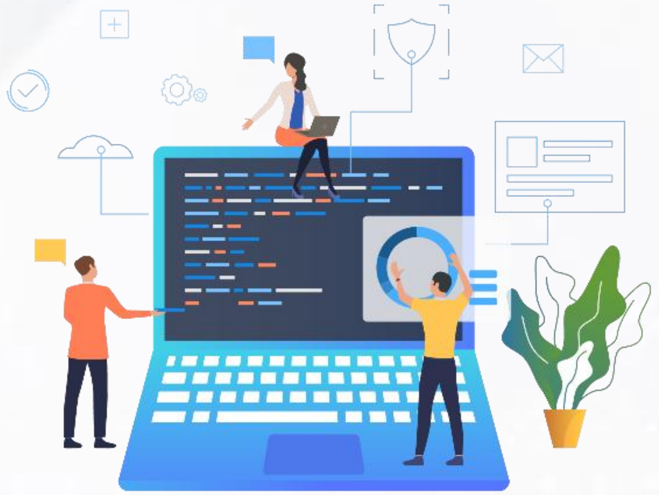
Unit-3

Relational Model & File Organization



CSE(Cyber Security), NRCM

Department of CSE(Cyber Security)
Narsimha Reddy Engineering College (NRCM)

Relational Model & File Organization

Relational Algebra, Calculus, Indexing & Hashing

Course: Database Management Systems

Unit III: Relational Model & File Organization

Contents Overview

1 Relational Model
Introduction, Structure, Schema, Algebra, Calculus

2 File Organization & Indexing
File Types, Indexes, B+ Tree, Hashing

This unit covers the **Relational Model** proposed by E.F. Codd, including its structure, schema design, and query languages (Relational Algebra and Calculus). It also explores **File Organization** techniques and **Indexing** strategies including B+ Trees and Hash-based methods for efficient data retrieval.

Part 1 Topics: Introduction to Relational Model · Basic Structure · Database Schema · Relational Algebra · Relational Calculus

Part 2 Topics: File Organization Introduction · Types of File Organizations · Overview of Indexes · Types of Indexes · Index Data Structures · Tree Structured Indexing · Hash Based Indexing

PART 01

Introduction, Structure, Schema, Algebra & Calculus

Relational Model

The mathematical foundation of modern database systems — relations, keys, and query languages

** This chapter covers the theoretical foundation of relational databases proposed by E.F. Codd in 1970*

Introduction to Relational Model

- **Origin**: Proposed by E.F. Codd in 1970 at IBM Research
- **Foundation**: Based on mathematical relations (tables)
- **Key Advantage**: Simplicity, independence, strong theory
- **Structure**: Relation = set of tuples with attributes
- **Query Languages**: Algebra & Calculus

Core Idea: Data organized as tables with formal query languages based on mathematical precision

Why Relational Model?

Simplicity: Tabular structure is intuitive

Independence: Logical and physical data independence

Flexibility: Ad-hoc queries without programming

Standardization: SQL is the universal standard

Maturity: 50+ years of proven reliability

Adoption: Oracle, MySQL, PostgreSQL, SQL Server

Basic Structure of Relational Model

Relation (Table)

A 2D structure of rows and columns representing an entity set

Tuple (Row)

A single record in a relation; one instance of the entity

Attribute (Column)

A named column representing a property of the entity

Additional Key Concepts

Domain: Set of permitted values for an attribute

Degree: Number of attributes (columns) in a relation

Cardinality: Number of tuples (rows) in a relation

Schema: R(A1, A2, ..., An) — logical structure

Primary Key: Unique identifier for each tuple

Foreign Key: References primary key of another relation

Candidate Key: Minimal set of attributes uniquely identifying tuples

Instance: Set of tuples at a given point in time

Database Schema

Three Levels of Schema

External (View Level): User-specific views that hide complexity and show only relevant data

Conceptual (Logical Level): Describes the complete database structure, relations, constraints, and security

