependency** – constraint between two attributes in which the value of one attribute is determined by the value of another attribute
- Example of attribute B being functionality dependent on attribute A
  - Dependency is represented by an arrow (→)
  - Attribute B is functionally dependent on attribute A if, for every valid value of A, that value of A uniquely determines the value of B
  - Represented as: A → B

Pearson

16

Functional Dependency (2 of 2)

9.2 Describe normalization and the rules for second and third normal form

- Functional dependency is not a mathematical dependency
- Instances (or sample data) in a relation do not prove the existence of a functional dependency
- Knowledge of problem domain is most reliable method for identifying functional dependency

Pearson

17

Second Normal Form (2NF)

9.2 Describe normalization and the rules for second and third normal form

- **Second normal form (2NF)** – relation is in second normal form if every nonprimary key attribute is functionally dependent on the whole primary key
- To convert a relation into 2NF, decompose the relation into new relations using the attributes, called **determinants**, that determine other attributes
- The determinants are the primary keys of the new relations

Pearson

18

Third Normal Form (3NF) (1 of 2)

9.2 Describe normalization and the rules for second and third normal form

- **Third normal form (3NF)** – relation is in second normal form and has no functional (transitive) dependencies between two (or more) nonprimary key attributes

Pearson

19

Figure 9-9: Removing Transitive Dependencies (a) Relation with Transitive Dependency (b) Relation in 3NF

SALES

Customer_ID	Customer_Name	Salesperson	Region
8023	Anderson	Smith	South
9167	Bancroft	Hicks	West
7924	Hobbs	Smith	South
6837	Tucker	Hernandez	East
8596	Eckersley	Hicks	West
7018	Arnold	Faulb	North

SALES1

Customer_ID	Customer_Name	Salesperson
8023	Anderson	Smith
9167	Bancroft	Hicks
7924	Hobbs	Smith
6837	Tucker	Hernandez
8596	Eckersley	Hicks
7018	Arnold	Faulb

SPERSON

Salesperson	Region
Smith	South
Hicks	West
Hernandez	East
Faulb	North

Pearson

20

Third Normal Form (3NF) (2 of 2)

9.2 Describe normalization and the rules for second and third normal form

- **Foreign key** – attribute that appears as a nonprimary key attribute in one relation and as a primary key attribute (or part of a primary key) in another relation
- **Referential integrity** – rule that states that either each foreign key value must match a primary key value in another relation or the foreign key value must be null (i.e., have no value)

Pearson

21

Transforming E-R Diagrams into Relations

9.3 Transform an entity-relationship (E-R) diagram into an equivalent set of well-structured (normalized) relations

- Transforming an E-R diagram into normalized relations and merging all of them into one consolidated set of relations takes four steps:
  1. Represent entities (each becomes a relation)
  2. Represent relationships (each must be represented in the relational database design)
  3. Normalize the relations (make them well structured)
  4. Merge the relations (renormalize if necessary to remove redundancy)

Pearson

22

Representing Entities

9.3 Transform an entity-relationship (E-R) diagram into an equivalent set of well-structured (normalized) relations

- Each regular entity is transformed into a relation
- The identifier of the entity type becomes the primary key of the corresponding relation
- The primary key must satisfy the following two conditions
  - The value of the key must uniquely identify every row in the relation
  - The key should be nonredundant
- The entity type label is translated into a relation name

Pearson

23

Representing Relationships (1 of 5)

9.3 Transform an entity-relationship (E-R) diagram into an equivalent set of well-structured (normalized) relations

- The procedure for representing relationships depends on:
  - The degree of the relationship (unary, binary, ternary)
  - The cardinality of the relationship
- **Binary 1:*N* Relationship** – represented by adding the primary key attribute (or attributes) of the entity on the one side of the relationship as a foreign key in the relation that is on the many side of the relationship

Pearson

24

Representing Relationships (2 of 5)

9.3 Transform an entity-relationship (E-R) diagram into an equivalent set of well-structured (normalized) relations

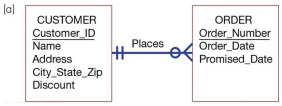
- **Binary or Unary 1:1 relationship** is represented by any of the following:
  - Add the primary key of A as a foreign key of B
  - Add the primary key of B as a foreign key of A
  - Both of the above

Pearson

25

Figure 9-11: Representing a 1:*N* Relationship (a) E-R Diagram (b) Relations

(a)



