CST 3604 Physical Design & Implementation

Chapter 2 – The Physical Data Model

Prof Rodriguez

Supplemental notes to book:

Essential Aspects of Physical Design and Implementation of Relational Databases

Agenda

- Goals of the physical model
- Tables – data types, constraints
- Indices
- Data storage organization
- Heap and organized data storage

Physical Model Definition

❑ Definition:

- Technical Specifications for processing & storing data that will ensure:
  - Adequate Efficiency & Performance (Fast retrieval)
  - Data Integrity (correct, consistent, accessible)
  - Security (secure, who is authorized or who can see it, etc)
  - Recoverability (redundancy & backup)
  - Other

❑ Database Professional that owns physical model:

- Database Administrator (DBA)

Physical Model Goals

- **Primary Goals – Based on user requirements achieve:**
  - **Efficiency** & **performance** – fast retrieval or **SPEED** (fast response time or get query results fast)

- **Secondary Goals:**
  - Data Integrity
  - Security
  - Recoverability
  - Reduced Maintenance/cost

Physical Model Goals

- **Focus of physical model to achieve goals:**
  - Column Data Types
  - Where data or table is stored on physical disk
  - How storage is organized
  - Other

Physical Model Goals

- **Other factor to achieve goals:**
  - **Help from features provided by chosen DBMS**
  - Compromise due to conflicts between requirements:
    - Decision to gain efficiency for one requirement such as read queries (SELECT) may negatively affect another requirement such as queries that modify (INSERT/UPDATE)
    - Compromise may require a solution that is **adequate** for **all** requirements instead of the best efficiency for one.

Database Tables

- Table Basics
- Table Design – Selecting Data Types
- Special Data Types
- Data types constrains or rules for data integrity

Tables

Tables:

- **Relational Database Management Systems store data in tables (Columns & Rows)**
  - **Columns –** represent attributes in EER model & where attribute data is stored
  - **Rows –** represent on record

Student ID	Student Name	Student Address	Birth Date	Major
642-17-7656	Samuel Johnson	100 E45 th Street, New York, NY 20011	08/16/1970	Math
456-54-9876	Maritza Pena	453 Flatbush Ave, Brooklyn, NY, 11210	01/23/1965	Computer Science
123-65-4532	Nancy Jones	312 Jay Street, Brooklyn, NY 11201	01/4/1960	Accounting

Tables – Column Data Types

- **Column Data Types:**
  - **Storage mechanism provided by DBMS to store the values of the attributes or columns**
  - **Every DBMS has a list of available data types to choose from the DBMS you choose will determine choices available**
  - **Data Types:**
    1. Define type of values or type of data can be entered to a column
    2. Define the Properties/characteristics (Name, meaning, data type, size and range, etc.) of the data
- **Data Types Properties/Size, Characteristics known as METADATA:**

Tables – Column Data Types

❑ Typical available Data Types in most DBMS

- **Numeric** – stores numbers
- **Character** – stores characters (numbers, letters, symbols etc.)
- **Dates** – stores dates (short dates: 01/12/1971, long dates: January 01, 1971, etc.)
- **Time** – stores timestamp
- **BLOB** – Binary Large Objects (for storing large amount of binary data such as images, videos & multimedia data)
- **CLOB** – Character Large Objects (for storing large amount of character data)
- Others

Tables – Column Data Types

❑ Selecting Data Types

- DBMS you choose will determine choices of data type available
- Example of Oracle 11g data types

TABLE 5-1 Commonly Used Data Types in Oracle 11g

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) specifies a field with a maximum length of 30 characters]. A string that is shorter than the maximum will consume only the required space.
CHAR	Fixed-length character data with a maximum length of 2,000 characters; default length is 1 character (e.g., CHAR(5) specifies a field with a fixed length of 5 characters, capable of holding a value from 0 to 5 characters long).
CLOB	Character large object, capable of storing up to (4 gigabytes – 1) * (database block size) of one variable-length character data field (e.g., to hold a medical instruction or a customer comment).
NUMBER	Positive or negative number in the range 10^{-130} to 10^{126}; 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). For example, NUMBER(5) specifies an integer field with a maximum of 5 digits, and NUMBER(5,2) specifies a field with no more than 5 digits and exactly 2 digits to the right of the decimal point.
DATE	Any date from January 1, 4712 B.C., to December 31, 9999 A.D.; DATE stores the century, year, month, day, hour, minute, and second.
BLOB	Binary large object, capable of storing up to (4 gigabytes – 1) * (database block size) of binary data (e.g., a photograph or sound clip).

