Physical Database Design

- Purpose
  - Translate the logical description of data into the *technical specifications* for storing and retrieving data
  - How the logical structure is to be physically implemented in the target DBMS
  - Create a design for storing data that will provide ***adequate performance***

Physical Database Design

- The process of producing a description of the implementation of the database on secondary storage; it describes the ***base relations, file Organizations, and indexes*** used to achieve efficient access to the data

Physical Database Design

- The steps of the physical database design methodology are as follows:
  - Step 3 Translate logical data model for target DBMS
  - Step 4 Design file organizations and indexes

Basic Concepts

- The database on secondary storage is organized into one or more **files**, where each file consists of one or more **records** and each record consists of one or more **fields**
- Consider the relation given on the next slide, when a user requests a tuple from the DBMS, for example Staff tuple SG37, the DBMS maps this **logical record** on to a **physical record** and retrieves the physical record into the DBMS **buffers** in primary storage using the operating system file access routines

Figure: Reduced Staff relation from *DreamHome* case study

staffNo	lName	position	branchNo
SL21	White	Manager	B005
SG37	Beech	Assistant	B003
SG14	Ford	Supervisor	B003
SA9	Howe	Assistant	B007
SG5	Brand	Manager	B003
SL41	Lee	Assistant	B005

Basic Concepts

- The **physical record** is the unit of transfer between disk and primary storage, and vice versa
- Generally, **a physical record consists of more than one logical records**, although depending on size, a logical record can correspond to one physical record
- It is even possible for a large logical record to span more than one physical record

Basic Concepts

- The terms **block** and **page** are sometimes used in place of **physical record**
- For example, the Staff tuples may be stored on two pages as shown in Figure on next slide
- **The order in which records are stored and accessed in the file is dependent on the *file organization***

Figure: Storage of Staff relation in pages

staffNo	lName	position	branchNo	Page
SL21	White	Manager	B005	
SG37	Beech	Assistant	B003	1
SG14	Ford	Supervisor	B003	
SA9	Howe	Assistant	B007	
SG5	Brand	Manager	B003	2
SL41	Lee	Assistant	B005	

File Organization

- Technique for physically arranging records of a file on secondary storage
- Factors for selecting file organization:
  - Fast data retrieval and throughput
  - Efficient storage space utilization
  - Protection from failure and data loss
  - Minimizing need for reorganization
  - Accommodating growth
  - Security from unauthorized use

File Organization

- The physical arrangement of data in a file into pages on secondary storage
- The main types of file organization are:
  - **Heap (unordered) files** Records are placed on disk in no particular order
  - **Sequential (ordered) files** Records are ordered by the value of a specified field
  - **Hash files** Records are placed on disk according to a hash function

Access Method

- Along with a file organization, there is a set of *access methods*
- **Access method:** The steps involved in storing and retrieving records from a file.
- Since some access methods can be applied only to certain file organizations (for example, we cannot apply an indexed access method to a file without an index), the terms file organization and access method are used interchangeably

Unordered (Heap)Files

- Simplest organization
- Records placed in same order inserted
- Linear search for retrieval
- Insertion efficient; retrieval not efficient
- Deletion process
  - Relevant page identified
  - Record marked as deleted
  - Page rewritten to disk
  - Deleted record space not reused → inefficient storage
- Best suited for bulk loading data; there is no overhead of calculating what page the record should go on
- Suitable when all of the records are to be retrieved in a typical scan

Ordered Files

- Ordered files
  - Aka sequential files
  - Sorted on field – ***ordering field***
  - If ordering field = key → ***ordering key***
  - Binary search for retrieval
  - Insertion and deletion problematic
    - Need to maintain order of records
  - Rarely used for database storage
  - Best if records are retrieved in some order or a ‘range’ of records are retrieved

