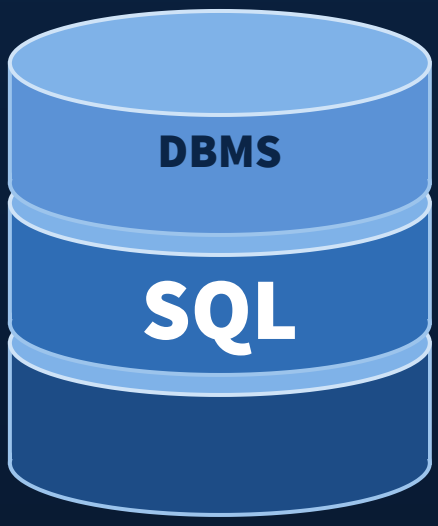
369 TESLA | KALVIYOGI

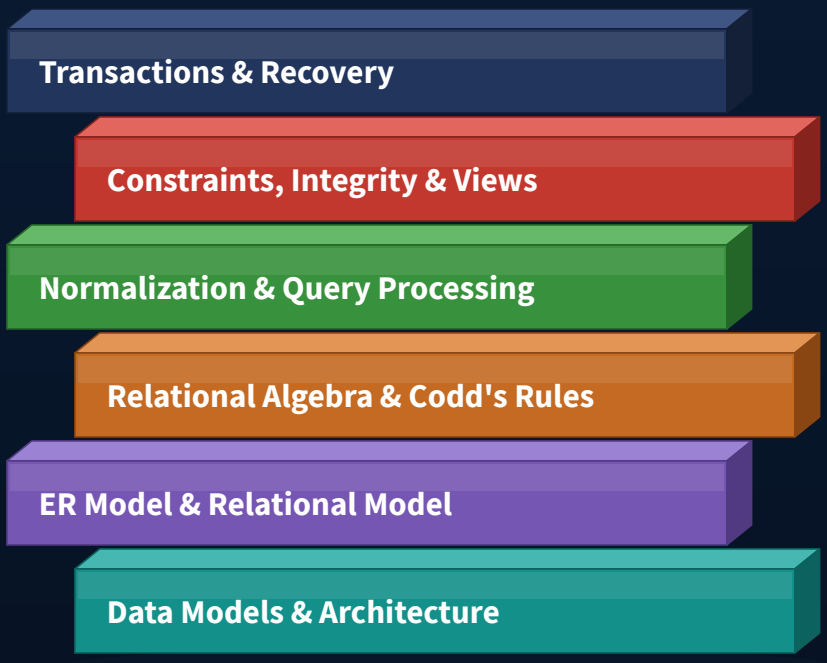
UNIT II

DATABASE MANAGEMENT SYSTEM

Visual notes
Real SQL screens (SQLite)
3D infographics
TRB exam theory




```
CREATE TABLE Student (
RollNo INT PRIMARY KEY,
DeptID INT REFERENCES Dept
);
SELECT Name FROM Student
WHERE CGPA > 8;
π Name (σ CGPA>8 (Student))
A → B, B → C ⊢ A → C
BEGIN; ... COMMIT;
GRANT SELECT ON Student
R1 ⋈ R2
LOCK-S(A) LOCK-X(B)
```



Kalviyogi Nagarajan

EDUNIVRA PVT LTD

UNIT II

DATABASE MANAGEMENT SYSTEM

Complete Visual Notes with Real SQL Screens, Result Windows, 3D Infographics and TRB Exam Points

Kalviyogi Nagarajan

EDUNIVRA PVT LTD

Database Management System — Unit II

Kalviyogi Nagarajan · 369 TESLA | KALVIYOGI

Published by EDUNIVRA PVT LTD.

This edition is a redesigned and edited version of the author's Unit II notes (Database Concepts and Architecture · Data Modeling · Normalization and Query Processing · Constraints, Integrity and Views · Transactions, Concurrency Control and Recovery). Every SQL screen in this book was run on SQLite 3.50 with foreign keys switched on; the result windows show the real rows and the real error messages returned by the database. Statements that only work on one product (Oracle, MySQL, SQL Server) are shown as code and marked with the product name.

Technical statements were checked and corrected where needed; each correction is marked as a **Common mistake** box so that you know which version the examiner expects. The figures are redrawn as vector 3D illustrations. Every chapter ends with a summary, and the last pages give a one-page revision of each part.

Contents

Database System Concepts and Architecture 14

Chapter 1 · Data Models 15
1.1 What Is a Data Model? 15
1.2 Purpose of Data Models 16
1.3 Components of a Data Model 17
1.4 Categories of Data Models 18
1.5 Conceptual Data Models 19
1.6 Logical Data Models 19
1.7 Physical Data Models 20
1.8 Relational Data Model 21
1.9 Hierarchical Data Model 22
1.10 Network Data Model 23
1.11 Object-Oriented Data Model 24
1.12 Object-Relational Data Model 25
1.13 Semi-Structured Data Model 26
1.14 NoSQL Data Models 27
1.15 Comparison of Data Models 28

Chapter 2 · Schemas, Instances and Metadata 30
2.1 Meaning of Database Schema 30
2.2 Schema Definition 31
2.3 Schema Diagram 32
2.4 External Schema 33
2.5 Conceptual Schema 33
2.6 Internal Schema 34
2.7 Logical Schema 34
2.8 Physical Schema 35
2.9 Meaning of Database Instance 36
2.10 Database State 36
2.11 Initial Database State 38
2.12 Current Database State 38
2.13 Valid Database State 39
2.14 Schema versus Instance 40
2.15 Intension and Extension 40
2.16 Metadata 41
2.17 Data Dictionary 41
2.18 System Catalog 42

Chapter 3 · Three-Schema Architecture 45
3.1 Three-Schema Architecture 45
3.2 Introduction and ANSI/SPARC Origin 47
3.3 Need for Three-Schema Architecture 47
3.4 External Level 48
3.5 External Schema 48
3.6 User Views and Subschemas 49
3.7 Conceptual Level 50
3.8 Global Logical View 51
3.9 Internal Level 51
3.10 Internal Schema 52
3.11 Physical Storage Structure 53

3.12 External–Conceptual Mapping 53
3.13 Conceptual–Internal Mapping 55
3.14 Data Abstraction 56
3.15 View-Level Abstraction 56
3.16 Logical-Level Abstraction 57
3.17 Physical-Level Abstraction 57
3.18 Advantages of Three-Schema Architecture 58
3.19 Limitations of Three-Schema Architecture 59

Chapter 4 · Data Independence 60

4.1 Data Independence 60
4.2 Meaning of Data Independence 61
4.3 Need for Data Independence 62
4.4 Program–Data Independence 62
4.5 Logical Data Independence 63
4.6 Changes in the Conceptual Schema 64
4.7 Examples of Logical Data Independence 65
4.8 Physical Data Independence 65
4.9 Changes in Storage Structures 66
4.10 Changes in File Organization 67
4.11 Changes in Indexes and Access Paths 68
4.12 Examples of Physical Data Independence 69
4.13 Logical versus Physical Data Independence 69
4.14 Advantages of Data Independence 70
4.15 Limitations in Achieving Data Independence 70

Chapter 5 · Database Languages 72

5.1 Database Languages and Interfaces 72
5.2 Introduction to Database Languages 73
5.3 Data Definition Language (DDL) 74
5.4 Data Manipulation Language (DML) 75
5.5 Data Query Language (DQL) 76
5.6 Data Control Language (DCL) 77
5.7 Transaction Control Language (TCL) 78
5.8 Storage Definition Language (SDL) 78
5.9 View Definition Language (VDL) 79
5.10 Procedural DML 80
5.11 Non-Procedural DML 81
5.12 Embedded SQL 82
5.13 Dynamic SQL 83

Chapter 6 · Database Interfaces 85

6.1 Command-Line Interfaces 86
6.2 Menu-Based Interfaces 86
6.3 Form-Based Interfaces 87
6.4 Graphical User Interfaces 88
6.5 Query-by-Example Interfaces 89
6.6 Natural-Language Interfaces 89
6.7 Web-Based Interfaces 90
6.8 Mobile Database Interfaces 91
6.9 Application Programming Interfaces 91
6.10 ODBC (Open Database Connectivity) 92
6.11 JDBC (Java Database Connectivity) 93
6.12 Interfaces for Database Administrators 94

6.13 Interfaces for Application Programmers 95
6.14 Interfaces for End Users 96

Chapter 7 · Centralized and Client–Server Architectures 97
7.1 Database-System Architecture 97
7.2 Centralized Database Architecture 99
7.3 Components of a Centralized DBMS 99
7.4 Working of a Centralized Database System 99
7.5 Advantages of Centralized Architecture 100
7.6 Limitations of Centralized Architecture 101
7.7 Client–Server Database Architecture 101
7.8 Client Components 102
7.9 Database Server Components 103
7.10 Request–Response Processing 103
7.11 One-Tier Architecture 104
7.12 Two-Tier Client–Server Architecture 104
7.13 Three-Tier Client–Server Architecture 105
7.14 N-Tier Architecture 106
7.15 Presentation Layer 106
7.16 Application or Business-Logic Layer 107
7.17 Database Layer 108
7.18 Middleware 108
7.19 Fat Client and Thin Client 109
7.20 Advantages of Client–Server Architecture 110
7.21 Limitations of Client–Server Architecture 110
7.22 Centralized versus Client–Server Architecture 111
7.23 Two-Tier versus Three-Tier Architecture 111
7.24 Applications of Client–Server DBMS 112

Data Modeling and the Relational Model 113

Chapter 8 · Data Modeling, Entities and Attributes 114
8.1 Meaning of Data Modeling 114
8.2 Purpose of Data Modeling 115
8.3 Levels of Data Modeling 115
8.4 Conceptual, Logical and Physical Data Models 116
8.5 Meaning of Entity 116
8.6 Entity Type and Entity Set 117
8.7 Strong and Weak Entities 117
8.8 Attributes and Their Types 119
8.9 Simple and Composite Attributes 120
8.10 Single-Valued and Multi-Valued Attributes 120
8.11 Stored and Derived Attributes 121
8.12 Key Attributes 121

Chapter 9 · Relationships and ER Diagrams 123
9.1 Meaning of Relationship 123
9.2 Relationship Type and Relationship Set 123
9.3 Degree of a Relationship 124
9.4 Cardinality Ratios 126
9.5 One-to-One Relationship 127
9.6 One-to-Many Relationship 128
9.7 Many-to-Many Relationship 129
9.8 Participation Constraints 130

9.9 Total and Partial Participation 130
9.10 Role Names and Recursive Relationships 131
9.11 ER Diagram Symbols 132
9.12 Steps for Constructing an ER Diagram 133
9.13 ER Diagram Worked Examples 134

Chapter 10 · The Enhanced ER Model 136
10.1 Introduction to the Enhanced ER Model 136
10.2 Superclass and Subclass 137
10.3 Specialization 138
10.4 Generalization 139
10.5 Inheritance of Attributes and Relationships 140
10.6 Disjoint and Overlapping Constraints 140
10.7 Total and Partial Specialization 142
10.8 Category or Union Type 143
10.9 Aggregation 144
10.10 Enhanced ER Diagram Worked Examples 146

Chapter 11 · Relational Model Concepts 148
11.1 Introduction to the Relational Model 148
11.2 Relation or Table 149
11.3 Tuple or Row 150
11.4 Attribute or Column 150
11.5 Domain 151
11.6 Degree of a Relation 152
11.7 Cardinality of a Relation 152
11.8 Relation Schema 153
11.9 Relation Instance 153
11.10 Characteristics of Relations 154
11.11 Base Relations and Views 155
11.12 Advantages of the Relational Model 157
11.13 Limitations of the Relational Model 157

Chapter 12 · Keys 159
12.1 Meaning and Importance of Keys 159
12.2 Super Key 160
12.3 Candidate Key 161
12.4 Primary Key 161
12.5 Alternate Key 162
12.6 Composite Key 163
12.7 Foreign Key 164
12.8 Surrogate Key 165
12.9 All Keys at a Glance 166

Chapter 13 · Relational Schemas and ER-to-Relational Mapping 167
13.1 Meaning of a Relational Database Schema 167
13.2 Database State or Instance 168
13.3 Schema Diagram 169
13.4 Relation Schema Notation 169
13.5 Representation of Primary Keys 169
13.6 Representation of Foreign Keys 170
13.7 Mapping Entities into Relations 170
13.8 Mapping Weak Entities into Relations 171
13.9 Mapping One-to-One Relationships 171

13.10 Mapping One-to-Many Relationships 171
13.11 Mapping Many-to-Many Relationships 172
13.12 Mapping Multi-Valued Attributes 172
13.13 Mapping Ternary Relationships 173
13.14 Mapping Specialization and Generalization 173
13.15 Conversion of ER Diagrams into Relational Schemas 174

Chapter 14 · Database Languages in Practice 177
14.1 Introduction to Database Languages 177
14.2 Data Definition Language 177
14.3 Data Manipulation Language 178
14.4 Data Query Language 179
14.5 Data Control Language 180
14.6 Transaction Control Language 180
14.7 Embedded Database Languages 182
14.8 Interactive Database Languages 182
14.9 Database Programming Interfaces 183

Chapter 15 · Database Design Process and Application Programming 185
15.1 Database-Design Process 185
15.2 Requirements Collection and Analysis 186
15.3 Conceptual Database Design 187
15.4 Logical Database Design 187
15.5 Physical Database Design 188
15.6 Schema Refinement 189
15.7 Integrity-Constraint Design 190
15.8 View Design 191
15.9 Transaction Design 192
15.10 Database Application Programming 192
15.11 Host Languages and Embedded SQL 194
15.12 Database Connectivity Interfaces 194
15.13 Forms, Reports and User Interfaces 194
15.14 Database Testing and Maintenance 195

Chapter 16 · Relational Algebra 197
16.1 Introduction to Relational Algebra 197
16.2 Unary and Binary Operations 198
16.3 Selection (σ) 199
16.4 Projection (π) 199
16.5 Rename (ρ) 200
16.6 Union, Set Difference and Intersection 201
16.7 Cartesian Product (×) 202
16.8 Join Operation 203
16.9 Theta Join 203
16.10 Equi Join, Natural Join and Inner Join 204
16.11 Left, Right and Full Outer Joins 205
16.12 Semi Join and Anti Join 206
16.13 Division (÷) 207
16.14 Assignment (←) 209
16.15 Aggregate Functions and Grouping (γ) 209
16.16 Relational-Algebra Query Expressions 210
16.17 Worked Examples 211