Table Design – Selecting Column Data Types

- **Design objectives for selecting Data Types**
  - **Data type must represent all possible values for the column**
  - **Minimized storage space (economical)**
  - **Enforce column data integrity (correctness, consistent, accessibility)** – Eliminate illegal values
  - **Support all required data manipulations** – Arithmetic Operations
  - **Make column maintenance easier**

Table Design – Selecting Column Data Types

- **Example of creating a table using CREATE statement & Oracle data types:**

```
CREATE TABLE Title (
    titleCode CHAR(2) PRIMARY KEY,
    titleDescription VARCHAR2(15),
    salary NUMBER (7));
```

Table Design – Selecting Column Data Types

Design objectives met by example:

Represent all possible values for the column:

- The following column **titleCode** uses Character Data Type **CHAR(2)** menans it can store either characters (BY, CO, etc.) or combination of character & digits (B5, C3, etc.)

```
titleCode CHAR(2)
```

- This represents all possible values for the titleCode column
- If numeric had been chosen, then only numbers of digits would be allowed & no characters

Table Design – Selecting Column Data Types

- **Design objectives met by example (Cont.):**
  - **Support all required data manipulations or operations:**
    - The following column could use either character or numeric. Nevertheless, numeric is a better choice for numeric operations on the column such as **Average(salary)**

```
salary NUMBER (7))
```

    - In addition, the **size** of the numeric number was chosen at 7 digit. Nevertheless, supposed salaries are NOT greater than $100K, then NUMBER (5) could have been a good candidate & we save storage. But it may not be enough to support operations such as **SUM(salary)**, which would be greater than NUMBER(5)

Table Design – Selecting Column Data Types

Design objectives met by example (Cont.):

Minimized storage space (economical)

- The following column uses VARIABLE CHARACTER or **VARCHAR2(15)** so that the value for every title description stored in the column occupies in memory the exact number of characters of the description. No more, no less.

```
titleDescription VARCHAR2(15)
```

- Supposed we used FIXED LENGH CHAR or **CHAR(15)** so that every title description value occupies 15 characters. Then if the description is "DBA", that only occupies 3 characters, **which means 12 empty character will be stored & wasted. Not economical**

Table Design – Selecting Column Data Types

Design objectives met by example (Cont.):

A word on VARCHAR & VARCHAR2 data types

- It is important to understand the details of data types supported in a particular DBMS. For example, ORACLE supports two character data types with variable length: **VARCHAR** and **VARCHAR2**.
- Where appropriate, it is recommended to use **VARCHAR2** since:
  1. VARCHAR can store up to 2000 bytes and VARCHAR2 can store up to 4DOO bytes
  2. NULL values of the column declared as VARCHAR will occupy space while NULL values of columns declared as VARCHAR2 will not.

Table Design – Selecting Column Data Types

❑ Design objectives met by example:

Enforce column data integrity:

- The following column **titleCode** uses Character Data Type **CHAR(2)** of size 2 since **titleCode CANNOT** be longer than 2 characters. This data type and size guarantees it, thus ensuring data integrity.

```
titleCode CHAR(2)
```

- On the other hand, there is one flaw in this choice of data type for column **titleCode**, the uses ***Character Data*** Type **ALLOWS** for the insertion of values with SYMBOLS such as !@#$%^&&*() etc., this will affect Data Integrity

Table Design – Selecting Column Data Types

- **Design objectives met by example (Alternate implementation to titleCode:**
  - **Enforce column data integrity:**
    - If the following column **titleCode** was composed of DIGITS ONLY, then instead of ***Character Data*** Type **CHAR(2)** of size 2, we would use ***NUMERIC*** or **NUMBER(2)** data type would be a better option since the ***Character Data*** Type would **NOT PREVENT** inserting values with SYMBOLS such as !@#$%^&&*() etc. ***NUMERIC*** or **NUMBER(2)** data type would guarantees numbers only, thus ensuring data integrity.

```
titleCode NUMERIC(2);
```

Table Design – Selecting Column Data Types

Design objectives met by example (Cont.):

Enforce column data integrity

- The following column uses a NUMERIC or **NUMBER(7)** data type. Thus guaranteeing that the value is a number and NOT a character. Thus we enforce data integrity.

```
salary NUMBER (7))
```

Tables – Special Column Data Types

❑ IDENTITY or AUTONUMBER Data Type

- Some DBMS offer a data type feature that **automatically generates and assigns a value to a column when inserting data**
- This is a convenient technique when you need unique IDs. Imagine if you had an EMPLOYEE table targeted for thousands of employees which required unique EmployeeID. You would need to create a unique ID for every employee. That would be a daunting task.
- Using special data types or special database objects that enforce automatic assignment of a new column value can be extremely beneficial

Tables – Special Column Data Types

❑ IDENTITY or AUTONUMBER Data Type (Cont.)

- Examples of such features are:
  - **Microsoft Access** – AUTONUMBER data type
  - **Microsoft SQL Server** – IDENTITY data type

- If, for example, the employeeID column in the table Employee is declared as AUTONUMBER, then for every inserted new employee the system generates the next integer number and assigns it to the ID column: '1' for the first inserted employee, '2' for the second, and so on.