(b) CUSTOMER

Customer_ID	Name	Address	City_State_ZIP	Discount
1273	Contemporary Designs	123 Oak St.	Austin, TX 28384	5%
6390	Casual Corner	18 Hoosier Dr.	Bloomington, IN 45821	3%

ORDER

Order_Number	Order_Date	Promised_Date	Customer_ID
57194	3/15/2020	3/28/2020	6390
63725	3/17/2020	4/01/2020	1273
80149	3/14/2020	3/24/2020	6390

Pearson

26

Representing Relationships (3 of 5)

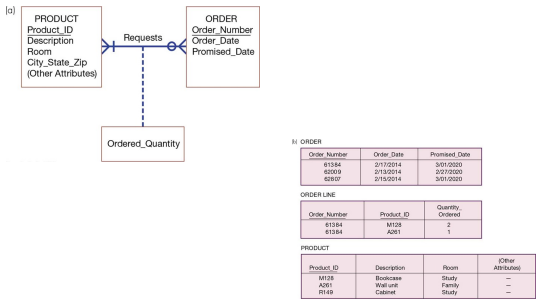
9.3 Transform an entity-relationship (E-R) diagram into an equivalent set of well-structured (normalized) relations

- **Binary and Higher-Degree *M*:*N* Relationships –** represented by creating another relations, include the primary keys of all relations into the new one as a primary key
  - Becomes a composite key
  - Any non-key attributes associated with the *M*:*N* relationship are included in the new relation

Pearson

27

Figure 9-12: Representing an *M*:*N* Relationship (a) E-R Diagram (b) Relations



28

Representing Relationships (4 of 5)

9.3 Transform an entity-relationship (E-R) diagram into an equivalent set of well-structured (normalized) relations

- **Unary 1:*N* Relationship** (also called recursive relationships):
  - Is modeled as a relation
  - Primary key of that relation is the same as for the entity type
  - Foreign key is added to the relation that references the primary key values
- **Recursive foreign key** – foreign key in a relation that references the primary key values of that same relation

29

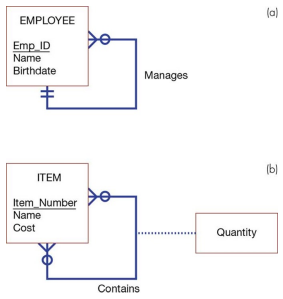
Representing Relationships (5 of 5)

9.3 Transform an entity-relationship (E-R) diagram into an equivalent set of well-structured (normalized) relations

- **Unary *M*:*N* Relationship** is modeled as one relation, then:
  - Create a separate relation to represent the *M*:*N* relationship
  - The primary key of the new relation is a composite key of two attributes that both take their values from the same primary key
  - Any attribute associated with the relationship is included as a nonkey attribute in this new relation

30

Figure 9-13: Two Unary Relationships (a) EMPLOYEE with Manages Relationship (1:*N*) (b) Bill-of-Materials Structure (*M:N*)



31

Table 9-1: E-R Diagrams to Relational Transformation

E-R Structure	Relational Representation
Regular entity	Create a relation with primary key and nonkey attributes.
Weak entity	Create a relation with a composite primary key (which includes the primary key of the entity on which this weak entity depends) and nonkey attributes.
Binary or unary 1:1 relationship	Place the primary key of either entity in the relation for the other entity or do this for both entities.
Binary 1:*N* relationship	Place the primary key of the entity on the one side of the relationship as a foreign key in the relation for the entity on the many side.
Binary or unary *M:N* relationship or associative entity	Create a relation with a composite primary key using the primary keys of the related entities, plus any nonkey attributes associative entity of the relationship or associative entity.
Binary or unary *M:N* relationship or associative entity with additional key(s)	Create a relation with a composite primary key using the primary keys of the related entities and additional primary key attributes associated with the relationship or associative entity, plus any nonkey attributes of the relationship or associative entity.
Binary or unary *M:N* relationship or associative entity with its own key	Create a relation with the primary key associated with the relationship or associative entity, plus any nonkey attributes of the relationship or associative entity and the primary keys of the related entities (as foreign key attributes).
Supertype/subtype	Create a relation for the superclass, which contains the primary relationship key and all nonkey attributes in common with all subclasses, plus create a separate relation for each subclass with the same primary key (with the same or local name) but with only the nonkey attributes related to that subclass.