Chapter 17 · Relational Calculus 214

17.1 Introduction to Relational Calculus 214
17.2 Relational Algebra versus Relational Calculus 215
17.3 Procedural and Declarative Query Languages 216
17.4 Tuple Relational Calculus 216
17.5 Tuple Variables 216
17.6 Atomic Formulas 217
17.7 Quantifiers and Logical Connectives 217
17.8 Free and Bound Variables 218
17.9 Tuple Relational Calculus Expressions 218
17.10 Domain Relational Calculus 219
17.11 Domain Variables 219
17.12 Domain Relational Calculus Expressions 219
17.13 Safe and Unsafe Expressions 220
17.14 Expressive Power of Relational Query Languages 221
17.15 Relational-Calculus Worked Examples 222

Chapter 18 · Codd's Relational Rules 224
18.1 Introduction to Codd's Rules 224
18.2 Rule 0 — Foundation Rule 226
18.3 Rule 1 — Information Rule 226
18.4 Rule 2 — Guaranteed Access Rule 226
18.5 Rule 3 — Systematic Treatment of Null Values 227
18.6 Rule 4 — Dynamic Online Catalogue 228
18.7 Rule 5 — Comprehensive Data Sublanguage 229
18.8 Rule 6 — View-Updating Rule 229
18.9 Rule 7 — High-Level Insert, Update and Delete 230
18.10 Rule 8 — Physical Data Independence 231
18.11 Rule 9 — Logical Data Independence 231
18.12 Rule 10 — Integrity Independence 232
18.13 Rule 11 — Distribution Independence 233
18.14 Rule 12 — Non-Subversion Rule 233
18.15 Importance of Codd's Rules 235
18.16 Limitations and Practical Implementation 235

Normalization, Query Processing and Advanced Topics 237

Chapter 19 · Functional Dependencies 238
19.1 Introduction to Functional Dependencies 238
19.2 Types of Functional Dependencies 239
19.3 Partial and Transitive Dependencies 240
19.4 Armstrong's Axioms 241
19.5 Additional Inference Rules 241
19.6 Attribute Closure and FD Closure 241
19.7 Candidate-Key Identification 242
19.8 Minimal Cover of Functional Dependencies 242

Chapter 20 · Normalization 245
20.1 Introduction to Database Normalization 245
20.2 Redundancy and Database Anomalies 246
20.3 First Normal Form (1NF) 247
20.4 Second Normal Form (2NF) 247
20.5 Third Normal Form (3NF) 248
20.6 Boyce–Codd Normal Form (BCNF) 248
20.7 Fourth Normal Form and Multivalued Dependencies 249

20.8 Fifth Normal Form and Join Dependencies 250
20.9 Lossless-Join Decomposition 251
20.10 Dependency Preservation 252
20.11 Normalization Algorithms 253
20.12 Complete Normalization Case Study 253

Chapter 21 · Query Processing and Optimization 256
21.1 Introduction to Query Processing 256
21.2 Query Parsing and Translation 257
21.3 Query Trees and Query Graphs 258
21.4 Query-Evaluation Plans 259
21.5 Selection and Projection Algorithms 259
21.6 Nested-Loop Join Algorithms 260
21.7 Sort-Merge and Hash Join Algorithms 261
21.8 Query-Optimization Techniques 264
21.9 Query-Rewriting Rules 265
21.10 Cost Estimation and Plan Selection 266

Chapter 22 · Transactions, Concurrency Control and Recovery 269
22.1 Introduction to Transaction Processing 269
22.2 ACID Properties 270
22.3 Transaction States 271
22.4 Transaction Schedules 271
22.5 Conflict Serializability 271
22.6 Recoverable Transaction Schedules 274
22.7 Concurrency Control and Concurrency Problems 275
22.8 Lock-Based Concurrency Control 277
22.9 Two-Phase Locking and its Strict Forms 277
22.10 Deadlock Management 278
22.11 Timestamp-Ordering Protocol 278
22.12 Optimistic and Multiversion Concurrency Control 279
22.13 Introduction to Database Recovery 279
22.14 Transaction Logs and Write-Ahead Logging 280
22.15 Deferred and Immediate Update 280
22.16 Checkpointing and Recovery 281
22.17 Shadow Paging and ARIES 283
22.18 Backup and Disaster Recovery 284

Chapter 23 · Object Databases and Database Security 285
23.1 Object-Oriented Databases 285
23.2 Object-Oriented Database Features 286
23.3 Object-Relational Databases 288
23.4 Comparison of Database Models 289
23.5 Database Security 290
23.6 Authorization and Access Control 292

Constraints, Integrity and Views 295

Chapter 24 · Database Constraints 296
24.1 Introduction to Database Constraints 296
24.2 Purpose and Importance of Constraints 297
24.3 Classification of Database Constraints 298
24.4 Column-Level and Table-Level Constraints 299
24.5 Constraint Declaration in SQL 300

24.6 Naming Database Constraints 301
24.7 Viewing and Managing Existing Constraints 301
24.8 Enabling, Disabling and Removing Constraints 302

Chapter 25 · NOT NULL, UNIQUE, PRIMARY KEY and DEFAULT 304
25.1 NOT NULL Constraint 304
25.2 NULL Values and Three-Valued Logic 305
25.3 UNIQUE Constraint 306
25.4 PRIMARY KEY Constraint 307
25.5 PRIMARY KEY versus UNIQUE 308
25.6 DEFAULT Constraint 309
25.7 AUTO-INCREMENT and Identity Columns 309
25.8 Candidate, Alternate and Super Keys 310
25.9 Comparison of Basic SQL Constraints 312

Chapter 26 · Integrity Constraints 313
26.1 Meaning of Database Integrity 313
26.2 Types of Database Integrity 314
26.3 Domain Integrity 315
26.4 Entity-Integrity Constraint 316
26.5 Key Constraints and Uniqueness 317
26.6 Semantic or Business-Rule Constraints 317
26.7 Integrity Enforcement by the DBMS 318
26.8 Declarative versus Procedural Integrity Rules 320

Chapter 27 · CHECK Constraints and Assertions 323
27.1 Introduction to CHECK Constraints 323
27.2 Column-Level CHECK Constraints 325
27.3 Table-Level CHECK Constraints 325
27.4 CHECK Constraints with Numeric Data 327
27.5 CHECK Constraints with Character Data 328
27.6 CHECK Constraints with Date and Time Values 329
27.7 CHECK Constraints with Logical Operators 330
27.8 Limitations of CHECK Constraints 331
27.9 Domain Objects and User-Defined Data Types 332
27.10 Assertions and Global Integrity Rules 333

Chapter 28 · Referential Integrity and Foreign Keys 337
28.1 Introduction to Referential Integrity 337
28.2 FOREIGN KEY Constraint 338
28.3 Parent Table and Child Table 340
28.4 Creating Foreign Keys during Table Creation 340
28.5 Adding Foreign Keys to Existing Tables 341
28.6 Composite Foreign Keys 342
28.7 Insert Operations and Referential Integrity 343
28.8 Update Operations and Referential Integrity 344
28.9 Delete Operations and Referential Integrity 345
28.10 ON DELETE CASCADE 346
28.11 ON DELETE SET NULL and SET DEFAULT 347
28.12 RESTRICT and NO ACTION Rules 348
28.13 ON UPDATE Referential Actions 349
28.14 Self-Referencing Foreign Keys 350
28.15 Circular and Multiple Referential Relationships 352
28.16 Referential-Integrity Errors and Solutions 353

Chapter 29 · Views and Unit Review 355
29.1 Introduction to Database Views 355
29.2 Creating and Displaying Simple Views 356
29.3 Complex Views 357
29.4 Advantages, Limitations and Applications of Views 359
29.5 Complete Unit Review: Constraints and Views 361

Transaction Management, Concurrency Control and Recovery 365

Chapter 30 · Transactions and ACID 366
30.1 Introduction to Transaction Management 366
30.2 Transaction Operations 367
30.3 Basic Structure of a Transaction 368
30.4 Examples of Database Transactions 369
30.5 Transaction States and the State Diagram 370
30.6 ACID Properties of Transactions 372
30.7 Atomicity and Consistency 372
30.8 Isolation and Durability 373
30.9 Transaction Manager and Storage Manager 375

Chapter 31 · Concurrency Problems 376
31.1 Introduction to Concurrency 376
31.2 Advantages of Concurrency 377
31.3 Problems Caused by Concurrency 377
31.4 Lost-Update Problem 378
31.5 Dirty-Read (Temporary-Update) Problem 380
31.6 Unrepeatable-Read Problem 381
31.7 Incorrect-Summary Problem 381
31.8 Phantom-Read Problem 382

Chapter 32 · Schedules and Serializability 385
32.1 What Is a Schedule? 385
32.2 Serial and Non-Serial Schedules 386
32.3 Serializability 386
32.4 Equivalent Schedules 386
32.5 Conflicting Operations 387
32.6 Conflict Serializability and the Precedence Graph 387
32.7 Worked Examples 388
32.8 View Serializability 390
32.9 Recoverable Schedules 392
32.10 Cascadeless Schedules and Cascading Rollback 392
32.11 Strict Schedules 392
32.12 Comparison of Schedule Types 394

Chapter 33 · Locking and Two-Phase Locking 396
33.1 Introduction to Locking 396
33.2 Binary Locks 396
33.3 Shared and Exclusive Locks 397
33.4 Lock Conversion 397
33.5 Locking Protocols 397
33.6 Two-Phase Locking (2PL) 398
33.7 Variants of 2PL 401
33.8 Multiple-Granularity Locking 402
33.9 Intention Locks 402

33.10 Lock Escalation and De-escalation 405

Chapter 34 · Deadlocks 406

34.1 Introduction to Deadlocks 406
34.2 Necessary Conditions for Deadlock 407
34.3 The Wait-For Graph 407
34.4 Deadlock Prevention 407
34.5 Wait-Die and Wound-Wait 408
34.6 Deadlock Avoidance 410
34.7 Deadlock Detection 411
34.8 Deadlock Recovery 411
34.9 Starvation and Its Prevention 413

Chapter 35 · Timestamp, Optimistic and Multiversion Control 415

35.1 Introduction to Timestamp Ordering 415
35.2 Read and Write Timestamps 415
35.3 The Basic Timestamp-Ordering Protocol 416
35.4 Thomas's Write Rule 418
35.5 Strict Timestamp Ordering 419
35.6 Timestamp Methods: Strengths, Weaknesses, and Locking 419
35.7 Optimistic Concurrency Control 419
35.8 Validation Rules 419
35.9 Multiversion Concurrency Control (MVCC) 422
35.10 Multiversion Timestamp Ordering (MVTO) 422
35.11 Snapshot Isolation 425
35.12 Isolation Levels in SQL 427
35.13 Isolation-Level Anomalies 427

Chapter 36 · Database Recovery 429

36.1 Introduction to Database Recovery 429
36.2 Types of Database Failures 430
36.3 The Recovery Manager 430
36.4 The Transaction Log 430
36.5 Write-Ahead Logging (WAL) 431
36.6 Deferred Database Modification 431
36.7 Immediate Database Modification 431
36.8 Undo and Redo Operations 432
36.9 Checkpoints 432
36.10 Recovery with Concurrent Transactions 435
36.11 Shadow Paging 435
36.12 The ARIES Recovery Method 436
36.13 Backup and Media Recovery 437
36.14 Disaster Recovery and Availability 440
36.15 Part V Review 440

PART I · UNIT II SYLLABUS 1

Database System Concepts and Architecture

Chapter 1 Data Models

Chapter 2 Schemas, Instances and Metadata

Chapter 3 Three-Schema Architecture

Chapter 4 Data Independence

Chapter 5 Database Languages

Chapter 6 Database Interfaces

Chapter 7 Centralized and Client–Server Architectures

Data models, schemas and instances · three-schema architecture · data independence · database languages and interfaces · centralized and client–server architectures for DBMS.

CHAPTER · UNIT II · DATABASE CONCEPTS

1 Data Models

OPENING SITUATION

The college library still keeps registers: one for books, one for members, one for loans. Finding "which CSE students hold overdue books" means turning pages in three registers. Before such a library can move to a database, someone must decide **what** will be stored, **how** the pieces are connected and **which rules** the data must obey. That plan is a **data model**. This chapter explains what a data model is, its levels, and the families of models from the 1960s tree model to today's NoSQL stores.

IN THIS CHAPTER

- 1.1 What Is a Data Model?
- 1.2 Purpose of Data Models
- 1.3 Components of a Data Model
- 1.4 Categories of Data Models
- 1.5 Conceptual Data Models
- 1.6 Logical Data Models
- 1.7 Physical Data Models
- 1.8 Relational Data Model
- 1.9 Hierarchical Data Model
- 1.10 Network Data Model
- 1.11 Object-Oriented Data Model
- 1.12 Object-Relational Data Model
- 1.13 Semi-Structured Data Model
- 1.14 NoSQL Data Models
- 1.15 Comparison of Data Models

YOU NEED

- The idea of a table with rows and columns.
- No database experience is assumed.

1.1 What Is a Data Model?

DEFINITION 1.1 — DATA MODEL

A **data model** is a collection of concepts used to describe the **structure** of a database (entities, attributes, relationships, data types), the **constraints** the data must satisfy and the **operations** that can be performed on it. It is the **blueprint** for organising real-world information so that it can be stored, retrieved and maintained efficiently.

Main idea. A database records facts about a small part of the real world, the **mini-world** (universe of discourse): a library, a college, a hospital. The data model gives a precise vocabulary for that mini-world. It provides **data abstraction**: it says *what* data is needed and *how* it relates, and hides *how* it is stored on disk. It is also the common language of users, designers and developers.