Tables – Special Column Data Types

❑ IDENTITY or AUTONUMBER Data Type (Cont.)

- Prior version of Oracle, did not support this feature.
- ORACLE 12 now has the following data type:
  - **IDENTITY** – IDENTITY feature on the numeric column.
  - This data type contains options with keyword **GENERATED ALWAYS** to implement this auto-number feature

Tables – Special Column Data Types

❑ IDENTITY or AUTONUMBER Data Type (Cont.)

- Example of this new ORACLE 12 feature:

```
CREATE TABLE Employee (
employeeID NUMBER GENERATED ALWAYS AS IDENTITY
emplName VARCHAR2(30),
emplAddress VARCHAR2(50);
) ;
```

Tables – Special Column Data Types

❑ IDENTITY or AUTONUMBER Data Type (Cont.)

- Now you can simply insert data about employees while the system will automatically provide the unique values for employeeID
- The SQUL INSERT query would look as follows:

```
INSERT INTO Employee (emplName, emplAddress)
VALUES ('John Smith', '300 Jay Street Brooklyn NY' );
```

- Note that the VALUES section DOES NOT contain the employee ID values. That is automatically inserted by the system

Tables – Special Column Data Types

❑ Oracle- Virtual Columns

- Virtual column definition:
  - A column that is virtual or NOT physically stored on disk
  - Virtual columns are derived from other columns of the table, thus reason not needed to be stored on disk.
  - Virtual columns are listed in the **CREATE TABLE** statement
  - But exists only in memory when queries are executed against table.

- Usage:
  - Scenarios where a required value is calculated from other columns
  - Thus calculated value can exist in memory and not on disk, thus saving space by not requiring another column

Tables – Special Column Data Types

Oracle Virtual Columns Example

- Suppose you have the following logical model for employees of a company:

```
Title (titleCode, titleDescription, salary)

Department (deptCode, deptName, location, deptType)

Employee (ID, emplName, emplType, deptCode, titleCode)
```

Tables – Special Column Data Types

Oracle Virtual Columns Example (Cont.)

- Example, employees have bonus that is calculated as a percent of their salary (salary*0.3) & you want to capture this in the **Title table** in a column named bonus.
- You could add another column to Title table, but that would be a waste of space
- Using Virtual Columns, the CREATE TABLE statement for the **Title table** would look as follows:

```
CREATE TABLE Title (
    titleCode CHAR(2) PRIMARY KEY,
    titleDescription VARCHAR2(15),
    salary NUMBER (7),
    bonus NUMBER GENERATED ALWAYS AS (ROUND(salary*0.3,2))
    VIRTUAL);
```

Tables – Special Column Data Types

Oracle Virtual Columns Example (Cont.)

- **INSERT Queries** to add records to this table DO NOT include the VIRUTAL Colum:
  - Sample query:

```
INSERT INTO Title (titleCode, titleDescription, salary)
VALUES ('T1', 'DBA', 60000);
```

  - Sample query results – The following record is inserted into the Title Table:

```
T1          DBA                    60000       18000
```

Tables – Special Column Data Types

❑ Oracle Virtual Columns Example (Cont.)

- Note that the bonus column is part of the CREATE TABLE statement. But the keyword **VIRTUAL** is used to indicate this column is derived and does not exist on the disk.
- Note that the **GENERATED ALWAYS AS** keyword is used and the bonus calculated.
- **SELECT Queries** executed on this table will return the bonus column, but this column is derived and does not exist on the disk:
  - Sample query:

```
SELECT * FROM Title WHERE titleCode = 'T1';
```

  - Sample query results:

TITLECODE	TITLEDESCRIPTION	SALARY	BONUS
T1	DBA	60000	18000

Tables – Data Types & Constraints

- **DBMS Column Data Types can enforce Data Integrity via constrains**
  - If you recall from previous semester, Integrity Constraints are rules imposed on attributes
  - DBMSs can provide constraints that help enforce data integrity & other data type objectives
  - Examples of these constraints are:
    - **NOT NULL** – Values are required in a column. No NULLS allowed
    - **CHECK** – A **CONDITION** or **VALIDATION** that is done every time row is **INSERTED** or **UPDATED**. Condition must be satisfied before a row is **INSERTED** or **UPDATED**
    - **UNIQUE** – Enforces uniqueness of a column or combination of columns in a table. Column values must be unique. Used to enforce uniqueness on columns that **ARE NOT PRIMARY KEY**.

Tables – Data Types & Constraints

NOT NULL Constraints example

- The following CREATE TABLE statement, enforces the **NOT NULL** constraints on the ***tableDescription*** & ***salary*** columns so these columns cannot contain NULLS:

```
CREATE TABLE Title (
    titleCode CHAR(2) PRIMARY KEY,
    titleDescription VARCHAR2(15) NOT NULL,
    salary NUMBER(7) NOT NULL
);
```

- The business rules enforced here are
  - Every title MUST have a description
  - A title MUST have a salary. No employee can work for free. Must be paid.