Internal (Physical Level): Defines physical storage — file organization, indexes, compression

Data Independence

Logical Independence: Change conceptual schema without affecting external schemas or applications

Physical Independence: Change internal schema (storage, indexes) without affecting conceptual schema

Integrity Constraints

Constraint Type	Description	Example
Entity Integrity	Primary key cannot be NULL	EmpID in Employee table
Referential Integrity	Foreign key must match a primary key or be NULL	DeptID references Dept table
Domain Constraint	Attribute values must be from declared domain	Salary must be integer > 0

Relational Algebra — Fundamental Operations

Unary Operations

SELECT (σ): Selects tuples satisfying a predicate — $\sigma_{condition}(R)$

PROJECT (π): Selects specific attributes — $\pi_{attr\_list}(R)$

RENAME (ρ): Renames relations/attributes — $\rho_{new}(old)$

Set Operations

UNION (∪): All tuples in R or S or both

INTERSECTION (∩): Tuples common to R and S

DIFFERENCE (−): Tuples in R but not in S

CARTESIAN PRODUCT (×): All combinations of R and S tuples

Binary Operations — JOIN Types

Join Type	Description	Notation
Natural Join	Equates common attributes, removes duplicates	$R \bowtie S$
Equi-Join	Join on equality condition (keeps both attributes)	$R \bowtie_{R.A=S.B} S$
Theta Join	Join on general condition (not just equality)	$R \bowtie_{\theta} S$
Division (÷)	Finds tuples in R associated with ALL tuples in S	$R \div S$

Relational Calculus

Tuple Relational Calculus (TRC)

Variables range over tuples

Format: $\{ t \mid P(t) \}$

Example: Find employees with salary > 50,000

$\{ t \mid t \in Employee \land t.salary > 50000 \}$

Uses $\exists$ (exists) and $\forall$ (for all) quantifiers

Domain Relational Calculus (DRC)

Variables range over domain values

Format: $\{ <x1, x2> \mid P(x1, x2) \}$

Example: Find employee IDs and names

$\{ <eid, name> \mid \exists s(<eid, name, s> \in Emp \land s > 50000) \}$

Variables represent domain values, not tuples

Algebra vs Calculus vs SQL

Relational Algebra is **procedural** — specifies HOW to retrieve data (sequence of operations)

Relational Calculus is **declarative** — specifies WHAT to retrieve (conditions on result)

Both are **relationally complete** — equivalent expressive power. SQL combines both: SELECT-FROM-WHERE from Calculus, operations from Algebra

PART 02

File Types, Indexes, B+ Tree & Hashing

File Organization & Indexing

Storage structures, access methods, and index data structures for efficient data retrieval

** This chapter covers physical database design — how data is stored and accessed efficiently on disk*

Introduction to File Organization

- **File**: Collection of related records on secondary storage
- **Goals**: Fast access, efficient storage, easy maintenance
- **Records**: Fixed-length vs Variable-length
- **Buffer**: Data transferred in blocks between disk and memory
- **Blocking Factor**: Records per block; affects I/O efficiency

Key Trade-off: Access time vs Storage space vs Maintenance cost vs Reorganization frequency

File Operations

Create: Allocate space for a new file

Open/Close: Prepare file for operations

Read: Retrieve records from file

Write: Insert or update records

Seek: Position file pointer to a location

Delete: Remove file or records

Types of File Organizations

Type	Structure	Search	Insertion	Best For
Heap	Records in insertion order; no sorting	Linear scan O(n)	Very Fast (append)	Small files, temp data
Sequential	Sorted on ordering key field	Binary O(log n)	Expensive (may shift)	Range queries, reports
Hash	Hash function maps key to bucket	Direct O(1)	Fast with collision handling	Equality searches
Clustered	Related records stored together	Varies by cluster key	Moderate	Join operations

Table: Comparison of File Organization Types — access patterns and performance characteristics

Overview of Indexes

What is an Index?