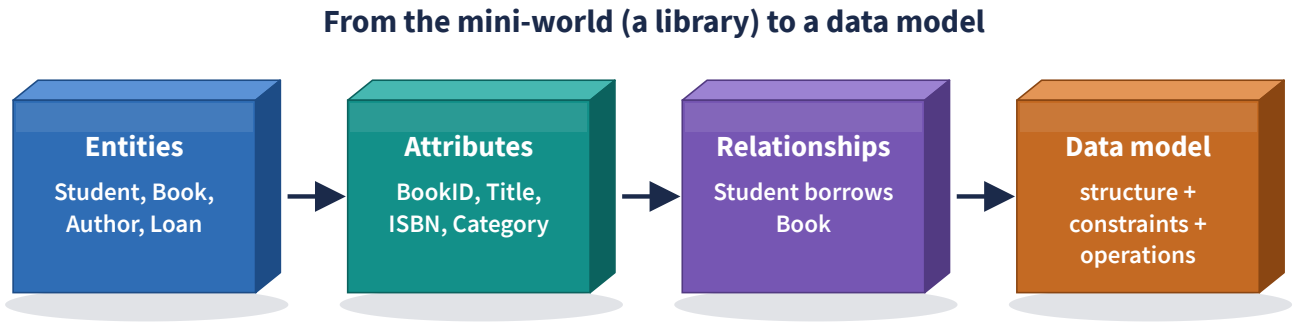


Figure 1.1 From the real world to a structured model: identify the **entities**, describe them with **attributes**, connect them with **relationships**, then add rules and operations.

1 What a data model contains

- **Entities**: Student, Book, Author, Loan.
- **Attributes**: Student(RollNo, Name, Dept, Email).
- **Relationships**: Student *borrows* Book; Author *writes* Book.
- **Constraints**: each Book has a unique ISBN.
- **Operations**: issue, return, search, update.

2 Why it is central to design

- Organises data **logically**.
- Reduces **redundancy**.
- Maintains **integrity**.
- Supports efficient **querying**.
- Adapts to **future change**.

REAL-LIFE VIEW — COLLEGE DATABASE

Entities: Student, Course, Department, Enrollment. Attributes: Student(Name, RollNo, Email), Course(CourseID, Title, Credits). Relationships: a student *enrolls in* a course; a course *belongs to* a department. Constraints: RollNo is unique; each course belongs to one department. Operations: add a student, update a course, list a student's courses.

EXAM POINT

A data model = **structure + constraints + operations**. Its main purpose is **abstraction**. Without a proper data model a database becomes inconsistent, redundant and hard to maintain.

★ **A data model is the blueprint that describes data, its relationships, its rules and its operations.**

1.2 Purpose of Data Models

Main idea. A data model is more than a picture of the data. It guides the design, development and operation of the whole database system. Without one, data becomes disorganised and inconsistent.

Table 1.1 The seven purposes of a data model

Purpose	What the model does
Why needed	Captures real-world data requirements and gives a structured way to manage them.
Organising data	Defines entities, attributes, relationships and rules so data is arranged meaningfully.
Improving communication	A common language for users, analysts and designers to discuss requirements.
Reducing complexity	Abstracts real-world detail into simple structures that are easy to understand.
Integrity and consistency	Defines constraints and rules that keep data accurate and reliable across the system.
Guiding schema design	Acts as the blueprint for the logical and physical schema.
Implementation and maintenance	Makes implementation smooth, updates easy and long-term maintenance cheaper.

REAL-LIFE VIEW — UNIVERSITY

The model captures Students, Courses, Instructors, Departments and Enrollments; shows that a student enrolls in a course taught by an instructor; and fixes rules such as "a course must belong to a department" and "a student cannot exceed the credit limit". Developers build from this plan, and it adapts as new requirements arrive.

EXAM POINT

Data models connect **real-world needs** to **technical design**. They save time and cost by reducing errors and rework, and support long-term growth.

★ **The purpose of a data model is a structured, simple and consistent representation of data.**

1.3 Components of a Data Model

Table 1.2 Five components and their roles

Component	Role	College example
Entities	Real-world objects or concepts we store data about; each has a unique identity.	Student, Course, Enrollment
Attributes	Properties that describe an entity.	Student(StudentID, Name, Email, DateOfBirth)
Relationships	How entities are connected, with cardinality 1:1, 1:N or M:N.	Student *enrolls in* Course (M:N, through Enrollment)
Constraints	Rules on entities, attributes and relationships that keep data accurate.	StudentID unique; Email not null; Grade between 0 and 100; EnrollmentDate not in the future
Operations	Actions on data: insert, update, delete, query.	Add a student, enroll in a course, update a grade

MEMORY TRICK — E A R C O

Entities have **A**ttributes, are joined by **R**elationships, obey **C**onstraints and are changed by **O**perations. What we store, how it is connected, how we keep it correct, what we can do with it.

NOTE

A many-to-many relationship (a student takes many courses, a course has many students) is implemented through a separate entity, **Enrollment**, which holds EnrollmentID, EnrollmentDate and Grade.

★ **Entities, attributes, relationships, constraints and operations together make a complete data model.**

1.4 Categories of Data Models

DEFINITION 1.2 — THREE CATEGORIES

By level of abstraction, data models are **conceptual** (high-level), **logical** or **representational** (implementation) and **physical** (low-level) models. They are three views of the **same** data.

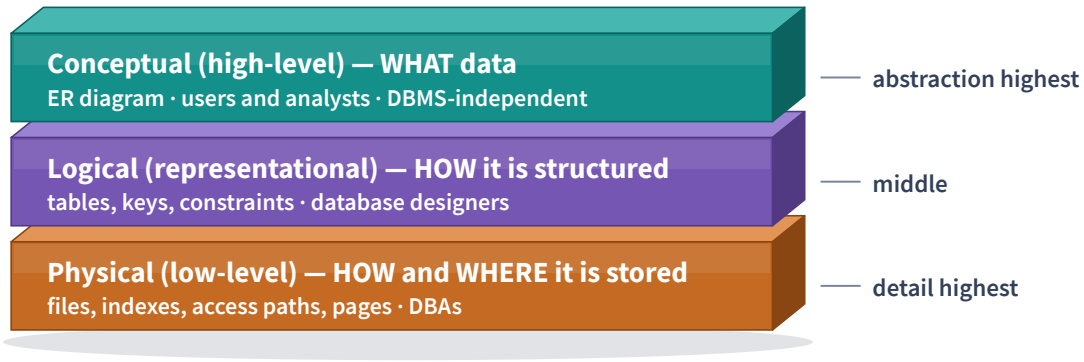


Figure 1.2 Moving from conceptual to physical, **detail increases** and **abstraction decreases**.

Table 1.3 Comparison of the data-model categories

Aspect	Conceptual (high-level)	Logical (representational)	Physical (low-level)
Abstraction	Highest	Medium	Lowest
Focus	What data is needed	How data is structured	How data is stored
Describes	Entities and relationships in business terms	Tables/records, keys, rules, constraints	Files, indexes, access paths, storage details
Independent of	DBMS and storage	Physical storage	— (depends on DBMS and device)
Primary users	Users, analysts	Database designers	System developers, DBAs
Change	Rarely	Moderately	Frequently
Example output	ER diagram	Relational or network schema	Table files, indexes, space allocation

REAL-LIFE VIEW — ONLINE BOOKSTORE

Conceptual: entities Customer, Book, Author, Order, Payment and relationships *places*, *writes*, *contains*. **Logical:** tables Customer(CustomerID, Name, …), Book(BookID, Title, Price, …), Order(OrderID, CustomerID, OrderDate, …) with primary and foreign keys. **Physical:** files on disk, indexes on BookID and CustomerID, data types, partitions of large tables, storage locations.

EXAM POINT

High-level = **what and why**; representational = **how it is structured**; low-level = **where and how it is stored**. Logical model families: relational, hierarchical, network.

★ **Conceptual, logical and physical models describe the same data at decreasing levels of abstraction.**

1.5 Conceptual Data Models

DEFINITION 1.3 — CONCEPTUAL DATA MODEL

A **conceptual data model** is a high-level, **technology-independent** representation of an organisation's data requirements: what data is needed, how it is related and the business rules that govern it, with no reference to storage or to a particular DBMS. It is usually drawn as an **Entity–Relationship (ER) diagram**.

Objectives

- Identify important data and relationships.
- Give a clear picture of business requirements.
- Act as a **communication bridge** between users and designers.
- Guide the logical and physical design.
- Ensure integrity through business rules.

Building blocks

- **User requirements**: gathered by interviews, documents, analysis.
- **Entities**: Student, Book, Doctor, Patient.
- **Attributes**: Name, DOB, ISBN; each has a **domain**.
- **Relationships**: issues, treats, manages; cardinality 1:1, 1:N, M:N.
- **Business rules**: unique IDs, policies.

REAL-LIFE VIEW — HOSPITAL

Entities Patient, Doctor, Appointment, Department. Attributes PatientID, Name, DOB; DoctorID, Specialty; AppointmentDate. A patient books many appointments; a doctor schedules many appointments. Rule: an appointment is with **one** doctor for **one** patient at a given date and time.

EXAM POINT

Advantages: simple view of requirements, better user–developer communication, **DBMS-independent**, catches incomplete or inconsistent requirements **early**, foundation for logical design. Notation: **ER diagram**.

★ **The conceptual model shows the real-world view of the data: what is needed and how it relates.**

1.6 Logical Data Models

DEFINITION 1.4 — LOGICAL DATA MODEL

A **logical data model** describes data and relationships in a structured, **implementation-neutral** form: entities, attributes with domains, primary and foreign keys, relationships and constraints, without storage devices, indexes, file organisations or partitions. It is the bridge between the conceptual model and the physical model.



The result is a **logical schema**, in the relational case a set of tables with attributes, keys and relationships, ready to be mapped to a physical design.

Table 1.4 University database — logical model

STUDENT	COURSE	ENROLMENT
StudentID (PK), Name, Email, DateOfBirth	**CourseID (PK)**, CourseName, Credits	**StudentID (FK)**, **CourseID (FK)**, EnrollDate, Grade

Table 1.5 Types of logical model

Model	Organisation
Relational	Tables (relations) with rows and columns; relationships through keys. Most widely used.
Hierarchical	Tree (parent–child); each child has only one parent.
Network	Graph; many-to-many relationships through sets (pointers).

EXAM POINT

ENROLMENT **resolves the many-to-many** relationship between STUDENT and COURSE. Primary keys give unique identification; foreign keys give **referential integrity**. A logical model is independent of storage, indexes and access paths.

★ **The logical model turns conceptual ideas into tables, attributes, keys and relationships.**

1.7 Physical Data Models

DEFINITION 1.5 — PHYSICAL DATA MODEL

A **physical data model** is the lowest level of data modelling. It describes **how data is actually stored** on secondary storage: file structures, indexes, record formats, storage allocation and access paths of a specific DBMS.

Table 1.6 What a physical model specifies

Item	Meaning
File organisation	How records are grouped into files: heap (unordered), sorted, hashed.
Indexes	B+ tree, hash, bitmap indexes that speed up retrieval.
Access paths	Full table scan, index scan, range scan.
Record formats	Field order, data types, lengths, alignment, record headers.
Storage allocation	Pages, extents, blocks; free-space and overflow management.
Performance	I/O cost, storage use, clustering, data locality, concurrency effects.

Logical model

- Focus: **what** data is stored and its relationships.
- High-level, DBMS-independent.
- Entities, attributes, relationships, constraints.
- Purpose: integration and consistency.

Physical model

- Focus: **how** data is stored and accessed.
- Low-level, DBMS-specific.
- Files, indexes, record formats, access paths.
- Purpose: performance and efficient storage.

The physical model can be seen directly in SQL. An index belongs to the physical level: the query does not change, only the way the DBMS reaches the rows.

SQL physical_index.sql — uses tables: college

```sql
CREATE INDEX idx_student_dept ON Student(DeptID);   -- physical: access path
EXPLAIN QUERY PLAN SELECT Name FROM Student WHERE DeptID = 10;
SELECT Name FROM Student WHERE DeptID = 10;          -- same query, same answer
```

▶ Result (SQLite)

id	parent	notused	detail
3	0	61	SEARCH Student USING INDEX idx_student_dept (DeptID=?)

Name
Aarav
Kiran

Screen 1.1 SQLite reports that it will **search** Student **using the index** instead of scanning every row. The result is the same two students; only the access path changed.

REAL-LIFE VIEW — ONLINE BOOKSTORE

Books stored as a clustered file on BookID; a B+ tree index on BookTitle for title search; a composite index on Orders(CustomerID, OrderDate); fixed-length records with aligned fields; 8 KB pages with free space kept on a free list.

EXAM POINT

Physical model = **files, indexes, record formats, storage allocation, access paths**. It changes as data volume and workload change, and good physical design improves performance and scalability.

★ **The physical model is the detailed blueprint of how data sits on disk and how it is reached.**

1.8 Relational Data Model

DEFINITION 1.6 — RELATIONAL DATA MODEL

The **relational data model** represents data as a collection of **relations (tables)**. Each table has a fixed set of uniquely named columns; each row (**tuple**) is one record about an entity. It was introduced by **E. F. Codd in 1970** and is based on **set theory** and **first-order predicate logic**.

Relation (table) — related data organised in rows and columns; each relation has a unique name

Tuple (row) — one record, one instance of the entity

Attribute (column) — a named property of the entity

Domain — the set of valid values of an attribute; all values in a column come from one domain

Primary key — the key chosen to identify each tuple uniquely

Foreign key — attribute(s) referring to the primary key of another table; it represents the relationship

SQL relational.sql — uses tables: college

```sql
SELECT s.StudentID, s.Name, s.DeptID, d.DeptName, d.Location
FROM Student s JOIN Department d ON s.DeptID = d.DeptID;
INSERT INTO Student VALUES (105, 'Rohan', 'rohan@mail.com', 40, '2023-07-09');
```

▶ Result (SQLite)

StudentID	Name	DeptID	DeptName	Location
101	Aarav	10	Computer Science	Block A
102	Diya	20	Electronics	Block B
103	Kiran	10	Computer Science	Block A
104	Meera	30	Mechanical	Block C

```
Error (line 3): FOREIGN KEY constraint failed
```

Screen 1.2 The relationship Student–Department is stored only as a **value**: the foreign key DeptID. A join follows the value. The last insert is refused because department 40 does not exist: the foreign key enforces **referential integrity**.