Tables – Data Types & Constraints

CHECK Constraints example

- The following CREATE TABLE statement, enforces the **CHECK** constraints that a salaries have the CONDITION that they must be within a specific range of 30K to 90K only:

```
CREATE TABLE Title (
  titleCode CHAR(2) PRIMARY KEY,
  titleDescription VARCHAR2(15) NOT NULL,
  salary NUMBER(7)CHECK(salary BETWEEN 30000 AND90000)NOT NULL
);
```

- New business rules enforced here are
  - Employees in this company can only make within 30K to 90K salaries
  - No employee makes over 90K in the company.

Tables – Data Types & Constraints

UNIQUE Constraints example 1

- The following CREATE TABLE statement, enforces the **UNIQUE** constraints that a username must be unique:

```
CREATE TABLE UserAccount (
  userAccountID NUMBER PRIMARY KEY,
  username VARCHAR2(25) NOT NULL UNIQUE,
  password VARCHAR2(25) NOT NULL
);
```

- Business rules enforced here are
  - Usernames must be unique

- I need to confirm this syntax
- Not sure if still supported

Tables – Data Types & Constraints

UNIQUE Constraints example 2

- The following CREATE TABLE statement will use the following syntax to implement **UNIQUE** constraints:

```
CREATE TABLE table_name
(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],
  ...

  CONSTRAINT constraint_name UNIQUE (uc_col1, uc_col2, ... uc_col_n)
);
```

Tables – Data Types & Constraints

UNIQUE Constraints example 2 (Cont.)

- The following CREATE TABLE statement, uses a different syntax to implement the **UNIQUE** constraints that a username must be unique :

```
CREATE TABLE UserAccount (
  userAccountID NUMBER PRIMARY KEY,
  username VARCHAR2(25) NOT NULL,
  password VARCHAR2(25) NOT NULL,
  CONSTRAINT user_unique UNIQUE(username)
);
```

- Business rules enforced here are
  - Usernames must be unique

Data Storage Management

- Files
- Tablespaces
- Segments, Extents & Blocks

Data Storage – Data Files

- Most DBMS store data in an Operating System Physical File
- Physical File (Data file):
  - A part of secondary memory (Hard Disk), given a name & allocated for the purpose of storing physical records (tables, objects, etc.)
  - Each database must have at least one data file
- Physical File Examples:
  - Tablespace (Oracle) – Logical storage space used by Oracle to organize data files
  - File Group (MS SQL) – Same definition as tablespace but for MS SQL Server