A **data structure** that speeds up data retrieval on a database table

Analogy: Like a **book index** — maps search key to record location

Structure: **Search Key** + **Data Pointer** (Record ID / Block Pointer)

When to Use / Avoid

Use for: Large tables, WHERE clause columns, join columns, high-selectivity columns

Avoid for: Small tables, frequently updated columns, low-selectivity columns

Index Evaluation Metrics

Metric	Description
Access Time	Time to find a data item using the index
Insertion Time	Time to insert data + update the index structure
Deletion Time	Time to delete data + update the index structure
Space Overhead	Additional storage required for the index data structure

Types of Indexes

Primary Index

Built on the ordering key field of a sequentially ordered file. One primary index per file.

Dense: Entry for every record. Sparse: Entry for some records.

Secondary Index

Built on a non-ordering field. Multiple secondary indexes allowed per file.

Provides alternative access paths. Always dense (entry for every record).

Clustering Index

Built on ordering non-key field. Records with same value are physically grouped together.

One clustering index per file. Improves range query performance.

Multi-Level & Composite

Multi-level: Index on index (reduces search space). Composite: Index on multiple columns.

Also: Ordered (tree-based, range queries) vs Unordered (hash-based, equality).

Index Data Structures

Tree-Structured Indexes

Ordered — Support range queries

Structure: Nodes contain key-pointer pairs

Balance: Height-balanced for consistent performance

Leaf Links: Linked for efficient range scans

Examples: B-Tree, B+ Tree, ISAM

Hash-Based Indexes

Unordered — Support equality queries

Hash Function: h(k) maps key to bucket address

Access: Direct access to data bucket O(1)

Collisions: Multiple keys → same bucket (handled)

Examples: Static, Extendible, Linear Hashing

Selection Criteria

Criteria	Use Tree-Based	Use Hash-Based
Query Type	Range queries, sorted output	Equality queries only
Data Characteristics	Dynamic, frequently changing	Static or predictable size
Performance Priority	Consistent query performance	Minimal space overhead
Examples	B+ Tree in MySQL, PostgreSQL	Hash indexes in Oracle

Tree Structured Indexing — B+ Tree

B+ Tree Properties

Order (p): Max pointers per internal node

Balance: All leaf nodes at same level

Fill Factor: Each node ≥ ⌈ p/2 ⌉ children

Leaf Links: Linked for range scans

Data: All data pointers in leaves only

B+ Tree Operations

Search: Traverse from root to leaf O(log n)

Insert: May cause node split; propagate up

Delete: May cause merge/redistribution

Range Query: Efficient via leaf links

Equality: Direct root-to-leaf path

Why B+ Tree is Dominant

Balanced Height

Guaranteed O(log n) search, insert, delete regardless of data distribution

Range Queries

Linked leaf nodes enable efficient sequential access and range scans

Dynamic Adaptability

Self-adjusts to insertions and deletions without requiring full reorganization

Hash Based Indexing

Extendible Hashing

Uses directory of bucket pointers

Global Depth (gd): Bits used from hash value for directory

Local Depth (ld): Bits each bucket actually uses

Split: When overflow and ld == gd, directory doubles

Advantage: No overflow chains, graceful growth

Linear Hashing

No directory needed; round-robin splits

Level (L): Current round number for hash function

Split Pointer (n): Next bucket to split when overflow

Split: Round-robin, one bucket at a time

Advantage: No directory overhead, incremental

Extendible vs Linear Hashing

Aspect	Extendible Hashing	Linear Hashing
Structure	Directory of bucket pointers	No directory; split pointer
Access Speed	Faster (direct directory lookup)	Slower (may need chain traversal)
Space Overhead	Directory can grow large	No directory overhead
Growth Pattern	Only split overflowing bucket	Round-robin incremental splits

Thank You

UNIT III Summary

Relational Model & File Organization

Relational Model · Relational Algebra · Relational Calculus · File Organization · Indexing