1 Manipulation

- **Relational algebra**: select, project, join, union, difference.
- **SQL**: declarative insert, retrieve, update, delete.

2 Advantages

- Data independence.
- Reduced redundancy.
- Integrity through keys and constraints.

3 More advantages

- Flexible: easy add, modify, delete.
- Powerful SQL querying.
- Scales from small to very large databases.

EXAM POINT

Relational model: **Codd, 1970** ("A Relational Model of Data for Large Shared Data Banks"). Relationships are represented by **matching key values**, not by pointers. Examples: Oracle, MySQL, PostgreSQL, SQL Server, DB2, SQLite.

★ **The relational model stores data in tables and links them through primary and foreign keys.**

1.9 Hierarchical Data Model

DEFINITION 1.7 — HIERARCHICAL DATA MODEL

The **hierarchical data model** organises data as a **tree**. Each record except the **root** has **exactly one parent** and zero or more children, so it supports only **one-to-many** relationships. It is one of the earliest models (IBM's **IMS**, 1960s).

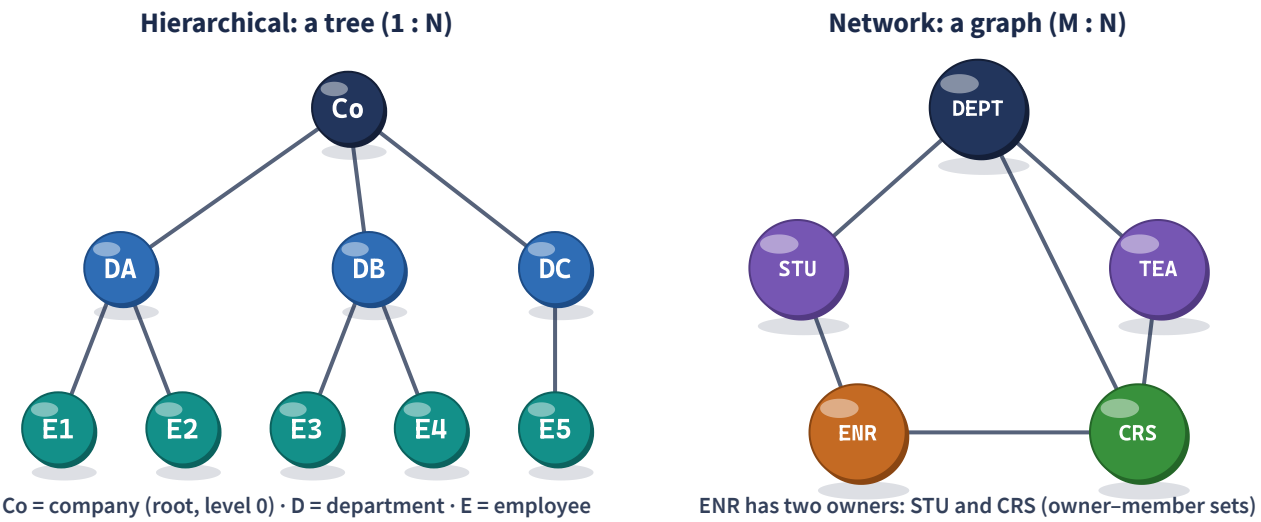


Figure 1.3 Left: a hierarchy with one root and levels 0, 1, 2; each employee has one parent department. Right: a network, where a record such as ENR may belong to several owners.

Features

- **Root**: the single topmost record (level 0).
- **Levels**: children of the root at level 1, theirs at level 2.
- **Parent–child** links; each child has one parent.
- **Navigation path**: start at the root and follow a unique path down.
- Efficient for naturally hierarchical data.

Limitations

- Only 1:N; **M:N cannot be modelled directly**.
- Rigid: structural change is difficult.
- Data redundancy (a record needed under two parents is stored twice).
- A child cannot exist without its parent; deleting a parent deletes its children.
- Navigation is complex in large hierarchies.

REAL-LIFE VIEW — LIBRARY TREE

Library → Section → Book. Root: Library; level 1: Science, Fiction, History; level 2: books. Path: Library → Science → Physics → "Basics of Physics". Each book belongs to one section, each section to the library. Organisation charts, file systems and classification data are natural hierarchies.

EXAM POINT

Hierarchical = tree, **1:N only**, one parent per child, access from the root (navigational, record-at-a-time). Example DBMS: **IBM IMS**. Compared with the relational model it lacks flexibility and easy querying.

★ **The hierarchical model is a tree of parent–child records with one root and only one-to-many links.**

1.10 Network Data Model

DEFINITION 1.8 — NETWORK DATA MODEL

The **network data model** represents data as a **graph** of records connected by links. A record may have **several owners (parents)** and several members (children), so **many-to-many** relationships are natural. It was standardised by the **CODASYL DBTG** (Data Base Task Group) report.

Record type — an entity such as STUDENT, COURSE, TEACHER

Set type — a named relationship between two record types (Enrolls, Teaches, OfferedBy)

Owner / member — in each set one record type is the owner, the other the member; a member may belong to several owners

Navigation — follow pointers from record to record; several valid paths may exist

Table 1.7 Network model versus hierarchical model

Aspect	Network model	Hierarchical model
Structure	Graph (network) with links	Tree
Parents	A record can have many parents	Exactly one parent
Relationships	1:N and M:N	1:N only
Navigation	Many possible paths	Single path from the root
Redundancy	Less	More
Complexity	More complex	Simpler
Use cases	Complex interrelated data (academic, telecom)	Simple hierarchies (org chart, file system)

REAL-LIFE VIEW — UNIVERSITY NETWORK

A student enrolls in many courses; a course has many students; a teacher teaches many courses; a department has many teachers and offers many courses. ENROLLMENT links a student and a course and carries the grade. Every entity has several connections.

EXAM POINT

Network model terms: **record type, set type, owner, member** (CODASYL DBTG, IDMS). Like the hierarchical model it is **navigational and pointer-based**, so programs depend on the access paths. Disadvantage: complex structure, hard to change.

★ **The network model is a graph of records joined by owner–member sets, so a record may have many parents.**

1.11 Object-Oriented Data Model

DEFINITION 1.9 — OBJECT-ORIENTED DATA MODEL

The **object-oriented data model** stores data as **objects** that combine **state** (attributes) and **behaviour** (methods). It applies object-oriented programming concepts to databases and supports complex data, inheritance, encapsulation and reuse.

Class — a template defining attributes and methods (Book)

Object — an instance of a class with actual values (memberID = M001, name = "Riya")

Methods — operations an object can perform: getDetails(), isAvailable(), updatePrice()

Object identity (OID) — a unique, system-given identity that distinguishes each object, independent of its values

Encapsulation — data and methods bundled together; internal details hidden

Inheritance — EBook and PrintedBook inherit Book's attributes and methods and add their own

Polymorphism — the same method name behaves differently in subclasses

Table 1.8 Relational versus object-oriented approach

Aspect	Relational	Object-oriented
Representation	Tables (rows and columns)	Objects (state + behaviour)
Structure	Flat, record-based	Complex, structured
Relationships	Foreign keys	Inheritance and references
Behaviour	No behaviour in data	Methods are part of objects
Reusability	Limited	High (inheritance)
Identity	Primary key	Object identity (OID)
Complex data	Difficult	Easily handled

EXAM POINT

Applications: **CAD**, simulations, multimedia (images, audio, video), engineering and scientific data, persistent OOP applications. Standard: **ODMG** with **ODL** and **OQL**. OID ≠ primary key: it is hidden and never reused.

★ **The object-oriented model stores objects that combine state with behaviour, using classes and inheritance.**

1.12 Object-Relational Data Model

DEFINITION 1.10 — OBJECT-RELATIONAL DATA MODEL

The **object-relational data model** extends the relational model with **user-defined (complex) types**, inheritance, methods and nested structures, while keeping tables and SQL. It joins the reliability of relational databases with the power of objects (SQL:1999; Oracle, PostgreSQL, IBM DB2).

Features added to tables

- **User-defined types** (Address object: Street, City, ZIP).
- **Large objects**: images and video as BLOB.
- **Inheritance**: Employee → Faculty, Staff.
- **Methods** stored with the type.
- **Nested** and collection attributes (arrays, lists).

Applications

- Geographic Information Systems (GIS).
- Computer-Aided Design (CAD).
- Multimedia databases.
- Scientific, engineering and medical imaging.
- E-commerce and CRM.

SQL customer_type.sql — Oracle

```sql
CREATE TYPE Customer_t AS OBJECT (
  CustomerID INT,
  Name       VARCHAR2(100),
  Phone      VARCHAR2(15),
  MEMBER FUNCTION GetSummary RETURN VARCHAR2
);
CREATE TABLE Orders (
  OrderID     INT PRIMARY KEY,
  Customer    Customer_t,          -- column of an object type
  OrderDate   DATE,
  TotalAmount NUMBER(10,2)
);
```

Screen 1.3 An Oracle object type with a method, used as a column type. SQLite has no object types, so this listing is shown but not run.

Table 1.9 Relational versus object-relational

Aspect	Relational	Object-relational
Data types	Primitive (int, char, date)	Primitive + complex + user-defined
Structure	Flat tables	Tables + objects + nested structures
Inheritance, methods	Not supported	Supported
Multimedia	Limited (BLOBs)	Better (objects for media)
Best for	Simple business data	Complex domain data (CAD, GIS, multimedia)

★ **The object-relational model is the relational model plus user-defined types, inheritance, methods and nesting.**

1.13 Semi-Structured Data Model

DEFINITION 1.11 — SEMI-STRUCTURED DATA

Semi-structured data has some structure but no fixed schema: different records in the same collection may have different or nested fields. It is **self-describing**, because tags or keys travel with the data, and is written in **XML** (eXtensible Markup Language) or **JSON** (JavaScript Object Notation).

XML — tagged hierarchy

SYNTAX

```
<student>
  <id>101</id>
  <name>Aarav Sharma</name>
  <courses><course>DBMS</course>
    <course>OS</course></courses>
  <address><city>Pune</city></address>
</student>
```

JSON — nested objects

SYNTAX

```
{ "id": 101,
  "name": "Aarav Sharma",
  "courses": ["DBMS", "OS"],
  "address": { "city": "Pune",
               "pin": "411001" } }
```

A rigid table forces every record to have the same columns. Semi-structured records may differ: record 1 has an address and several courses, record 2 one course, record 3 an extra phone number. Modern relational systems can store and query JSON as well:

SQL json_docs.sql

```sql
CREATE TABLE StudentDoc (doc TEXT);
INSERT INTO StudentDoc VALUES
 ('{"id":101,"name":"Aarav","courses":["DBMS","OS"],"address":{"city":"Pune"}}'),
 ('{"id":102,"name":"Neha","courses":["DBMS"],"phone":"98400"}');
SELECT json_extract(doc,'$.name') AS name,
       json_array_length(doc,'$.courses') AS courses,
       json_extract(doc,'$.address.city') AS city,
       json_extract(doc,'$.phone') AS phone
FROM StudentDoc;
```

▶ Result (SQLite)

name	courses	city	phone
Aarav	2	Pune	*NULL*
Neha	1	*NULL*	98400

Screen 1.4 Two documents with different fields in one column. A missing field (Neha's city, Aarav's phone) simply comes back as NULL.

Advantages

- Flexible schema for changing requirements.
- Self-describing; less need for an external schema.
- Easy exchange across platforms and languages.
- Nested, hierarchical data handled naturally.

Limitations

- Less efficient for complex queries.
- Limited validation and constraints.
- More storage for large volumes (tags repeat).
- Performance depends on parsing.

EXAM POINT

XML and JSON are **self-describing**, schema-less (or "schema-later") formats. Uses: web services and REST APIs, document and content management, configuration files and logs, data interchange between heterogeneous systems.

★ **Semi-structured data carries its own structure in tags or keys and does not need a fixed schema.**

1.14 NoSQL Data Models

DEFINITION 1.12 — NOSQL

NoSQL ("Not Only SQL") databases are **non-relational** databases designed for **scalability**, schema flexibility and very large volumes of distributed, semi-structured or unstructured data, with high performance and availability.

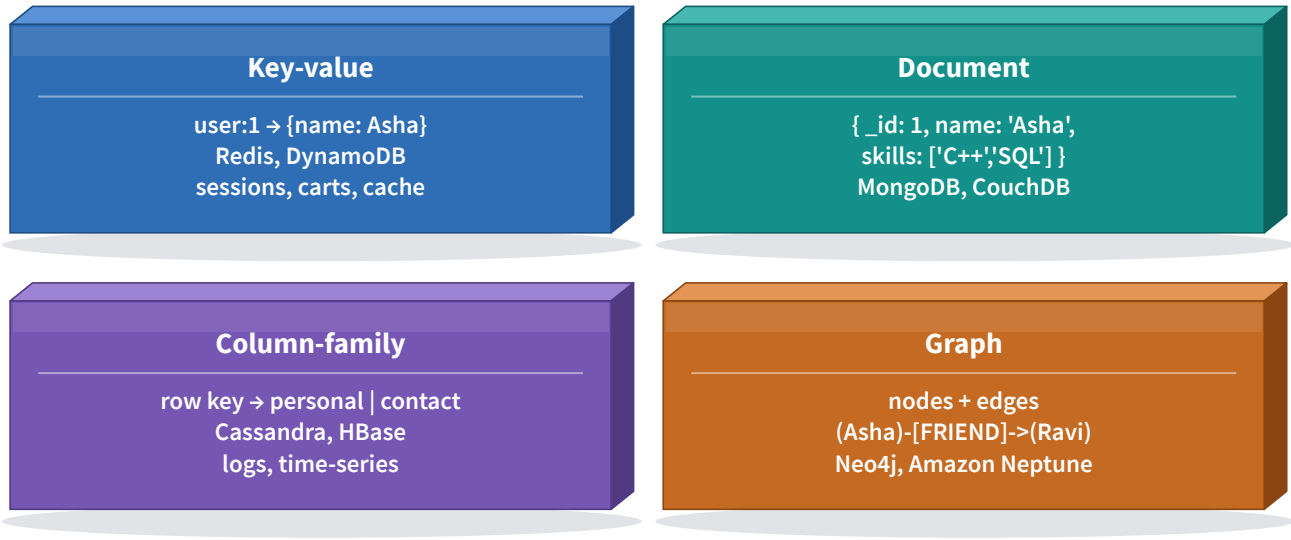


Figure 1.4 The four NoSQL families, each with typical products and uses.