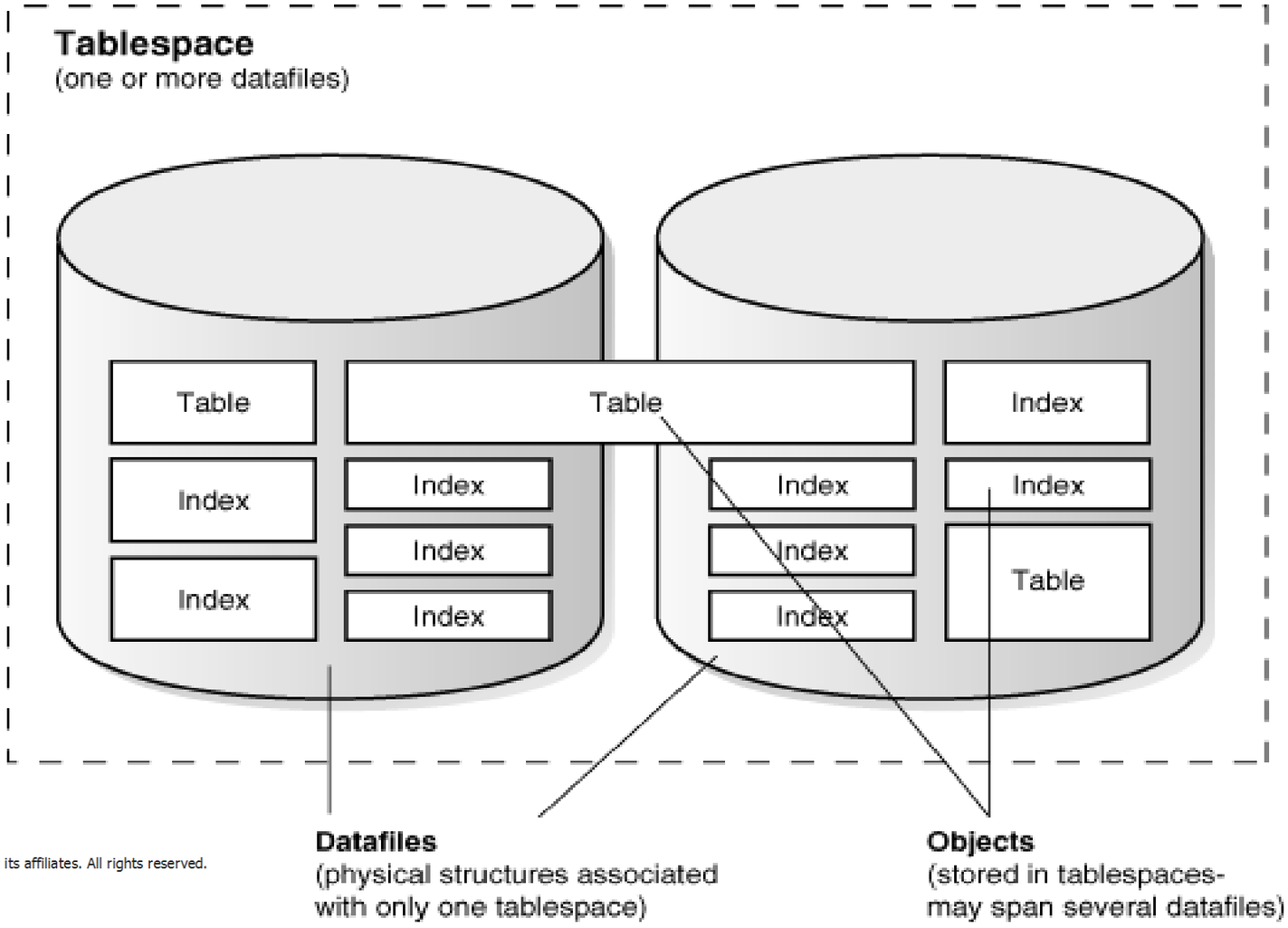
Data Storage – Tablespace

- **Tablespace is logical storage unit used by Oracle to organize data files**
  - **A database is divided into one or more tablespaces**
  - **Oracle stores data objects (tables/views/objects, etc.) in tablespaces (logical storage unit)**
  - **A Tablespace Consists of ONE or MORE Physical Operating System Files**
  - **Tables & tablespace:**
    - Tables are specified by Tablespaces not by data file, therefore a table can be stored in more than one data file, but **ONLY ONE** tablespace.

Data Storage – Tablespace

Illustration of a tablespace

- Note data objects such as tables, indexes, views, etc. can expand more than one data file, but ONLY one tablespace



Data Storage – Segment & Extent

Tablespace components

Segment

- Logical units that make up a tablespace. A **tablespace** is divided into segments. Segments are divided into extents.
- Segments are allocated to **specific type of information** stored in the same **tablespace**. For example, a table's data is store in one segment
- Segments can expand multiple files

Extent

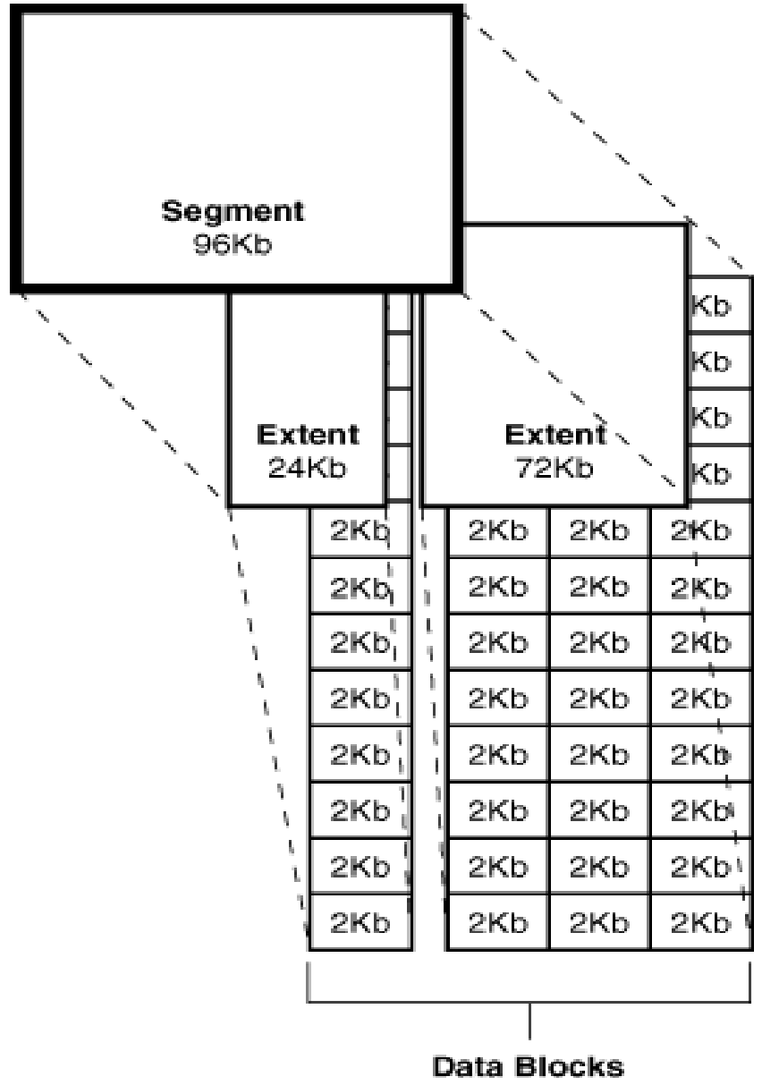
- Segments are divided into contiguous (together in a sequence) section of disk space called extents.
- Extents store **specific type of information**.
- Extents belong to specific files and do not extend multiple files.
- Extents are composed of data blocks