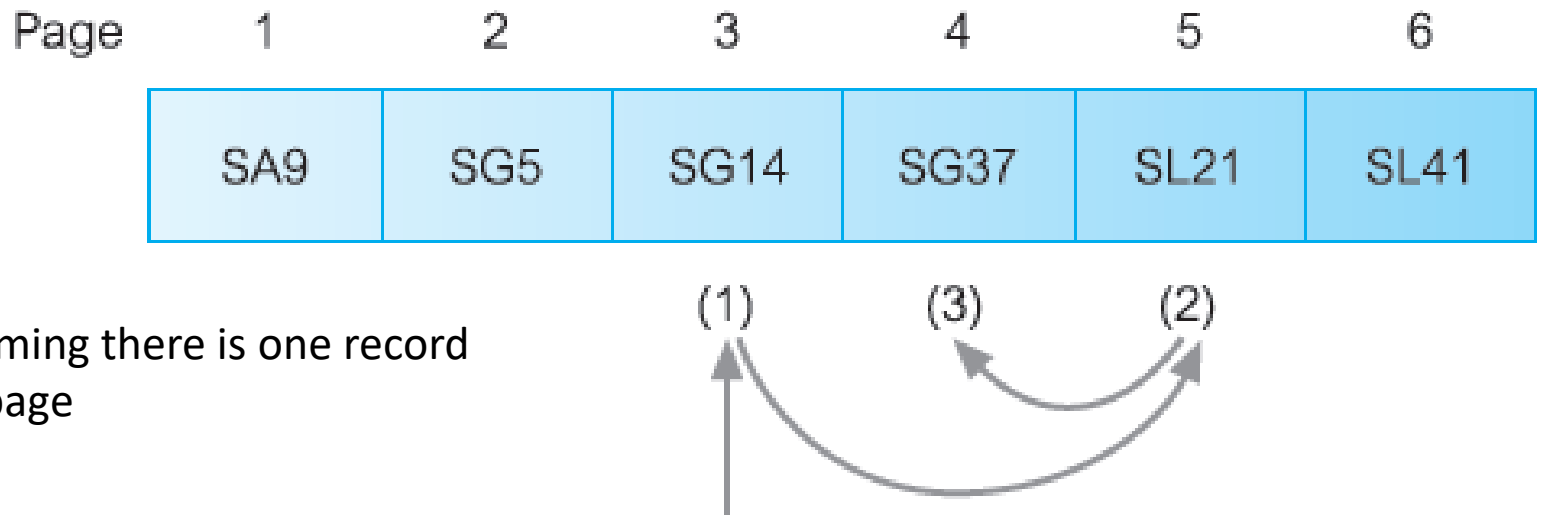
Ordered Files

- If the tuples are ordered on staffNo, under certain conditions we can use a **binary search** to execute queries that involve a search condition based on staffNo
- For example, consider the following SQL query:

```
SELECT *
FROM Staff
WHERE staffNo = 'SG37';
```

Binary search on an ordered file

staffNo	lName	position	branchNo
SL21	White	Manager	B005
SG37	Beech	Assistant	B003
SG14	Ford	Supervisor	B003
SA9	Howe	Assistant	B007
SG5	Brand	Manager	B003
SL41	Lee	Assistant	B005