Table 1.10 Relational versus NoSQL

Aspect	Relational (SQL)	NoSQL
Data model	Tables (rows and columns)	Key-value, document, column-family, graph
Schema	Fixed	Schema-less / flexible
Scaling	**Vertical** (bigger server)	**Horizontal** (more servers)
Joins	Supported	Limited or not supported
Transactions	Strong **ACID**	**BASE** / eventual consistency
Best for	Complex queries, transactions	Big data, high scalability, real-time apps

REAL-LIFE VIEW — SOCIAL MEDIA PLATFORM

Key-value: sessions, preferences, like counts. Document: profiles, posts, comments. Column-family: activity logs (time series). Graph: friendships, followers, connections.

EXAM POINT

BASE = Basically Available, Soft state, Eventual consistency. **CAP theorem**: a distributed store can guarantee at most two of Consistency, Availability and Partition tolerance. Redis = key-value; MongoDB = document; Cassandra/HBase = column-family; Neo4j = graph.

★ **NoSQL models trade fixed schemas and strict ACID for scale, flexibility and availability.**

1.15 Comparison of Data Models

Main idea. Each model suits a different kind of problem. **Classical** models (hierarchical, network, relational) are structure-driven; **modern** models (object-oriented, object-relational, semi-structured, NoSQL) add flexibility and scale.

Table 1.11 Comparison of the major data models

Model	Structure	Schema	Relationships	Query style	Strength / limitation
Relational	Tables	Fixed (schema-on-write)	Foreign keys	SQL (set-based)	Simple, mature, standard / joins can be costly
Hierarchical	Tree	Rigid	Strict 1:N	Navigational	Fast hierarchical access / inflexible
Network	Graph (owner sets)	Rigid	M:N via set types	Navigational	Efficient M:N / complex programming
Object-oriented	Classes, objects	Class-based	Object references	OQL / methods	Models the real world / no standard language
Object-relational	Tables + UDTs	Mostly fixed	Keys + object refs	SQL + extensions	SQL power + complex types / complexity
Semi-structured	Tagged tree	Flexible	Nesting	XPath, XQuery, JSON paths	Irregular data / weak validation
NoSQL	KV, document, column, graph	Highly flexible	Varies	APIs, own languages	Distributed scale / eventual consistency

Table 1.12 How to choose a data model

Question	Guideline
Data structure	Strict hierarchies → hierarchical/network; tabular → relational; complex objects → OO; nested/irregular → semi-structured.
Scale and performance	Massive, distributed → NoSQL; moderate scale with strong consistency → relational.
Relationship needs	Rich relationships → relational/object-relational; simple 1:N → hierarchical; M:N → network.
Schema flexibility	Fixed → relational/object-relational; evolving or unknown → semi-structured or NoSQL.
Consistency vs availability	Strong consistency → relational; high availability and partition tolerance → NoSQL.
Tools and growth	Prefer mature query languages (SQL) and models that scale and integrate.

EXAM POINT

Record-based models: **hierarchical, network, relational**. Object-based models: **ER, object-oriented**. Data independence: relational is highest; hierarchical and network are lowest. The trend has moved from rigid, structure-bound models to flexible, distributed ones.

★ **The right data model depends on structure, relationships, scale, flexibility and consistency needs.**

Chapter 1 summary

- A data model describes **structure, constraints and operations**; components: entities, attributes, relationships, constraints, operations.
- Purposes: organise, communicate, reduce complexity, keep integrity, guide schema design, ease maintenance.
- Categories: **conceptual** (ER, what), **logical** (tables and keys, how structured), **physical** (files and indexes, how stored).
- **Relational** (Codd 1970): tables, tuples, attributes, domains, keys; relationships by foreign-key values.
- **Hierarchical** (IMS): tree, one parent, 1:N only. **Network** (CODASYL): graph, owner–member sets, M:N.
- **Object-oriented**: objects = state + behaviour, OID, inheritance. **Object-relational**: tables + UDTs, methods, nesting.
- **Semi-structured**: XML/JSON, self-describing, flexible schema. **NoSQL**: key-value, document, column-family, graph; BASE, horizontal scaling.
- Choose a model by structure, relationships, scale, schema flexibility and consistency needs.

A data model is a way of describing data. Chapter 2 looks at the description itself, the **schema**, and at the data it describes at any moment, the **instance**.

CHAPTER · UNIT II · DATABASE CONCEPTS

2 Schemas, Instances and Metadata

OPENING SITUATION

The class register printed at the start of the year has fixed columns: Roll No, Name, Class, Attendance. That printed format hardly ever changes. The names written in it change every week as students join and leave. A database works the same way: the fixed design is the **schema**, the rows present today are the **instance**, and the DBMS keeps a register of its own design, the **metadata**, in the **data dictionary** and **system catalog**.

IN THIS CHAPTER

- 2.1 Meaning of Database Schema
- 2.2 Schema Definition
- 2.3 Schema Diagram
- 2.4 External Schema
- 2.5 Conceptual Schema
- 2.6 Internal Schema
- 2.7 Logical Schema
- 2.8 Physical Schema
- 2.9 Meaning of Database Instance
- 2.10 Database State
- 2.11 Initial Database State
- 2.12 Current Database State
- 2.13 Valid Database State
- 2.14 Schema versus Instance
- 2.15 Intension and Extension
- 2.16 Metadata
- 2.17 Data Dictionary
- 2.18 System Catalog

YOU NEED

- Chapter 1: data models, tables, primary and foreign keys.

2.1 Meaning of Database Schema

DEFINITION 2.1 — DATABASE SCHEMA

A **database schema** is the **formal description of the structure** of a database: its tables (entities), attributes, relationships, constraints and the organisation of data. It is the logical blueprint and contains **no actual data values**.

Table 2.1 Student–Course schema

STUDENT	ENROLLMENT	COURSE
StudentID (PK), Name, Email, DateOfBirth, Department	**EnrollmentID (PK)**, StudentID (FK), CourseID (FK), EnrollDate, Grade	**CourseID (PK)**, CourseName, Credits, Department
One student can have many enrollments; one course can have many enrollments.		

Schema (structure)

- The design: which tables exist, their attributes, relationships and rules.
- Stable; remains the same while data changes.

Instance (data)

- The actual values stored in those tables at a particular time.
- Changes with every insert, update and delete.

EXAM POINT

A schema describes **tables, attributes, relationships, constraints and organisation**. It is the foundation for creating and managing a database and ensures consistency, integrity and efficient organisation. Each object in it (STUDENT, COURSE) is a **schema construct**.

★ **A schema is the blueprint of the database: structure and rules, never the data itself.**

2.2 Schema Definition

DEFINITION 2.2 — SCHEMA DEFINITION

Schema definition is the process of formally describing the structure of a database **before any data is stored**: table names, attributes, data types, keys, relationships, integrity constraints and business rules. It is written in the **Data Definition Language (DDL)**.

Main idea. DDL commands create, modify or remove database objects: `CREATE`, `ALTER`, `DROP`, `TRUNCATE`, `RENAME`. The DBMS records each definition in its catalog, and from then on checks every row against it.

SQL schema_definition.sql

```sql
CREATE TABLE Department (DeptID INT PRIMARY KEY, DeptName TEXT);
CREATE TABLE Student (
  StudentID INT PRIMARY KEY,
  Name      VARCHAR(50) NOT NULL,
  Email     VARCHAR(100) UNIQUE,
  DeptID    INT,
  CONSTRAINT fk_student_dept FOREIGN KEY (DeptID) REFERENCES Department(DeptID)
);
SELECT name AS column_name, type AS data_type, "notnull", pk FROM
pragma_table_info('Student');
```

▶ Result (SQLite)

column_name	data_type	notnull	pk
StudentID	INT	0	1
Name	VARCHAR(50)	1	0
Email	VARCHAR(100)	0	0
DeptID	INT	0	0

Screen 2.1 The schema of STUDENT read back from the DBMS: every column with its data type, whether NULL is refused and which column is the primary key. No row has been stored yet.

Table 2.2 STUDENT table structure as defined

Attribute	Data type	Constraint	Description
StudentID	INT	PRIMARY KEY	Unique ID of the student
Name	VARCHAR(50)	NOT NULL	Student's full name
Email	VARCHAR(100)	UNIQUE	Student's email address
DeptID	INT	FOREIGN KEY	References DEPARTMENT

EXAM POINT

Schema definition describes **structure, not data**, and is implemented with **DDL**. A well-defined schema gives accuracy and consistency; schema changes must be planned carefully because programs depend on them.

★ **Schema definition is the DDL specification of tables, types, keys, relationships and rules.**

2.3 Schema Diagram

DEFINITION 2.3 — SCHEMA DIAGRAM

A **schema diagram** is a **visual representation** of a database schema: entities (tables), attributes and relationships drawn with symbols and lines. The common forms are **ER diagrams** and **relation (schema) diagrams**. It shows structure only, never data values.

Table 2.3 Common ER symbols

Symbol	Shape	Meaning
Entity	Rectangle	A real-world object or concept (table): STUDENT, BOOK
Attribute	Oval	A property (field): Name, Email, ISBN
Primary key	Underlined attribute	Uniquely identifies each record: StudentID, BookID
Relationship	Diamond	An association between entities: BORROWS
Cardinality	1, N written on the lines	How many instances are related

How to read one

- Find the **entities** (rectangles).
- Read the **attributes** (ovals) attached to each.
- Look at the **relationships** (diamonds).
- Check the **cardinality** (1 or N).
- Understand the rules it expresses.

Why useful

- Clear overview of the database.
- Reveals missing data or relationships.
- Improves team communication.
- Guides development and future change.
- Documentation and maintenance.

REAL-LIFE VIEW — LIBRARY

STUDENT(StudentID, Name, Email, Phone) and BOOK(BookID, Title, Author, ISBN) joined by the diamond BORROWS. One student can borrow many books; over time one book is borrowed by many students.

★ **A schema diagram is a visual map of entities, attributes, relationships and cardinality.**

2.4 External Schema

DEFINITION 2.4 — EXTERNAL SCHEMA

An **external schema** (**user view** or **subschema**) describes how one user group or application sees the database: only the data relevant to it, in a convenient form, with the rest hidden. It is derived from the conceptual schema and sits between the user and the conceptual schema.

Table 2.4 Three external schemas of one college database

View	Attributes shown	Purpose
Student view	StudentID, Name, CourseName, Grade	a student sees own courses and grades
Faculty view	FacultyID, Name, Department, CourseName	teaching details
Accounts view	StudentID, CourseName, FeesPaid, Balance	payments and dues only

✓ Characteristics

- Shows only relevant **attributes** and a **subset of rows**.
- Derived from the conceptual schema.
- Hides unnecessary data.
- **Many** external schemas can exist.
- Arranges data in a form easy for that user.

★ Benefits

- **Security**: sensitive data is invisible.
- Simpler access for end users.
- Supports different requirements.
- Reduces complexity.
- Gives **logical data independence**.

EXAM POINT

External schema = user view = subschema. Changes in the conceptual schema do not affect an external schema as long as the data that view needs remains available: this is **logical data independence**.

★ **An external schema is one user's personalised, secure view of the database.**

2.5 Conceptual Schema

DEFINITION 2.5 — CONCEPTUAL SCHEMA

The **conceptual schema** is the complete logical description of the **whole database** at the **community level**: all entities, attributes, relationships, constraints and the overall organisation of data, independent of user views and of physical storage.

Main idea. It integrates the data requirements of the entire organisation into **one unified view**. For a university: STUDENT, COURSE, ENROLLMENT, DEPARTMENT and FACULTY with all their keys, links and rules. It is the bridge between user requirements and implementation.

Table 2.5 The three schema levels compared

Level	Focus	Scope	View
External schema	User-specific views	Individual users	Partial view (what users see)
Conceptual schema	Complete logical structure	Whole organisation	Unified view of all data
Internal schema	Physical storage details	DBMS / system	How data is stored

★ The conceptual schema is the single, consistent logical view of the whole database.

2.6 Internal Schema

DEFINITION 2.6 — INTERNAL SCHEMA

The **internal schema** (physical schema) is the **storage-level** description of the database: how data is physically stored and accessed on secondary storage, through files, indexes, record formats, access paths and storage structures.

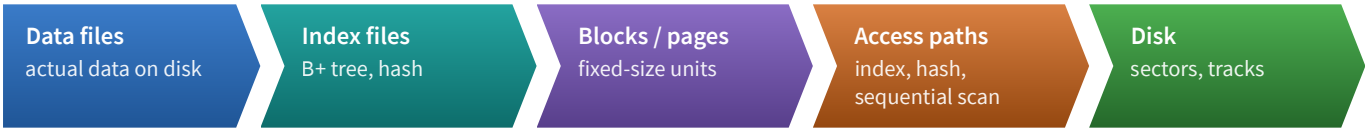


Table 2.6 What the internal schema contains

Item	Details
File organisation	Heap, sorted, hashed or clustered files; clustering stores related records together to cut disk I/O.
Indexes	B+ trees and hash indexes for fast lookup; primary and secondary indexes kept as separate index files.
Record formats	Fixed-length or variable-length; fields, delimiters, null indicators, overhead.
Access paths	Sequential scan, index scan, clustered index scan, direct access.
Storage structures	Pages/blocks grouped into extents on disk; contiguous, linked or indexed allocation.

REAL-LIFE VIEW — HOW STUDENT IS STORED

A data file on disk; a B+ tree index on StudentID in a separate index file; variable-length records; 4 KB pages grouped in 64 KB extents; search by StudentID uses an index scan.

★ The internal schema says how data is physically stored, organised and reached on disk.

2.7 Logical Schema

DEFINITION 2.7 — LOGICAL SCHEMA

The **logical schema** is the logical design of the database **as understood by the DBMS**: tables, columns, primary and foreign keys, relationships and integrity constraints, independent of how the data is physically stored.