Data Storage – Tablespace

- **Tablespace components**
  - **Data block**
    - Smallest unit of storage within an extent (called logical blocks, oracle blocks or pages)
    - One data block corresponds to specific number of bytes of physical space on disk

Data Storage – Tablespace

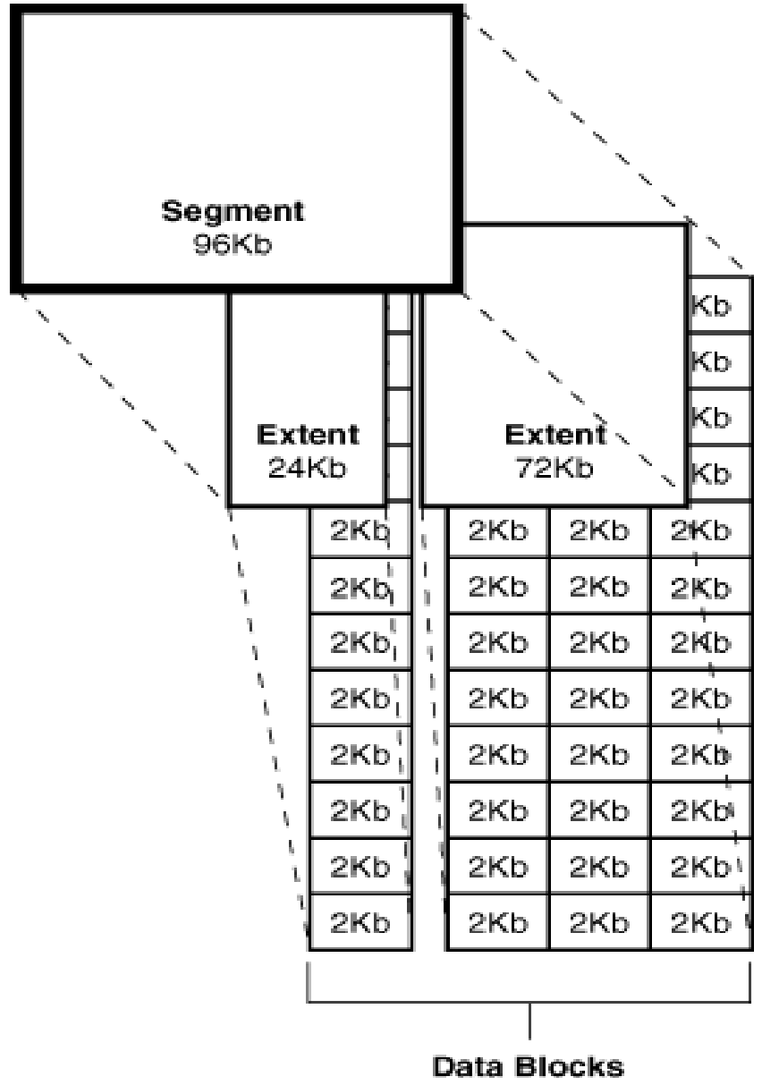
- **Illustration of a Segments, Extents & Data Blocks**
  - Figure shows one **segment** of 96kb
  - Segments can expand multiple files
  - The **segment** is divided into 2 **extents**. One of 24kb & one of 72kb.
  - Each extent is divided into **data blocks** or smallest unit that stores the data. Each data block is of 2kb size.