32

Merging Relations

9.4 Merge normalized relations from separate user views into a consolidated set of well-structured relations

- Merging relations
  - Is the last step in the logical database design
  - Purpose is to remove redundant relations
  - Example when given two relations:
    - EMPLOYEE1(Emp_ID,Name,Address,Phone)
    - EMPLOYEE2(Emp_ID,Name,Address,Jobcode,Number_of_Years)
  - They can be merged together:
    - EMPLOYEE(Emp_ID,Name,Address,Phone,Jobcode,Number_of_Years)

33

View Integration Problems (1 of 4)

9.4 Merge normalized relations from separate user views into a consolidated set of well-structured relations

- Must understand the meaning of the data and be prepared to resolve any problems that arise in the process
- **Synonym** – two different names used for the same attribute
  - When merging, get agreement from users on a single, standard name
  - Example of two relations with synonym primary keys (representing SSN numbers) of different names:
    - STUDENT1(Student_ID,Name)
    - STUDENT2(Matriculation_Number,Name,Address)

Pearson

34

View Integration Problems (2 of 4)

9.4 Merge normalized relations from separate user views into a consolidated set of well-structured relations

- **Homonym –** single attribute name that is used for two or more different attributes
  - Resolved by creating a new descriptive name
  - Example: home address vs local address?
    - STUDENT1(Student_ID,Name,Address)
    - STUDENT2(Student_ID,Name,Phone_Number,Address)

Pearson

35

View Integration Problems (3 of 4)

9.4 Merge normalized relations from separate user views into a consolidated set of well-structured relations

- **Dependencies between nonkeys** occurs when two 3NF relations are merged to form a single relation such as:
  - STUDENT1(Student_ID,Major)
  - STUDENT2(Student_ID,Adviser)
  - Since both have the same primary key they can be merged as follows:
    - STUDENT(Student_ID,Major,Adviser)
- If a transitive dependency exists such as Major → Advisor
  - You need to **normalize** to remove the transitive dependency
    - STUDENT(Student_ID,Major)
    - MAJOR ADVISER(Major,Adviser)

Pearson

36

View Integration Problems (4 of 4)

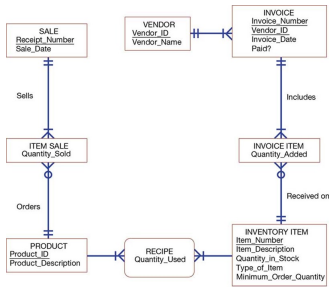
9.4 Merge normalized relations from separate user views into a consolidated set of well-structured relations

- **Class/Subclass** relationships may be hidden in user views or relations. Consider the following:
  - PATIENT1(Patient_ID,Name,Address,Date_Treated)
  - PATIENT2(Patient_ID,Room_Number)
- What if PATIENT can refer to both inpatient and outpatient? Then what?
- The answer? Convert it to a Supertype/Subtype!
  - PATIENT(Patient_ID,Name,Address)
  - INPATIENT(Patient_ID,Room_Number)
  - OUTPATIENT(Patient_ID,Date_Treated)

Pearson

37

Figure 9-16: E-R Diagram Corresponding to Normalized Relations of Hoosier Burger's Inventory Control System



Pearson

38

Final Normalized Relations for Hoosier Burger

9.4 Merge normalized relations from separate user views into a consolidated set of well-structured relations

Final normalized relations for Hoosier Burger:

SALE(Receipt_Number,Sale_Date)
PRODUCT(Product_ID,Product_Description)
INVOICE(Vendor_ID,Invoice_Number,Invoice_Date,Paid?)
INVENTORY ITEM(Item_Number,Item_Description,Quantity_in_
Stock,Minimum_Order_Quantity,Type_of_Item)
ITEM SALE(Receipt_Number,Product_ID,Quantity_Sold)
INVOICE
ITEM(Vendor_ID,Invoice_Number,Item_Number,Quantity_Added)
RECIPE(Product_ID,Item_Number,Quantity_Used)
VENDOR(Vendor_ID,Vendor_Name)

Pearson

39

Physical File and Database Design

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- Designing physical files/databases requires the following information:
  - Normalized relations, including volume estimates
  - Definitions of each attribute
  - Descriptions of where and when data are used: entered, retrieved, deleted, and updated (including frequencies)
  - Expectations or requirements for response time and data integrity
  - Descriptions of the technologies used for implementing the files and database so that the range of required decisions and choices for each is known

Pearson

40

Designing Fields

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- **Field** – smallest unit of named application data recognized by system software
  - An attribute from a relation is now recognized as a field in a database
- **Data type** – coding scheme recognized by system software for representing organizational data