Table 2.7 Conceptual, logical and physical schema

Aspect	Conceptual	Logical	Physical
Focus	What data is needed	How data is structured	How data is stored
Viewpoint	User / business	DBMS	Storage / hardware
Details	High-level entities and relationships	Tables, columns, keys, relationships, constraints	Files, indexes, partitions, storage paths

Example: in ENROLLMENT(EnrolID, StudentID, CourseID, EnrollDate, Grade), StudentID references STUDENT(StudentID) and CourseID references COURSE(CourseID); primary keys identify rows, foreign keys link tables and enforce referential integrity.

COMMON MISTAKE 2.1 — "THE LOGICAL SCHEMA GIVES LOGICAL DATA INDEPENDENCE: CHANGES IN STORAGE DO NOT AFFECT THE LOGICAL DESIGN"

The source page lists this as an advantage. Storage changes that leave the logical schema untouched are **physical** data independence. **Logical** data independence means that changes to the conceptual/logical schema (a new column, a split table) do not affect the external views and application programs (Chapter 4).

EXAM POINT

The logical schema is the interface between the users' view and physical storage. It gives a precise structure, lets the DBMS manage data efficiently, supports integrity through keys and constraints, and eases application development. Changes to physical storage do not require changes to it.

★ **The logical schema is the DBMS's view of tables, keys, relationships and constraints.**

2.8 Physical Schema

DEFINITION 2.8 — PHYSICAL SCHEMA

The **physical schema** is the actual implementation of the database on storage devices: storage files, indexes, partitions, clustering, record placement, access methods and performance details.

Table 2.8 What the physical schema covers

Component	Examples
Storage files	Data files, log files, temporary files, backup files
Indexes	Primary and secondary indexes; B-tree, hash
Partitions	Range, hash and list partitioning of large tables
Clustering	Physical ordering of data to reduce I/O and speed up joins
Record placement	Page size, block size, slot management, free-space handling
Access methods	Sequential, index, bitmap and clustered index scans
Performance details	Compression, caching, buffer management, I/O scheduling

REAL-LIFE VIEW — A STUDENT TABLE WITH MILLIONS OF ROWS

B-tree index on StudentID; 8 KB pages; partitioned by Department (CSE, ECE, MECH); clustered on Department; index scan for searches and sequential scan for reports; compression and caching for speed.

EXAM POINT

The physical schema is **transparent to users** but decides performance, space efficiency, scalability and reliability (backup, recovery, fault tolerance).

★ The physical schema decides how logical data is laid out on storage for speed and space.

2.9 Meaning of Database Instance

DEFINITION 2.9 — DATABASE INSTANCE

A **database instance** is the **actual data** stored in the database **at a specific moment**: a snapshot of all tables. The schema stays the same; the instance changes whenever data is inserted, updated or deleted.

Characteristics

- **Current state**: data at one point in time.
- **Varies over time**.
- **Based on the schema**: must follow its rules.
- Holds **actual values**.
- **Many instances** exist for one schema over time.

Example

- STUDENT at 10:00 AM on 01-May-2025: four rows (Aarav, Meera, Rohan, Sneha).
- At 02:00 PM a new student is added or an email updated.
- A **different instance** of the **same** schema.

EVERYDAY ANALOGY

The schema is the fixed format of the class register (StudentID, Name, Email columns). The instance is the list of students written in it on a given day. Tomorrow some students are added or removed; the format stays, the content changes.

★ An instance is the real data in the database at one moment; it changes as data is modified.

2.10 Database State

DEFINITION 2.10 — DATABASE STATE

The **database state** is the content of the database at a particular point in time. The database passes through a **sequence of states** S0 → S1 → S2 → ... as transactions insert, update and delete data.

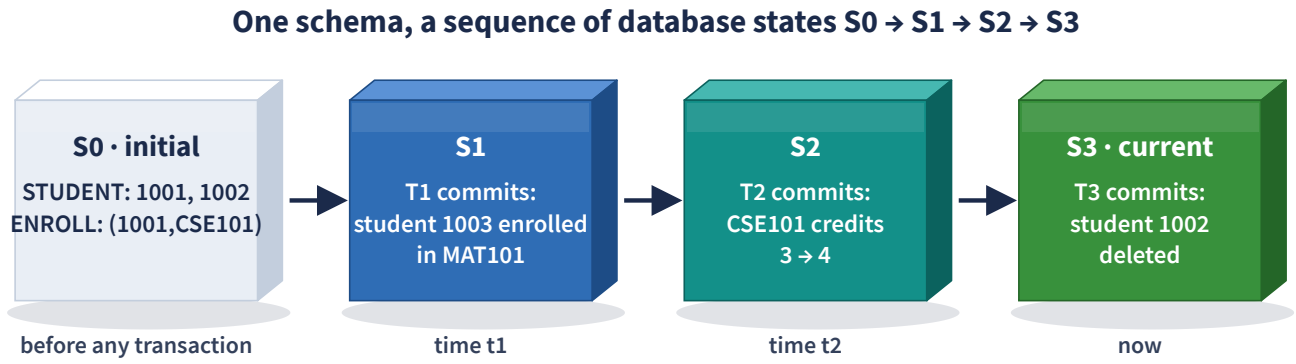


Figure 2.1 STUDENT–ENROLLMENT database. Each committed transaction moves the database to the next state; the schema never changes.

Table 2.9 How operations change the state

Operation	Effect on the state
INSERT + COMMIT	adds data → new state
UPDATE + COMMIT	modifies data → new state
DELETE + COMMIT	removes data → new state
ROLLBACK	undoes uncommitted changes → state remains the same

SQL rollback_state.sql — uses tables: univ

```sql
BEGIN;
INSERT INTO Student VALUES (1003, 'Rohan', 'rohan@clg.edu', 1);
SELECT COUNT(*) AS rows_inside_transaction FROM Student;
ROLLBACK;                                   -- change discarded
SELECT COUNT(*) AS rows_after_rollback FROM Student;
```

▶ Result (SQLite)

rows_inside_transaction
3

rows_after_rollback
2

Screen 2.2 Inside the transaction the new row is visible (3 rows), but after `ROLLBACK` the database is back in its earlier state (2 rows). Only committed work becomes a new state.

NOTE — TWO WAYS OF SPEAKING

Elmasri–Navathe say that **every** insert, delete or modify changes the database state. Transaction-oriented texts, like this unit, count only **committed** states, because an uncommitted change may still be rolled back. Both agree that the committed state is what other users see and what recovery restores.

EXAM POINT

The sequence of committed states is the basis of **concurrency control, isolation and recovery**: older states can be restored after a failure, and the database must be in a valid state after every commit.

★ The database moves from state to state as transactions commit; rollback leaves the state unchanged.

2.11 Initial Database State

DEFINITION 2.11 — INITIAL STATE

The **initial database state** is the state immediately after the database is created from its schema or first loaded (from a file, template or migration script), before normal transactions start.

Table 2.10 Three possible forms of the initial state (college database)

Form	What the tables hold	Example
Empty database	All tables exist, no rows.	STUDENT, COURSE, ENROLLMENT: 0 rows
Default values	Rows with system-defined defaults.	(1001, 'Unknown', 'N/A', 'Undeclared')
Starter records	A small set of essential data preloaded.	(C101, 'Database Systems', 4, 'CSE')



EXAM POINT

All constraints are **active even in the initial state**: primary keys must be unique from the start and foreign keys must reference existing rows. The initial state is the predictable baseline for testing, development, migration and all later operations.

★ The initial state is the database's valid starting point right after creation or loading.

2.12 Current Database State

DEFINITION 2.12 — CURRENT STATE

The **current database state** is the data stored in the database **at the present moment**, reflecting all transactions committed up to now. A **snapshot** is a view of the state at one particular time.

Why it changes often

- Users insert, update and delete records.
- Many transactions run concurrently.
- System processes and scheduled jobs modify data.
- External events (file imports, logs) update tables.

Transactions and the current state

- A **committed** change becomes part of the current state.
- A **rolled-back** change is discarded.
- Concurrent committed transactions together shape the state all users see.

EXAM POINT

Current state = **committed data at the present moment**; uncommitted changes are not part of it. Reports, dashboards, troubleshooting and auditing are based on it.

★ The current state is what the database holds right now: every change committed so far.

2.13 Valid Database State

DEFINITION 2.13 — VALID STATE

A **valid database state** is one that satisfies **all** integrity constraints and business rules defined in the schema. The DBMS must move the database only from one valid state to another.



SQL valid_state.sql — uses tables: univ

```sql
INSERT INTO Student VALUES (1001, 'Diya', 'diya@clg.edu', 1);   -- duplicate PK
INSERT INTO Student VALUES (1004, NULL, 'x@clg.edu', 1);        -- NULL name
INSERT INTO Student VALUES (1005, 'Myra', 'myra@clg.edu', 9);   -- dept 9 missing
INSERT INTO Course VALUES ('PHY101', 'Physics', 12);            -- credits out of range
INSERT INTO Student VALUES (1006, 'Kiran', 'kiran@clg.edu', 2); -- valid
SELECT StudentID, Name, DeptID FROM Student;
```

▶ Result (SQLite)

```
Error (line 1): UNIQUE constraint failed: Student.StudentID
Error (line 2): NOT NULL constraint failed: Student.Name
Error (line 3): FOREIGN KEY constraint failed
Error (line 4): CHECK constraint failed: Credits BETWEEN 1 AND 6
```

StudentID	Name	DeptID
1001	Aarav	1
1002	Meera	2
1006	Kiran	2

Screen 2.3 Four inserts would lead to an invalid state, so the DBMS rejects each one with the constraint it breaks: entity integrity (primary key), NOT NULL, referential integrity (foreign key) and a domain rule (CHECK). Only the valid row is stored.

REAL-LIFE VIEW — COLLEGE RULES

An enrollment cannot reference a student who does not exist; a grade must lie in 0–100; a student cannot enroll twice in the same course in the same term. Only when every such rule holds is the database in a valid state.

EXAM POINT

Entity, referential and domain constraints are essential for validity. The DBMS checks them **automatically** on insert, update and delete. An invalid state leads to wrong results and failures.

★ **A valid state satisfies every constraint of the schema, and the DBMS refuses any change that would break one.**

2.14 Schema versus Instance

Table 2.11 Schema versus instance

Aspect	Schema	Instance
Meaning	Structure or blueprint of the database	Actual data stored at a specific time
Nature	Logical design	Real data
Changes	Infrequently	Frequently (insert, update, delete)
Contains	Tables, attributes, relationships, constraints	Rows in the tables
Defined by	Database designer / DBMS	Users and applications
Operations	DDL: CREATE, ALTER, DROP	DML: INSERT, UPDATE, DELETE
Example	Student(StudentID, Name, DeptID, Email)	(101, Aarav, D01, aarav.sharma@example.com)
Permanence	Persistent, long-term	A snapshot at a point in time

MEMORY TRICK

Building plan vs the people inside: the plan (schema) is drawn once; the occupants (instance) change daily. Programming view: the schema is the **variable declaration**, the instance is its **current value**.

★ **Schema is the design; instance is the content at a given time.**

2.15 Intension and Extension

DEFINITION 2.14 — INTENSION AND EXTENSION

The **intension** of a relation is its **schema**: name, attribute names, domains and constraints. The **extension** is the **set of tuples** present in the relation at a given time (its instance).

Table 2.12 Intension versus extension

Aspect	Intension (schema)	Extension (instance)
Level	Schema level	Instance level
Meaning	Definition of the relation	Actual tuples at a given time
Changes	Rarely; same for all instances	Often; different instances have different extensions
Content	Attribute names, domains, constraints	Values recorded in the tuples
Example	STUDENT(StudentID, Name, Dept, Email)	(101, Aarav Sharma, CSE, ...), (102, Meera Nair, ECE, ...)

EXAM POINT

Intension = schema (stable, helps design); **extension = instance** (changes, helps operation). One intension can have many extensions over time.

★ **Intension is the definition of a relation; extension is the set of rows it holds now.**

2.16 Metadata

DEFINITION 2.15 — METADATA

Metadata is **data about data**. It describes the structure, properties and characteristics of the data in a database, and does not contain user data itself.

What metadata describes

- Names of databases, tables and views.
- Column names, data types and sizes.
- Constraints: NOT NULL, UNIQUE, CHECK.
- Primary keys, foreign keys, indexes, relationships.
- Owners, privileges, creation and modification dates.

Why it matters

- Explains the database structure.
- Guides correct and efficient queries.
- Supports administration and maintenance.
- Enables security and access control.
- Integration and documentation.

EVERYDAY ANALOGY

A book's text, images and chapters are the data. Its title, author, publisher, ISBN, category and page count in the library catalogue are the metadata.

EXAM POINT

Metadata is stored in the **system catalog / data dictionary** and used by the DBMS for validation, optimisation and security.

★ **Metadata is data about data: names, types, keys, constraints, owners and dates.**

2.17 Data Dictionary

DEFINITION 2.16 — DATA DICTIONARY

A **data dictionary** is a **centralised repository of metadata**: definitions and descriptions of all database objects such as tables, columns, data types, constraints, defaults, keys, relationships and user privileges, with their business meaning.

Table 2.13 Sample data-dictionary entry — STUDENT

Attribute	Type	Len	Key	Nulls	Default	Description
StudentID	INT	10	PK	NO		Unique identifier
FirstName	VARCHAR	50		NO		First name
Email	VARCHAR	100	UQ	YES	NULL	Email address
AdmissionDate	DATE			NO	CURRENT_DATE	Date of admission
DepartmentID	INT	10	FK	NO		References DEPARTMENT

1 Functions

- Documentation.
- Standardised naming.
- Maintenance.

2 More functions

- Impact analysis of changes.
- Communication.
- Auditing and compliance.

3 Who consults it

- Developers: correct definitions.
- DBAs: structure, changes, permissions.
- Analysts: meaning of data, reports.

EXAM POINT

A data dictionary stores **metadata, not data**, and is the single source of truth about the database. An **active** dictionary is updated automatically by the DBMS; a **passive** one is separate documentation that must be updated by hand.

★ A data dictionary is the central reference of every object's definition and meaning.