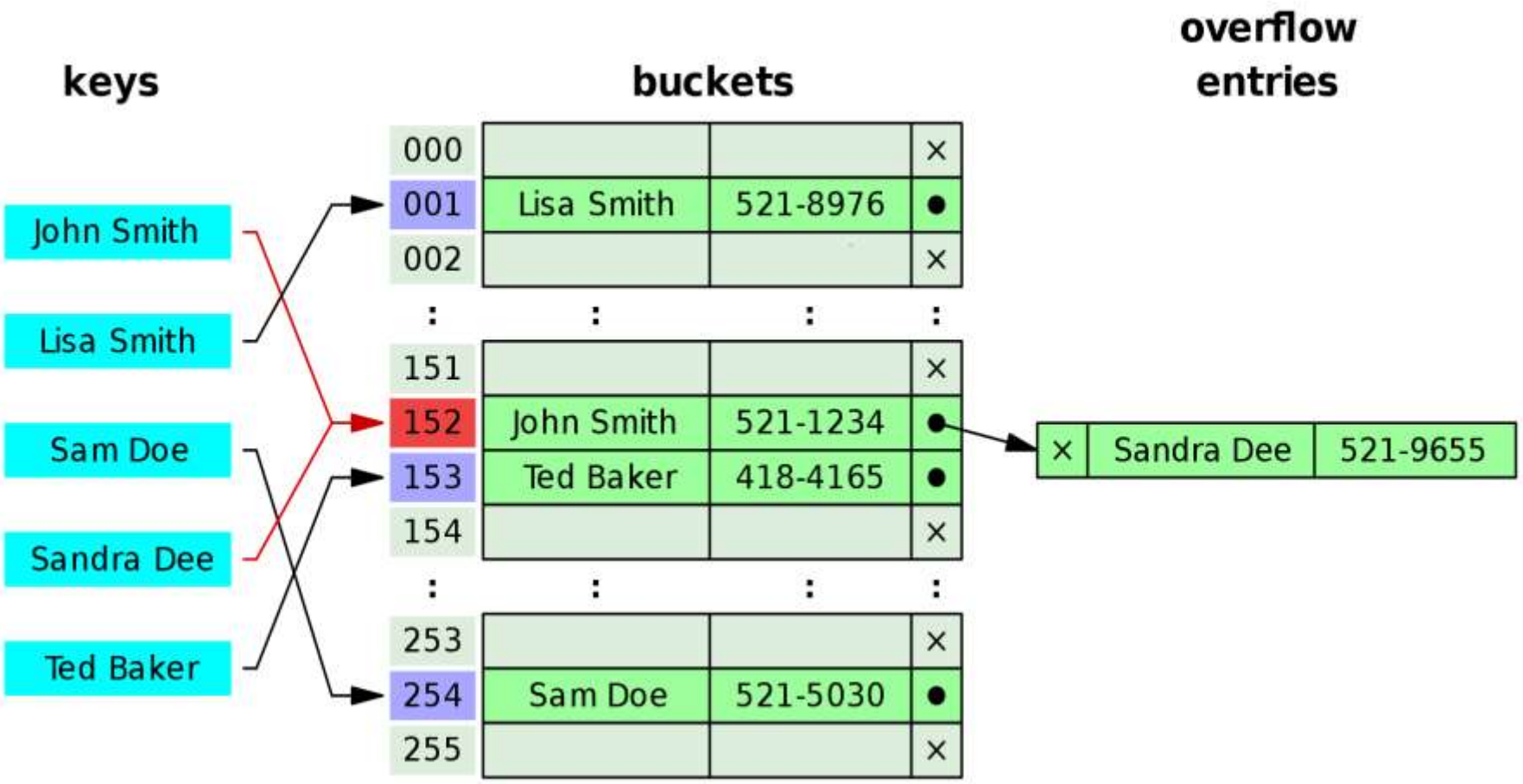


Assuming there is one record per page

Hash Files

- Hash files
  - Good for equality selections
  - Aka random/direct files
  - ***Hash function*** is used to determine the page address for storing record
    - Each address corresponds to a page/bucket
    - Each bucket has slots for multiple records
    - Hash function is chosen to provide most even distribution of records – min. collisions
    - Examples:
      - Division-remainder – uses mod function to determine field value
      - Folding – applying arithmetic function to hash field e.g. + 7
- Collision
  - Hash function calculates same address for 2 or more records

Hash Files


keys
buckets
overflow entries
John Smith
Lisa Smith
Sam Doe
Sandra Dee
Ted Baker
000
001
002
151
152
153
154
253
254
255
Lisa Smith
521-8976
John Smith
521-1234
Ted Baker
418-4165
Sam Doe
521-5030
Sandra Dee
521-9655

Indexes

- Data structure that allows DBMS to locate particular records in file more quickly
- Similar to index in book
- Main types of indexes:
  - Primary index
    - Index a key field
  - Clustering index
    - File sequentially ordered on non-key field i.e. more than one records can correspond with index

Step 3 Translate logical data model for target DBMS

- Extend the design of base relations (defined in logical data model) by identifying data types, lengths, and constraints that the target DBMS supports

```
    Domain PropertyNumber:          variable length character string, length 5
    Domain Street:                  variable length character string, length 25
    Domain City:                    variable length character string, length 15
    Domain Postcode:                variable length character string, length 8
    Domain PropertyType:            single character, must be one of 'B', 'C', 'D', 'E', 'F', 'H', 'M', 'S'
    Domain PropertyRooms:           integer, in the range 1–15
    Domain PropertyRent:            monetary value, in the range 0.00–9999.99
    Domain OwnerNumber:             variable length character string, length 5
    Domain StaffNumber:             variable length character string, length 5
    Domain BranchNumber:            fixed length character string, length 4
PropertyForRent(
    propertyNo      PropertyNumber      NOT NULL,
    street          Street              NOT NULL,
    city            City                NOT NULL,
    postcode        Postcode,
    type            PropertyType        NOT NULL DEFAULT 'F',
    rooms           PropertyRooms       NOT NULL DEFAULT 4,
    rent            PropertyRent        NOT NULL DEFAULT 600,
    ownerNo         OwnerNumber         NOT NULL,
    staffNo         StaffNumber,
    branchNo        BranchNumber        NOT NULL,
    PRIMARY KEY (propertyNo),
    FOREIGN KEY (staffNo) REFERENCES Staff(staffNo) ON UPDATE CASCADE ON DELETE SET NULL,
    FOREIGN KEY (ownerNo) REFERENCES PrivateOwner(ownerNo) and BusinessOwner(ownerNo)
                                    ON UPDATE CASCADE ON DELETE NO ACTION,
    FOREIGN KEY (branchNo) REFERENCES Branch(branchNo)
                                    ON UPDATE CASCADE ON DELETE NO ACTION);
```

Figure 18.1 DBDL for the PropertyForRent relation.

Step 4 Design file organizations and indexes

- The choice of file organizations should be fully documented, along with the reasons for the choice
- The choice of indexes should be fully documented along with the reasons for the choice