Part 2 – Design Documentation

This section presents a detailed overview of the low-level architecture of the database system. The design emphasizes efficient data storage, organization, and retrieval on disk, adhering to principles of modularity and scalability.

The system employs a page-based storage model, wherein each data type is allocated a dedicated file composed of sequential pages. Records within these pages are of fixed length, enabling predictable access patterns and simplifying memory management. Slot allocation within each page is managed by a bitmap, which facilitates rapid identification of free and occupied slots, thereby optimizing insertion and deletion operations.

Metadata regarding type definitions, field layouts, and primary key constraints is centrally maintained in a system catalog. This catalog-driven approach ensures consistency and integrity across all data operations, supporting both extensibility and robust schema enforcement.

Collectively, these design choices provide a solid foundation for reliable, high-performance data management in the implemented database system.

1 2.1 Type-Level File Structure

In the student database system, each **type** (corresponding to a relation or table) is managed independently and stored in its own binary file. The file is named using the convention `<type_name>.db`, where `<type_name>` is the name of the relation.

File Organization

- Each type-level file consists of multiple **pages** stored sequentially within the file.
- The system does *not* load all pages into memory at once; instead, it accesses pages as needed, supporting efficient memory usage even for large files.
- Pages are appended to the file as the number of records grows.

Page Structure Each page within a type file has a fixed structure:

- **Bitmap:** At the beginning of each page, a bitmap is used to indicate which record slots are occupied and which are free.
- **Records:** Following the bitmap, the page contains up to 10 fixed-size record slots. Each slot can store one record of the type.

Record Insertion Policy

- When a new record is inserted, the system searches for the first page with available space (i.e., a free slot as indicated by the bitmap).
- The record is inserted into the first available slot in that page.
- If all existing pages are full, a new page is created and appended to the file.

Summary

- Each type is stored in a separate binary file named `<type_name>.db`.
- Files are composed of sequential pages, each with a bitmap and up to 10 records.
- The system efficiently manages memory by loading only the required pages.
- Record insertion always targets the first available slot in the earliest page with space.

2 2.2 Slotted Page Format with Bitmap

Each page in the system is organized using an **unpacked slotted page format** to efficiently manage fixed-length records and support fast insertion and deletion.

Page Layout

Each page consists of the following components, stored sequentially:

- **Page Header:** Stores metadata such as:
  - `Page ID` – Unique identifier for the page
  - `Record Count` – Number of records currently stored in the page
  - `Free Space Info` – Information about available slots
- **Bitmap:** A 10-bit array, where each bit corresponds to a record slot:
  - `1` indicates the slot is occupied
  - `0` indicates the slot is free
- **Record Slots:** 10 fixed-length slots, each capable of storing one record.

Page Structure Diagram

```
+-------------------+-------------------+------------------------------+
|   Page Header     |      Bitmap       |      Record Slots (10x)      |
+-------------------+-------------------+------------------------------+
| Page ID           |  [b0 b1 ... b9]   |  Slot 0 | Slot 1 | ... | Slot 9 |
| Record Count      |                   |  (fixed size per record)     |
| Free Space Info   |                   |                              |
+-------------------+-------------------+------------------------------+
```

Bitmap Usage

- The bitmap is used to track the occupancy of each record slot.
- Each bit b_i (where $i = 0, 1, ..., 9$) corresponds to slot i:
  - $b_i = 1$ if slot i is occupied
  - $b_i = 0$ if slot i is free

Record Insertion

- To insert a record:
  - Scan the bitmap for the first 0 bit (free slot).
  - Place the new record in the corresponding slot.
  - Set the bitmap bit to 1 and increment the record count in the header.

Record Deletion

- To delete a record:
  - Locate the slot of the record to be deleted.
  - Set the corresponding bitmap bit to 0.
  - Decrement the record count in the header.
  - The slot is now available for future insertions.

Summary

- The slotted page format with bitmap enables efficient tracking of record occupancy.
- Insertions and deletions are performed in constant time by updating the bitmap and header.
- The fixed-length slots and bitmap structure simplify page management and free space tracking.

3 2.3 Record and Field Structure

Each record in the system is designed for efficient storage and access, using a fixed-size layout. The structure of a record is as follows:

- **Validity Flag (1 byte):** Indicates whether the record is valid (`1`) or has been deleted (`0`).
- **Fixed-Length Fields:**
  - **Integer fields:** Each uses 4 bytes.
  - **String fields:** Each is padded to a fixed size (e.g., 20 bytes) regardless of actual string length.
- **Field Layout:** The order and types of fields for each type are defined in the system catalog.
- **Maximum Fields:** Each type can have up to 6 fields.
- **Fixed Record Size:** The total size of a record is fixed and known at type creation time.

Sample Record Layout

Validity	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
1 byte	4/20 bytes	4/20 bytes	4/20 bytes	4/20 bytes	4/20 bytes	4/20 bytes

Table 1: Record Layout: Each field is either a 4-byte integer or a 20-byte padded string.

Notes:

- The actual number of fields per record depends on the type definition (up to 6).
- Unused fields (if any) may be left empty or zero-padded.
- The fixed record size allows for efficient slot management and direct access.

4 2.4 System Catalog Structure

The system catalog is a central metadata file (e.g., `system_catalog.json`) that stores the definitions of all types (relations) in the database. This catalog enables the system to manage and validate records according to their type definitions.

Catalog Contents

For each type, the catalog stores the following information:

- **Type Name:** A string identifier for the type (maximum 12 characters).
- **Fields:** A list of field definitions, where each field includes:
  - **Name:** Field name (maximum 20 characters)
  - **Type:** Field type, either `str` (string) or `int` (integer)
- **Primary Key Index:** The 0-based index of the primary key field in the field list.

Catalog Management

- The system reads the catalog file at startup to load all existing type definitions into memory.
- When a new type is created, the system updates the catalog file to include the new type's definition.
- The catalog ensures that all type and field constraints are enforced consistently across the database.

Sample JSON Catalog Entry

```
{
  "types": [
    {
      "type_name": "Student",
      "fields": [
        {"name": "id", "type": "int"},
        {"name": "name", "type": "str"},
        {"name": "year", "type": "int"}
      ],
      "primary_key_index": 0
    }
  ]
}
```