2.18 System Catalog

DEFINITION 2.17 — SYSTEM CATALOG

The **system catalog** is the collection of tables **maintained by the DBMS itself** that store metadata about every database object: tables, columns, indexes, constraints, views, users, roles, privileges, storage details and statistics.

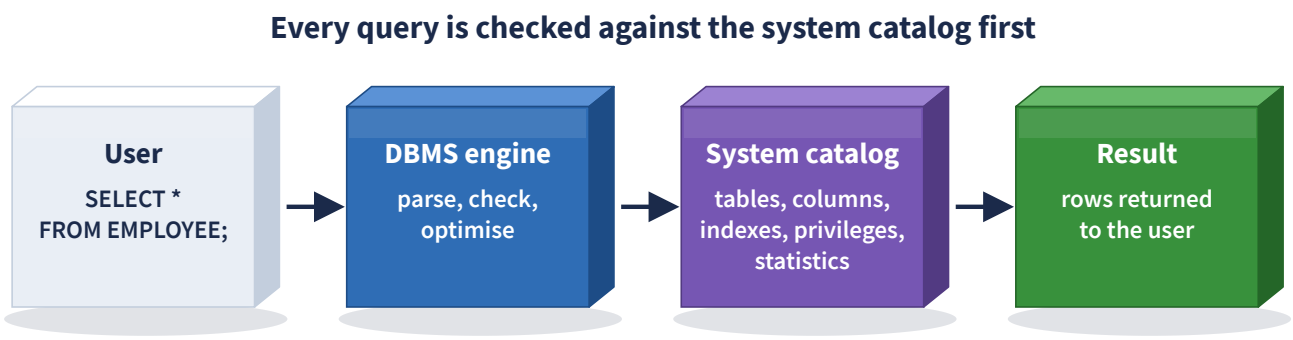


Figure 2.2 Before running `SELECT * FROM EMPLOYEE`, the DBMS consults the catalog to find the table, its columns, the user's privileges and the statistics needed to pick a plan.

SQL system_catalog.sql — uses tables: univ

```sql
CREATE VIEW vw_student_dept AS SELECT s.Name, d.DeptName FROM Student s JOIN Department d
USING (DeptID);
CREATE INDEX idx_enroll_course ON Enrollment(CourseID);
SELECT type, name, tbl_name FROM sqlite_master ORDER BY type, name;
SELECT "table", "from" AS fk_column, "to" AS refers_to FROM
pragma_foreign_key_list('Enrollment');
```

▶ Result (SQLite)

type	name	tbl_name
index	idx_enroll_course	Enrollment
index	sqlite_autoindex_Course_1	Course
index	sqlite_autoindex_Department_1	Department
index	sqlite_autoindex_Enrollment_1	Enrollment
index	sqlite_autoindex_Student_1	Student
index	sqlite_autoindex_Student_2	Student
table	Course	Course
table	Department	Department
table	Enrollment	Enrollment
table	Student	Student
view	vw_student_dept	vw_student_dept

table	fk_column	refers_to
Course	CourseID	CourseID
Student	StudentID	StudentID

Screen 2.4 SQLite's catalog table `sqlite_master` lists every table, view and index (including the automatic indexes behind PRIMARY KEY and UNIQUE). A second catalog query lists the foreign keys of Enrollment. The catalog is read with ordinary SQL.

Table 2.14 Why the DBMS consults the catalog

Task	Use of the catalog
Query processing	finds object structures to parse and execute queries
Optimisation	uses statistics to choose the cheapest execution plan
Security	checks privileges and permissions
Object management	creates, alters and drops objects
Integrity checking	validates constraints and relationships

Data dictionary

- General term for any metadata repository.
- May be a separate documentation tool.
- Used mainly by people.

System catalog

- The DBMS's **internal** data dictionary.
- Always up to date and DBMS-controlled.
- Users read it but do not modify it directly.

EXAM POINT

Catalog names in real systems: Oracle `USER_TABLES` , `ALL_TAB_COLUMNS` ; SQL standard `INFORMATION_SCHEMA` ; PostgreSQL `pg_catalog` ; SQLite `sqlite_master` . The **DDL compiler** writes into the catalog. Codd's Rule 4: the catalog is itself relational and queried with the same language.

★ **The system catalog is the DBMS's own always-current metadata store, read with SQL.**

Chapter 2 summary

- **Schema** = structure (tables, attributes, relationships, constraints), defined with DDL, stable; shown by a **schema diagram**.
- Schema levels: **external** (many user views), **conceptual** (one community view), **internal** (one storage description).
- **Logical schema** = DBMS view of tables and keys; **physical schema** = files, indexes, partitions, clustering.
- **Instance / state** = data at a moment. Initial state, current state, valid state; ROLLBACK leaves the state unchanged.
- A **valid state** satisfies entity, referential and domain constraints; the DBMS rejects changes that break them.
- **Intension** = schema; **extension** = instance.
- **Metadata** = data about data, kept in the **data dictionary**; the **system catalog** is the DBMS's internal, always-current dictionary.

The three schema levels met here (external, conceptual, internal) are the backbone of the ANSI/SPARC **three-schema architecture**, the subject of Chapter 3.

CHAPTER · UNIT II · DATABASE CONCEPTS

3 Three-Schema Architecture

OPENING SITUATION

In a restaurant the customer reads a **menu**, the chef works from the complete list of **dishes and recipes**, and the store room holds the **raw ingredients** on shelves. The customer never sees the shelves, and the store keeper can rearrange the shelves without reprinting the menu. A database is organised the same way in three levels — **external** (what each user sees), **conceptual** (what the whole database contains) and **internal** (how it is stored) — joined by two **mappings**.

IN THIS CHAPTER

- 3.1 Three-Schema Architecture
- 3.2 Introduction and ANSI/SPARC Origin
- 3.3 Need for Three-Schema Architecture
- 3.4 External Level
- 3.5 External Schema
- 3.6 User Views and Subschemas
- 3.7 Conceptual Level
- 3.8 Global Logical View
- 3.9 Internal Level
- 3.10 Internal Schema
- 3.11 Physical Storage Structure
- 3.12 External–Conceptual Mapping
- 3.13 Conceptual–Internal Mapping
- 3.14 Data Abstraction
- 3.15 View-Level Abstraction
- 3.16 Logical-Level Abstraction
- 3.17 Physical-Level Abstraction
- 3.18 Advantages of Three-Schema Architecture
- 3.19 Limitations of Three-Schema Architecture

YOU NEED

- Chapter 2: external, conceptual and internal schemas; schema vs instance.

3.1 Three-Schema Architecture

DEFINITION 3.1 — THREE-SCHEMA ARCHITECTURE

The **three-schema architecture** is the **ANSI/SPARC** framework that describes a database at **three levels** — **external**, **conceptual** and **internal** — and separates how users view data from how data is logically organised and how it is physically stored.

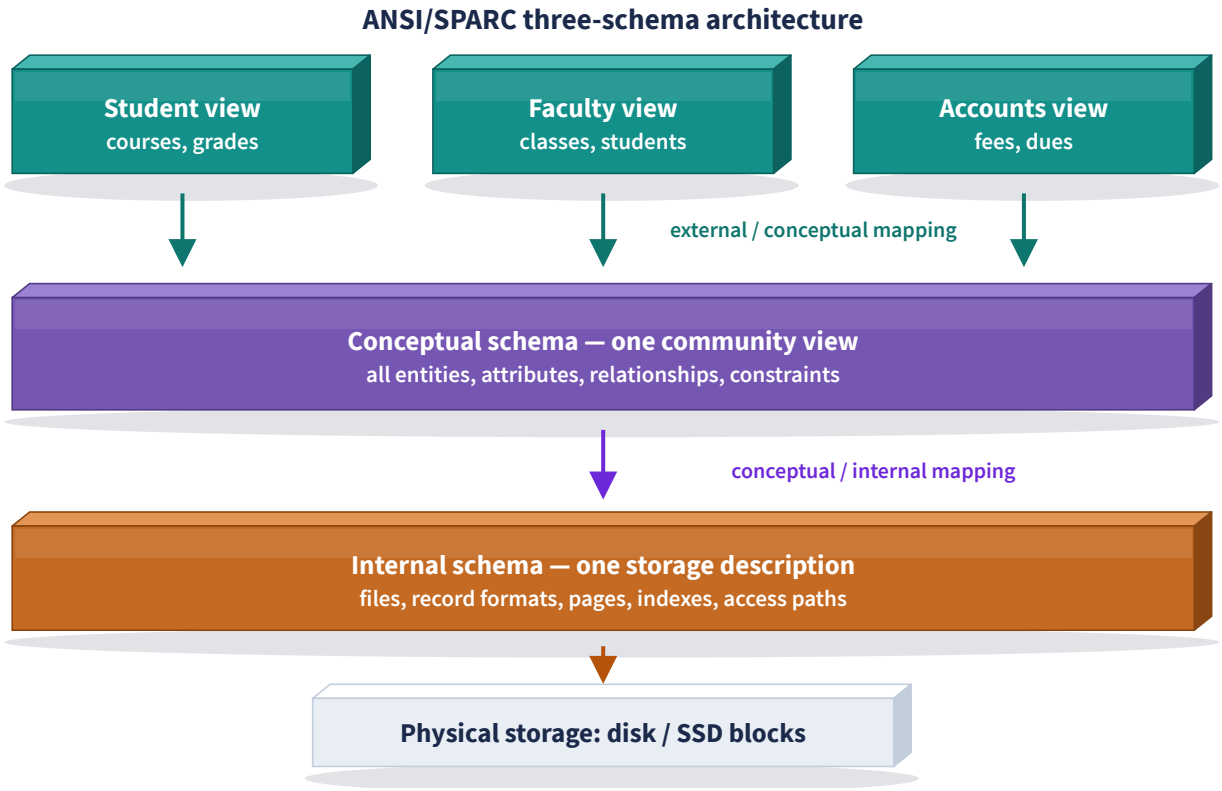


Figure 3.1 Many external schemas sit on one conceptual schema, which sits on one internal schema. The two mappings between the levels are what make data independence possible.

Table 3.1 The three levels in a college database

Level	Who uses it	What it shows	Example	Mapping
External (view)	Students, faculty, librarians, admin	Only the data relevant to each user	Student: courses, grades · Faculty: courses, salary · Librarian: books, issues	External/conceptual: translates each view to the conceptual schema
Conceptual (logical)	DBA, designers, all applications	Complete logical structure and constraints	Tables Student, Course, Faculty, Book, Issue, Department	Conceptual/internal: maps the conceptual schema to physical storage
Internal (physical)	DBA, system administrators	How data is stored: record formats, access paths	Files, indexes, disk blocks	

1 Abstraction

- Hides complexity from users.
- Users focus on data, not storage (**separation of concerns**).

2 Multiple views

- Different users see only what they need.
- **Security**: sensitive data hidden from unauthorised users.

3 Data independence

- A change at one level does not affect the other levels.
- Achieved through the **mappings**.

★ **Three levels (external, conceptual, internal) + two mappings = abstraction, multiple views, security and data independence.**

3.2 Introduction and ANSI/SPARC Origin

The architecture was proposed by the **ANSI/X3/SPARC** Study Group on DBMS (**S**tandards **P**lanning **a**nd **R**equirements **C**ommittee of the American National Standards Institute) — interim report **1975**, final report **1978**. Its goal was a systematic framework that supports **data independence** and **multiple user views**. It is a **reference model**: it describes how a database *should* be described, not the code of any one product.

Table 3.2 The three schemas at a glance

Level	Describes	How many
External (view)	How one user or application sees the database; shows the required data and hides the rest	**Many** external schemas
Conceptual (logical)	The whole database as one logical structure: what is stored and the relationships among data	**One** conceptual schema
Internal (physical)	How data is actually stored: file organisation, indexes, access paths	**One** internal schema

REAL-LIFE VIEW — UNIVERSITY

Admin view: all student, faculty and course details for management. **Registrar view**: enrolments, courses, schedules. **Faculty view**: their subjects, students and grades. **Student view**: personal profile, enrolled courses, grades. All four are different external schemas built on the **same** conceptual schema, stored once in data files, indexes, logs and backups.

EXAM POINT

Main objectives: **simplicity**, **abstraction**, **multiple views**, **data independence**. Two kinds of independence follow: **logical** (conceptual changes do not affect external schemas) and **physical** (internal changes do not affect conceptual and external schemas) — Chapter 4.

★ **ANSI/SPARC (1975–78) proposed the three-level reference model for data independence and multiple views.**

3.3 Need for Three-Schema Architecture

Main idea. Without layers, every user and program works directly on one single description of the data, which is large, exposes storage details and breaks whenever anything changes. The layered approach hides complexity, gives each user a customised view, secures data and makes design and maintenance easier.

Without layers (one level)

- **High complexity**: everyone sees the entire database.
- **Poor flexibility**: any structural change affects all users and programs.
- **Difficult maintenance**: every program must be modified.
- **Exposed physical details**: risk of misuse.
- **Limited customisation**: only one view possible.

With three schemas

- **Data abstraction**: users see only what they need.
- **Multiple user views**: one external schema per role.
- **Security**: sensitive data and storage hidden.
- **Easier design**: one global logical structure.
- **Simpler maintenance** and **data independence**.

Table 3.3 One college database, three views

View	Sees only	Does not see
Student view	Course name, grade, fee status	Salaries, other students' data
Faculty view	Course name, enrolled students, grades	Fee and payment details
Accounts view	Fee details, payments, dues	Grades, teaching details

★ The DBMS needs three levels to hide complexity, personalise access, secure data and allow change.

3.4 External Level

DEFINITION 3.2 — EXTERNAL LEVEL

The **external level** is the **user-level view** of the database. It shows only the data relevant to a particular user or application and hides the rest. It is the **highest** level of the architecture and the one **closest to end users**; it is the interface between the end user and the conceptual level.

Characteristics

- **Many external views** for different users and applications.
- Shows only relevant data; **hides the rest**.
- Supports **security** (limits access) and **simplicity** (easy interface).

Benefits

- Security by hiding unnecessary information.
- Simpler, faster access for end users.
- Supports users with different requirements.
- Reduces the complexity users face.