Data Storage – Tablespace

Space Allocation

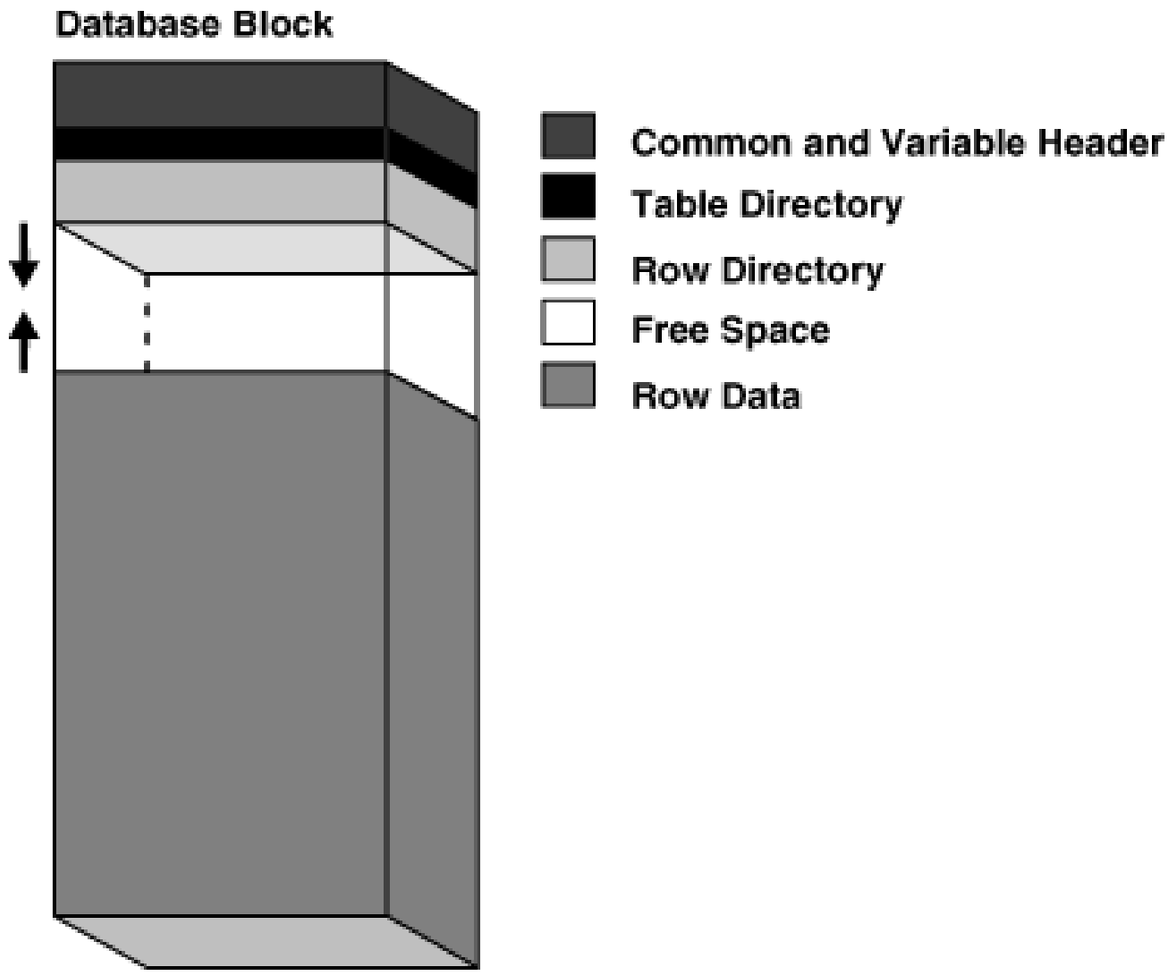
- Oracle allocates space for segment in units of extents.
- When extent is full, oracle allocates another extent.



Data Storage – Blocks

Illustration a Data Blocks

- Figure shows a **data blocks** is divided into several sections.
- Some of header and directory information used by the system.
- Some for free space to accommodate UPDATES or expansion of data
- Section that contains the table's row data.



Data Storage – Blocks & Performance

Block Free Space Allocation

- DBMS allow DB Admin to specify Size of block & percent of free space in block:
  - Target is to have blocks full to store more rows, but must also have enough free-space for updates as needed.
  - Compromise is required between having block full and risk of having no free-space for updates.
  - Choice of datatype can help: ex. VARCHAR (variable length) results in more rows per block than CHAR
- Oracle allows different block size for different tablespaces:
  - Selecting the right size of a block can help performance.

Data Storage – Blocks & Performance

- **Consideration when choosing a block size:**

	Small blocks	Large blocks
Advantages	• Good for small rows with intensive random access as it reduces block contention (typical for Online Transaction Processing databases).	• Has lower overhead of the header; there is more room to store data. • Permits reading more rows into the buffer[4] with a single read/write (depending on row size and block size). • Good for very large rows.
Disadvantages	• Has relatively large space overhead due to the block header. • Not recommended for large rows. There might only be a few rows stored for each block; the change of migrating rows is significant.	• Wastes space in the buffer when accessing small rows. For example, with an 8 KB block size and 50 byte row size, 7,950 bytes loaded in the buffer are not used.

Data Storage – Tablespace

- **Illustration of a Segments, Extents & Data Blocks in Malyuta & Satyanarayana's book**
  - Figure shows table Title is stored in **tablespace** USER_DATA that expands two files df1.dbf & df2.dbf.
  - We can assume one **segment** divided into 4 **extents**.
  - Two extents in each file
  - Each extent is divided into **data blocks** where table data is stored.

