Lecture2 – Relational Models and Database Schema

ICDT 1202Y – Database Systems

LEARNING OUTCOME

After this session you will be able to :

- Differentiate between the various data models
- Explain the properties of a relation
- Define relational keys
- Explain relational contraints
- Differentiate between database instances and schemas
- Describe a two tier and three tier architecture

DATA MODELS

- A ***data model*** is a collection of high level description concepts for describing data. It hides many low-level storage details
- It is a set of concepts to describe the *structure* of a database, and certain *constraints* that the database should obey.
- A ***schema*** is a description of a particular collection of data, using the given data model

CATEGORIES OF DATA MODELS

- **Conceptual (high-level, semantic) data models**:

Provides concepts that are close to the way many users perceive data.

- Uses concepts such as *entities*, *attributes* and *relationships*.
- An ***entity*** represents a real-world object or concept e.g student or lecturer
- An ***attribute*** represents some property of interest that further describes an entity.
- A ***relationship*** among two entities represents an association between two entities e.g. teaches relationship.

CATEGORIES OF DATA MODELS

- **Physical** (**low-level**, **internal**) data models:

– Provide concepts that describe details of how data is stored in the computer.

- **Implementation** (**representational**) data models:

– Provide concepts that fall between the above two, balancing user views with some computer storage details.

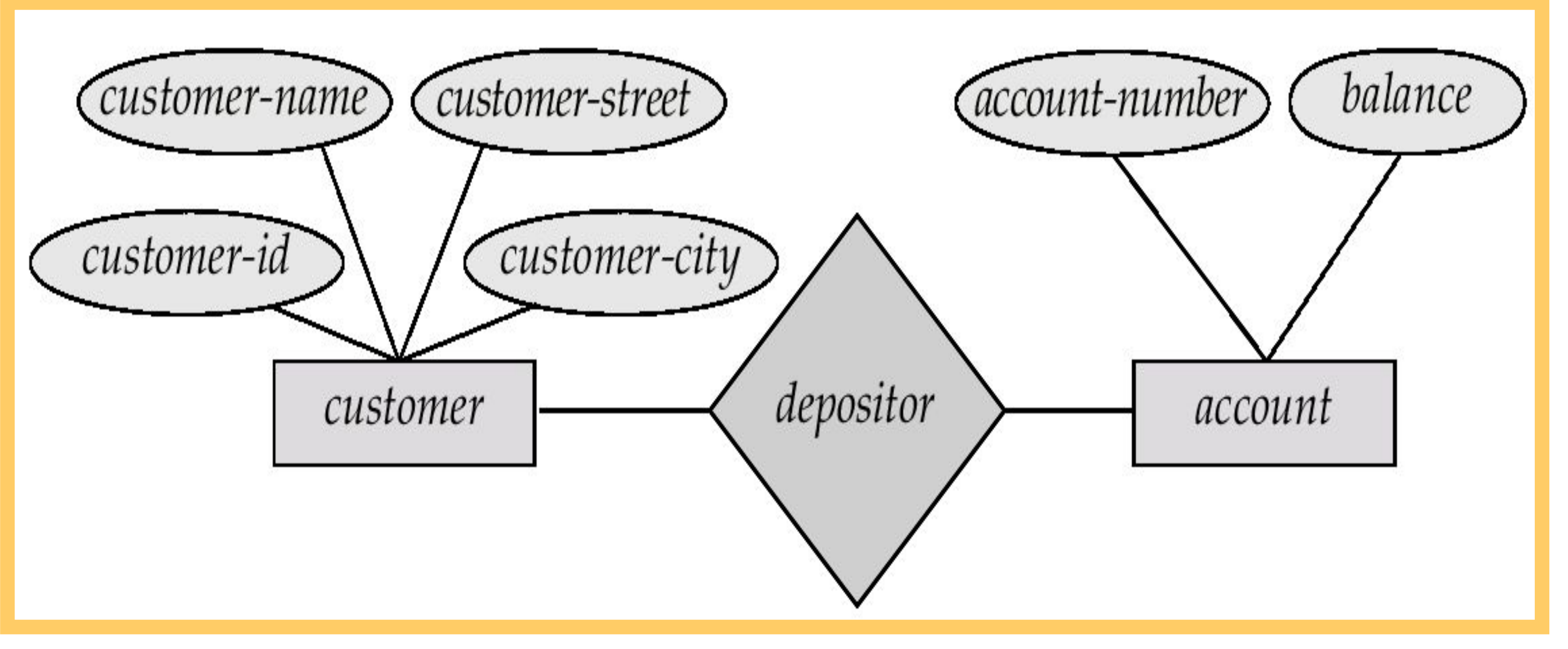
– These include the widely-used relational data model and network and hierarchical data models.