Pearson

41

Choosing Data Types

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- Selecting a data type requires balancing four objectives:
  - Minimize storage space
  - Represent all possible values of the field
  - Improve data integrity of the field
  - Support all data manipulations desired on the field

Pearson

42

Table 9-2: Commonly Used Data Types in Oracle 10g

Data Type	Description
VARCHAR2	Variable-length character data with a maximum length of 4,000 characters; you must enter a maximum field length (e.g., VARCHAR2(30) for a field with a maximum length of 30 characters). A value less than 30 characters will consume only the required space.
CHAR	Fixed-length character data with a maximum length of 255 characters; default length is 1 character (e.g., CHAR(5) for a field with a fixed length of five characters, capable of holding a value from 0 to 5 characters long).
LONG	Capable of storing up to two gigabytes of one variable-length character data field (e.g., to hold a medical instruction or a customer comment).
NUMBER	Positive and negative numbers in the range 10–130 to 10126; can specify the precision (total number of digits to the left and right of the decimal point) and the scale (the number of digits to the right of the decimal point) (e.g., NUMBER(5) specifies an integer field with a maximum of 5 digits and NUMBER(5, 2) specifies a field with no more than five digits and exactly two digits to the right of the decimal point).
DATE	Any date from January 1, 4712 BC to December 31, 4712 AD; date stores the century, year, month, day, hour, minute, and second.
BLOB	Binary large object, capable of storing up to four gigabytes of binary data (e.g., a photograph or sound clip).

Pearson

43

Calculated Fields

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- **Calculated field** – field that can be derived from other database fields. Also known as a computed field or a derived field.
- It is common for an attribute to be mathematically related to other data
- The database will either stored or compute the calculated field when requested

Pearson

44

Controlling Data Integrity

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- **Default value** – value a field will assume unless an explicit value is entered for that field
- **Range control –** limits values (numeric or alpha-numeric data) that can be entered into a field
- **Referential integrity** – constraint specifying that the value (or existence) of an attribute in one relation depends on the value (or existence) of the same attribute in another relation
- **Null value** – special field value, distinct from zero, blank, or any other value, that indicates that the value for the field is missing or otherwise unknown

Pearson

45

Figure 9-17: Examples of Referential Integrity Field Controls (a) Referential Integrity Between Relations (b) Referential Integrity Within a Relation

CUSTOMER (**Customer_ID**,Cust_Name,Cust_Address, . . .)

CUST_ORDER (Order_ID,**Customer_ID**,Order_Date, . . .)
and Customer_ID may not be null because every order must be for some existing customer

EMPLOYEE(**Employee_ID**,**Supervisor_ID**,Empl_Name, . . .)
and Supervisor_ID may be null because not all employees have supervisors

Pearson

46

Designing Physical Tables (1 of 3)

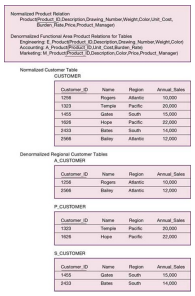
9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- A relational table is a set of related tables related by foreign keys referencing primary keys
- **Physical table** – named set of rows and columns that specifies the fields in each row of the table
- **Denormalization** – process of splitting or combining normalized relations into physical tables based on affinity of use of rows and fields

Pearson

47

Figure 9-18: Examples of Denormalization (a) Denormalization by Columns (b) Denormalization by Rows



Pearson

48

Designing Physical Tables (2 of 3)

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- Partitioning is the capability to split a table into separate sections. Partitioning types include:
  - **Range partitioning:** partitions are defined by nonoverlapping ranges of values for a specified attribute
  - **Hash partitioning:** a table row is assigned to a partition by an algorithm and then maps the specified attribute value to a partition
  - **Composite partitioning:** combines range and hash partitioning by first segregating data by ranges on the designated attribute, and then within each of these partitions

Pearson

49

Designing Physical Tables (3 of 3)

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- Partitioning helps speed up system performance
- Denormalization can increase change of errors
- Three common situations where denormalization is used are:
  1. Two entities with a one-to-one relationship
  2. A many-to-many relationship (associative entity) with nonkey attributes
  3. Reference data

Pearson

50

Arranging Table Rows (1 of 5)

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- The result of denormalization is the definition of one or more physical files
- **Physical file** – named set of table rows stored in a contiguous section of secondary memory
- **File organization** – technique for physically arranging the records of a file

Pearson

51

Arranging Table Rows (2 of 5)