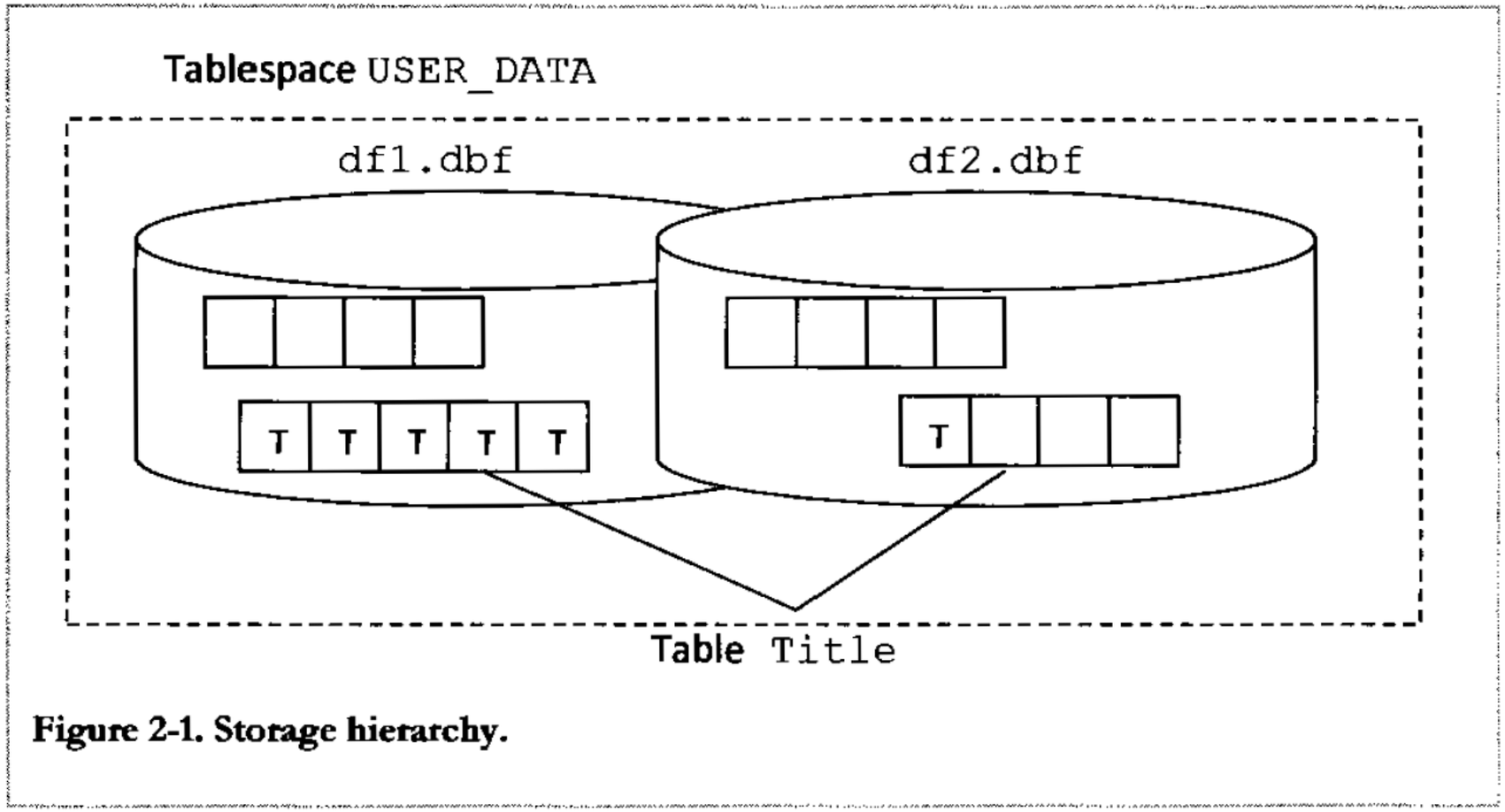


Figure 2-1. Storage hierarchy.

Data Storage – Tablespace Design Guidance

❑ Guidance from Book for physical design consideration

- Improve Performance:
  - If tablespace contains multiple files, store data files in different physical hard disks:
    - Reduce read/write operations

- Ease of management:
  - Store data for each application in separate tablespaces.
  - If maintenance is required on a table space, it only affects one application if tablespace taken off-line
  - You can backup each tablespace separately since most DBMS allow backup of individual table space

Data Storage Summary

Data Storage Summary

- **Segment –** Logical units that make up a tablespace. A **tablespace** is divided into segments.
- **Extent -** Segments are divided into a sequence of section of disk space called extents.
- **Data block –** Smallest unit of storage within an extent & where data is stored
- **Space Allocation**
  - Oracle allocates space for a segment in units of **extents**.
  - When extent is full, oracle allocates another extent.