Table 3.4 External views over one conceptual level

Student view	Faculty view	Accounts view
StudentID, Name, CourseName, Grade	FacultyID, Name, Department, CourseName	StudentID, FeesPaid, Balance, PaymentDate
student-related data only	faculty and teaching details	payment and financial details only

★ The external level is the highest, user-closest level: many customised views, the rest hidden.

3.5 External Schema

DEFINITION 3.3 — EXTERNAL SCHEMA

An **external schema** is the **formal description of one particular user view** (or **subschema**) at the external level. It states which data a user or application may see and how it is presented.

Table 3.5 What an external schema contains

Part	Meaning
Selected attributes	Only the columns relevant to the user or application
Allowed records	Only the rows the user is authorised or needs to see
Presentation format	How the data is arranged, grouped or presented
Security restrictions	Hides sensitive or unnecessary data; enforces access control

In SQL an external schema is written as a **view**: a named query over the conceptual tables. Screen 3.1 builds two external schemas for the college.

SQL external_schema.sql — uses tables: college3

```sql
CREATE VIEW student_view AS
  SELECT s.StudentID, s.Name, g.CourseName, g.Grade
  FROM Student s JOIN Grade g ON g.StudentID = s.StudentID;
CREATE VIEW accounts_view AS
  SELECT StudentID, FeesPaid, Balance FROM Fee WHERE Balance > 0;
SELECT * FROM student_view;
SELECT * FROM accounts_view;
```

▶ Result (SQLite)

StudentID	Name	CourseName	Grade
101	Ananya	DBMS	A
101	Ananya	OS	B
102	Bharat	Signals	A
103	Charu	DBMS	B

StudentID	FeesPaid	Balance
102	30000	20000
103	45000	5000

Screen 3.1 Two external schemas over the same conceptual tables. `student_view` selects attributes (no phone number, no salary); `accounts_view` also restricts the allowed records (only students with dues).

External level

- A **level** of the architecture; a logical layer.
- Holds many views.
- The interface between users and the conceptual schema.

External schema

- The **actual description** of one view.
- Each view has its own external schema.
- Defines what is visible and how it is seen.

★ **An external schema is the formal description of one user view: chosen columns, allowed rows, format, security.**

3.6 User Views and Subschemas

DEFINITION 3.4 — USER VIEW AND SUBSCHEMA

A **user view** is a customised way a user sees the data. A **subschema** is a **subset of the conceptual schema** tailored to a user or application. Both show only relevant data and hide the rest.

Table 3.6 User view vs subschema

Aspect	User view	Subschema
Definition	A customised view of the database from the user's perspective	A subset of the conceptual schema tailored to a user or application
Scope	May be dynamic, created as needed	Usually defined at design time and used by applications
Purpose	Personalised access, simpler work	Supports a specific user group or application
Example	A student portal showing grades and courses	An accounts subschema with payments and dues



EXAM POINT

The term **subschema** comes from the CODASYL network model, where an application program declared the subschema it used. Benefits of views and subschemas: **simplicity, privacy, security, task-focused access**.

★ **User views and subschemas are role-specific subsets derived from the one conceptual schema.**

3.7 Conceptual Level

DEFINITION 3.5 — CONCEPTUAL LEVEL

The **conceptual level** is the **community (organisation-wide) logical view** of the whole database. It describes **what** data is stored, the **relationships** among data and the **rules** that govern it, independent of physical storage.

It defines

- **Entities**: Student, Course, Faculty, Department.
- **Attributes**: StudentID, Name, Email.
- **Relationships**: a student ENROLLS in a course; a course is TAUGHT_BY a faculty.
- **Constraints**: unique keys, mandatory fields, Credits > 0.

Characteristics

- Complete logical structure for the organisation.
- Not influenced by physical storage.
- One **shared** description for all users.
- Basis for deriving external schemas and the internal schema.

Table 3.7 Conceptual schema of a college (relationships and rules)

Entity	Attributes	Rule
STUDENT	**StudentID**, Name, Email, Phone, DOB, DepartmentID (FK)	Email unique
COURSE	**CourseID**, CourseName, Credits, DepartmentID (FK)	Credits > 0; taught by exactly one faculty
FACULTY	**FacultyID**, Name, Email, Phone, Designation, DepartmentID (FK)	Email unique
DEPARTMENT	**DepartmentID**, DepartmentName, OfficeLocation	Has one or more courses and faculties

WHY IT SITS IN THE MIDDLE

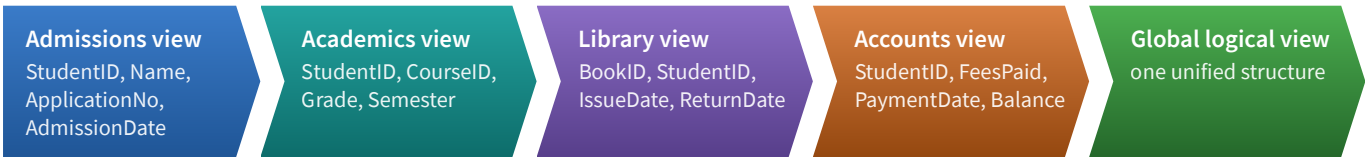
Above the external level because it includes all data for the organisation, not just one user's portion. **Above the internal level** because it ignores files, indexes and storage structures. It is the **bridge** that turns user requirements into a logical structure that can be implemented physically. When the college adds a department, only the conceptual schema changes logically; if storage technology changes, the conceptual schema stays the same.

★ **The conceptual level is the single, DBMS-independent, organisation-wide logical view.**

3.8 Global Logical View

DEFINITION 3.6 — GLOBAL LOGICAL VIEW

The **global logical view** is the **unified overall logical representation** of the entire database for the organisation. It **integrates** the local views of all departments into one consistent whole and is closely associated with the conceptual schema.



User views

- Show only the data relevant to one user or application.
- Many and customised.
- Partial.

Global logical view

- Combines **all** data and relationships into one structure.
- One and organisation-wide.
- Integrates views across departments and functions.

EXAM POINT

Advantages: **consistency** (uniform definitions and rules), **integration**, **reduced redundancy** (duplicates such as StudentID in four views are identified and stored once) and **enterprise-wide understanding**. Building the global view by merging user views is called **view integration**.

★ **The global logical view is the integrated, organisation-wide logical picture built from all user views.**

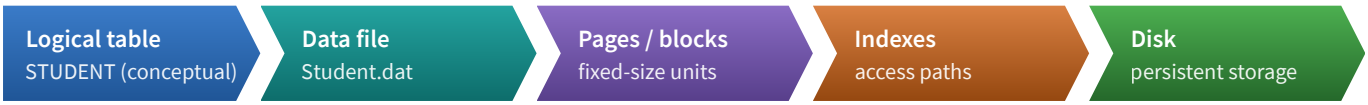
3.9 Internal Level

DEFINITION 3.7 — INTERNAL LEVEL

The **internal level** is the **lowest** level of the architecture. It describes **how data is actually stored** on secondary storage: files, record formats, blocks/pages, indexes, access paths and storage structures. It is the DBMS's own view and is **not visible to users**.

Table 3.8 Contents of the internal level

Item	Meaning
Files	Physical files that store records (e.g. `Student.dat`)
Record formats	How the fields of a record are arranged on disk
Indexes	Structures that speed up retrieval (index on StudentID, on Department)
Blocks / pages	Fixed-size storage units; a page may hold, say, 4 records of STUDENT
Access paths	Methods to locate records: index lookup, hashing, sequential scan
Storage structures	Heap files, sorted files, B+ trees, hashing



EXAM POINT

Benefits of a good internal level: **performance**, **space optimisation**, **scalability** (partitioning, indexing) and **reliability** (blocks, backups, recovery). Changes here do not affect the conceptual or external levels — **physical data independence**.

★ **The internal level is the hidden, lowest level: files, records, pages, indexes and access paths.**

3.10 Internal Schema

DEFINITION 3.8 — INTERNAL SCHEMA

The **internal schema** is the **description** of the internal level: the storage structures, files, indexes, access paths, record formats, block size, and pointer and space management used for the database.

Table 3.9 STUDENT described at two levels

Level	Description
Conceptual (logical view)	Table STUDENT with attributes RollNo, Name, Dept, Age, CGPA
Internal schema (physical view)	File `STUDENT.DAT` (heap file of 4 KB data pages); index `IDX_STUDENT_RollNo` (B+ tree); record format [RollNo | Name | Dept | Age | CGPA] + NULL indicators; pages linked by pointers

Storage-level details

- **Files**: heap or organised files.
- **Indexes**: B+ tree, hash.
- **Access paths**: full scan, index scan, range scan.
- **Record format**: fixed or variable length, NULL indicators.
- **Blocks**: fixed size, e.g. 4 KB or 8 KB.

Why users do not see it

- It is complex and system-dependent.
- Hiding it protects users from physical detail.
- Storage can change without affecting applications.
- Search `RollNo = 1045`: the index jumps straight to the right page.

★ **The internal schema connects the logical world of tables with the physical world of files and pages.**

3.11 Physical Storage Structure

DEFINITION 3.9 — PHYSICAL STORAGE STRUCTURE

The **physical storage structure** is the way data is actually kept on **secondary storage** (HDD, SSD): files made of **records**, records made of **fields**, records grouped into fixed-size **blocks/pages** written to disk sectors, and **indexes and pointers** that give fast access paths. The DBMS manages space allocation and retrieval.

WORKED EXAMPLE — HOW MANY PAGES?

STUDENT has **1,00,000** records of **100 bytes** each; page size = **4 KB** (≈ 4000 bytes usable).

Records per page = 4000 / 100 = **40** · Pages needed = 1,00,000 / 40 = **2,500 pages**.

A sparse index on RollNo keeps **one entry per page** (the first key of the page): key 1001 → page 1, 3041 → page 76, 5001 → page 125 (RollNo 5001–5040). To find RollNo 5017 the DBMS searches the index, jumps to page 125 and reads one page instead of 2,500.

Table 3.10 File organisations

Organisation	Idea and trade-off
Heap (unordered)	Records inserted anywhere; insertion simple, search slower (full scan)
Sequential (ordered)	Records kept in key order; good for range queries
Clustered	Records with related keys stored together on the same pages
Encrypted / compressed	Improves security and saves space, at some CPU cost

★ **Physical storage = files → records → fields, packed into pages on disk, reached through indexes and pointers.**

3.12 External–Conceptual Mapping

DEFINITION 3.10 — EXTERNAL/CONCEPTUAL MAPPING

The **external/conceptual mapping** defines the correspondence between **each external schema** and the **conceptual schema**: how every attribute of a view is derived from the logical database. User requests on a view are translated through it into requests on the conceptual schema, and results are returned in the view's format.



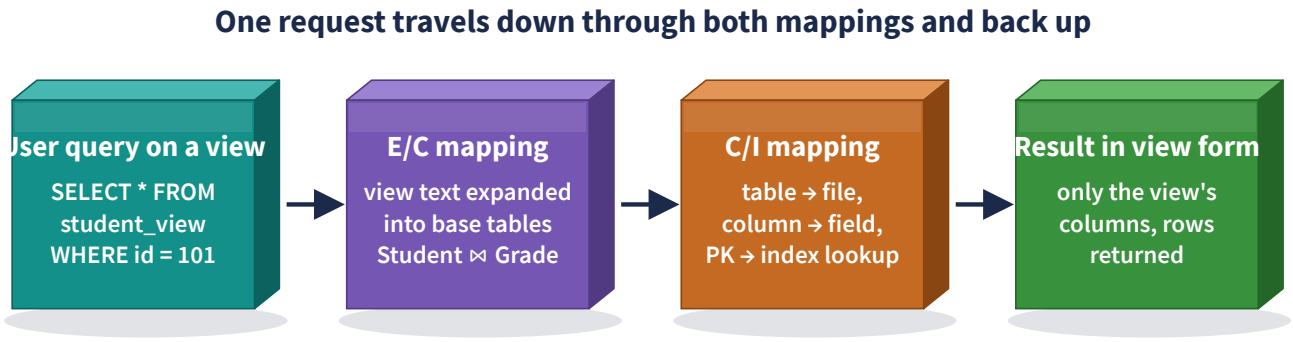


Figure 3.2 A query on `student_view` is rewritten by the external/conceptual mapping into a join of base tables, and the conceptual/internal mapping turns that into file and index operations.

In SQL the mapping of a view **is its stored definition**. Screen 3.2 reads that mapping from the catalog, then changes the conceptual schema and shows the view still works.

SQL ec_mapping.sql — uses tables: college3

```sql
CREATE VIEW student_view AS
  SELECT s.StudentID, s.Name, g.CourseName, g.Grade
  FROM Student s JOIN Grade g ON g.StudentID = s.StudentID;
SELECT name, substr(replace(sql, char(10), ' '), 1, 58) || ' ...' AS stored_mapping
  FROM sqlite_master WHERE type = 'view';
ALTER TABLE Student ADD COLUMN BloodGroup TEXT;   -- conceptual change
SELECT * FROM student_view WHERE StudentID = 101;  -- view unchanged
```

▶ Result (SQLite)

name	stored_mapping
student_view	CREATE VIEW student_view AS SELECT s.StudentID, s.Name, ...

StudentID	Name	CourseName	Grade
101	Ananya	DBMS	A
101	Ananya	OS	B

Screen 3.2 The mapping is stored in the catalog as the view's SQL text. Adding `BloodGroup` to STUDENT changes the conceptual schema, yet the external schema and every program using it keep working: **logical data independence**.

Table 3.11 External schema vs conceptual schema

Aspect	External schema (user view)	Conceptual schema (logical level)
Scope	One user or application	The entire database for the community
Content	Subset of data as required	Complete logical structure and constraints
Focus	What a user needs to see	What data exists and how it relates
Changes	Can change without affecting others	Absorbed by updating mappings (logical independence)
Benefit	Personalised, simple, secure	Consistency, integration, independence

COMMON MISTAKE 3.1 — "A VIEW IS A COPY OF THE DATA"

An ordinary view stores **no rows**; it stores only its definition, which is the mapping. Every query on the view is rewritten against the base tables, so the view always shows current data. Only a **materialized view** (Oracle, PostgreSQL) keeps a stored copy that must be refreshed.