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- Objectives for choosing file organization include:
  - Fast data retrieval
  - High throughput for processing transactions
  - Efficient use of storage space
  - Protection from failures or data loss
  - Minimizing need for reorganization
  - Accommodating growth
  - Security from unauthorized use

Pearson

52

Arranging Table Rows (3 of 5)

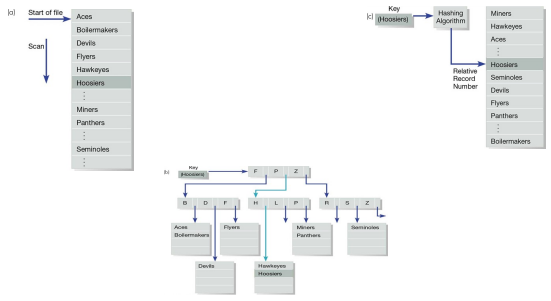
9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- **Pointer** – field of data that can be used to locate a related field or row of data
- Three basic families of file organization:
  - **Sequential file organization** – file organization in which rows in a file are stored in sequence according to a primary key value
  - **Indexed file organization** – file organization in which rows are stored either sequentially or nonsequentially, and an index is created that allows software to locate individual rows
  - **Hashed file organization** – file organization in which the address of each row is determined using an algorithm

Pearson

53

Figure 9-20: Comparison of File Organizations (a) Sequential (b) Indexed (c) Hashed



Pearson

54

Arranging Table Rows (4 of 5)

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- **Index** – table used to determine the location of rows in a file that satisfy some condition
- **Secondary key –** represents one or a combination of fields for which more than one row may have the same combination of values
  - Allows an index to point to more than one record

Pearson

55

Arranging Table Rows (5 of 5)

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- Indexed file organization
  - Advantage is allowing for both random and sequential processing
  - Disadvantages include:
    - Extra space required to store indexes
    - Extra time necessary to access and maintain indexes
  - Guidelines for choosing indexes include:
    - Specify a unique index for the primary key of each table
    - Specify an index for foreign keys
    - Specify an index for nonkey fields that are referenced in qualification, sorting and grouping commands for the purpose of retrieving data

Pearson

56

Table 9-3: Comparative Features of Sequential, Indexed, and Hashed File Organizations

Factor	File Sequential	Organization Indexed	Hashed
Storage space	No wasted space	No wasted space for data, but extra space for index	Extra space may be needed to allow for addition and deletion of records
Sequential retrieval on primary key	Very fast	Moderately fast	Impractical
Random retrieval on primary key	Impractical	Moderately fast	Very fast
Multiple key retrieval	Possible, but requires scanning whole file	Very fast with multiple indexes	Not possible
Deleting rows	Can create wasted space or require reorganizing	If space can be dynamically allocated, this is easy, but requires maintenance of indexes	Very easy
Adding rows	Requires rewriting file	If space can be dynamically allocated, this is easy, but requires maintenance of indexes	Very easy, except multiple keys with same address require extra work
Updating rows	Usually requires rewriting file	Easy, but requires maintenance of indexes	Very easy

Pearson

57

Designing Controls for Files (1 of 2)

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- Two goals of physical table design:
  1. Protection from failure or data loss
  2. Security from unauthorized use
- These goals are achieved primarily by implementing controls on each file
- Two additional types of controls address
  1. Backup
  2. Security

Pearson

58

Designing Controls for Files (2 of 2)

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

- Techniques for file restoration:
  - Periodically making a backup copy of a file
  - Storing a copy of each change to a file in a transaction log or audit trail
  - Storing a copy of each row before or after it is changed
- Means of building data security into a file:
  - Coding, or encrypting, the data in the file
  - Requiring data file users to identify themselves by entering user names and passwords
  - Prohibiting users from directly manipulating any data in the file by forcing users to work with a copy (real or virtual)

Pearson

59

Summary (1 of 2)

In this chapter you learned how to:

9.1 Describe the database design process, its outcomes, and the relational database model

9.2 Describe normalization and the rules for second and third normal form

9.3 Transform an entity-relationship (E-R) diagram into an equivalent set of well-structured (normalized) relations

9.4 Merge normalized relations from separate user views into a consolidated set of well-structured relations

Pearson

60

Summary (2 of 2)

In this chapter you learned how to:

9.5 Describe physical database design concepts including choosing storage formats for fields in database tables, translating well-structured relations into efficient database tables, explaining when to use different types of file organizations to store computer files, and describing the purpose of indexes and the important considerations in selecting attributes to be indexed

Pearson

61

Copyright

Pearson

62