ENTITY RELATIONSHIP MODEL

- E-R model of real world
  - Entities (objects)
    - E.g. customers, accounts, bank branch
  - Relationships between entities
    - E.g. Account A-101 is held by customer Johnson
    - Relationship set *depositor* associates customers with accounts
- Widely used for database design
  - Database design in E-R model usually converted to design in the relational model (coming up next) which is used for storage and processing

ENTITY RELATIONSHIP MODEL

- Example of schema in the entity- relationship model



RELATIONAL MODEL

- A simple data model: the *Relational Data Model* - data stored in relations (tables)
- Schema: *RelationName(field1 : type1 , ..., fieldn : typen)*
- A declarative query language: SQL
  - SELECT balance
  - FROM account
  - WHERE branch = 'Springfield'
- Programmer specified what answers a query should return, but not how the query is executed
- DBMS picks the best execution strategy
- Provides physical data independence (applications need not need to worry about how data is physically structured and stored)

RELATIONAL MODEL

- The relational database model achieves structural independence - any type of association be it one-to-one, one-to-many, many-to-many can be easily implemented with the relational model.
- The relational database model has a very powerful and flexible query capability.

RELATIONAL MODEL – EXAMPLE 2

- A relation of students:
  Students(sid : string, name : string, age : integer, gpa : real)
- An instance of the students relation can be represented as follows:

sid	*name*	*login*	*age*	*gpa*
53666	Jones	Jonescs	18	7.4
53668	Smith	smithee	18	7.8
53650	Smith	smithmath	19	6.4
53831	Madayan	madayan@music	11	8.0
53832	Guldy	guldu@music	12	2.0

RELATIONAL MODEL – EXAMPLE 2

- A sample relational database:

customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account-number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer-id	*account-number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

EXAMPLE RELATION

Attributes

Branch

Relation

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Cardinality

Degree

Primary key

Foreign key

Staff

Relation

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

EXAMPLE OF ATTRIBUTE DOMAINS

Attribute	Domain Name	Meaning	Domain Definition
branchNo	BranchNumbers	The set of all possible branch numbers	character: size 4, range B001–B999
street	StreetNames	The set of all street names in Britain	character: size 25
city	CityNames	The set of all city names in Britain	character: size 15
postcode	Postcodes	The set of all postcodes in Britain	character: size 8
sex	Sex	The sex of a person	character: size 1, value M or F
DOB	DatesOfBirth	Possible values of staff birth dates	date, range from 1-Jan-20, format dd-mmm-yy
salary	Salaries	Possible values of staff salaries	monetary: 7 digits, range 6000.00–40000.00

DEFINITIONS

Informal Terms		Formal Terms
Table		Relation
Column		Attribute/Domain
Row		Tuple
Values in a column		Domain
Table Definition		Schema of a Relation
Populated Table		Extension

RELATIONAL MODEL TERMINOLOGY

- A **relation** is a table with columns and rows.
  - Only applies to logical structure of the database, not the physical structure.
- An **attribute** is a named column of a relation.
- **Domain** is the set of allowable values for one or more attributes.
- **Tuple** is a row of a relation.
- **Degree** is the number of attributes in a relation.
- **Cardinality** is the number of tuples in a relation.
- **Relational Database** is a collection of normalized relations with distinct relation names.

PROPERTIES OF RELATIONS

- Relation name is distinct from all other relation names in relational schema.
- Each cell of relation contains exactly one atomic (single) value.
- Each attribute has a distinct name.
- Values of an attribute are all from the same domain.
- Each tuple is distinct; there are no duplicate tuples.
- Order of attributes has no significance.
- Order of tuples has no significance, theoretically.

RELATIONAL KEYS

- Superkey
  - An attribute, or set of attributes, that uniquely identifies a tuple within a relation.
- Candidate Key
  - Superkey (K) such that no proper subset is a superkey within the relation.
- Primary Key
  - Candidate key selected to identify tuples uniquely within relation.
- Alternate Keys
  - Candidate keys that are not selected to be primary key.
- Foreign Key
  - Attribute, or set of attributes, within one relation that matches candidate key of some (possibly same) relation.

REFERENTIAL INTEGRITY

- A constraint involving *two* relations (the previous constraints involve a *single* relation).
- Used to specify a *relationship* among tuples in two relations: the **referencing relation** and the **referenced relation**.
- Tuples in the *referencing relation* R_1 have attributes FK (called **foreign key** attributes) that reference the primary key attributes PK of the *referenced relation* R_2. A tuple t_1 in R_1 is said to **reference** a tuple t_2 in R_2 if $t_1[FK] = t_2[PK]$.
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from $R_1.FK$ to $R_2.PK$

RELATIONAL INTEGRITY CONTRAINTS

Statement of the constraint

The value in the foreign key column (or columns) FK of the **referencing relation** R_1 can be either:

(1) a value of an existing primary key value of the corresponding primary key PK in the **referenced relation** $R_{2,}$, or..

(2) a null.

In case (2), the FK in R_1 should not be a part of its own primary key.

SEMANTIC INTEGRITY CONTRAINTS

- Based on application semantics and cannot be expressed by the model per se
- E.g., "the max. no. of hours per employee for all projects he or she works on is 56 hrs per week"
- A *constraint specification language* may have to be used to express these

EXAMPLE SCHEMA

Figure 7.5 Schema diagram for the COMPANY relational database schema; the primary keys are underlined.

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

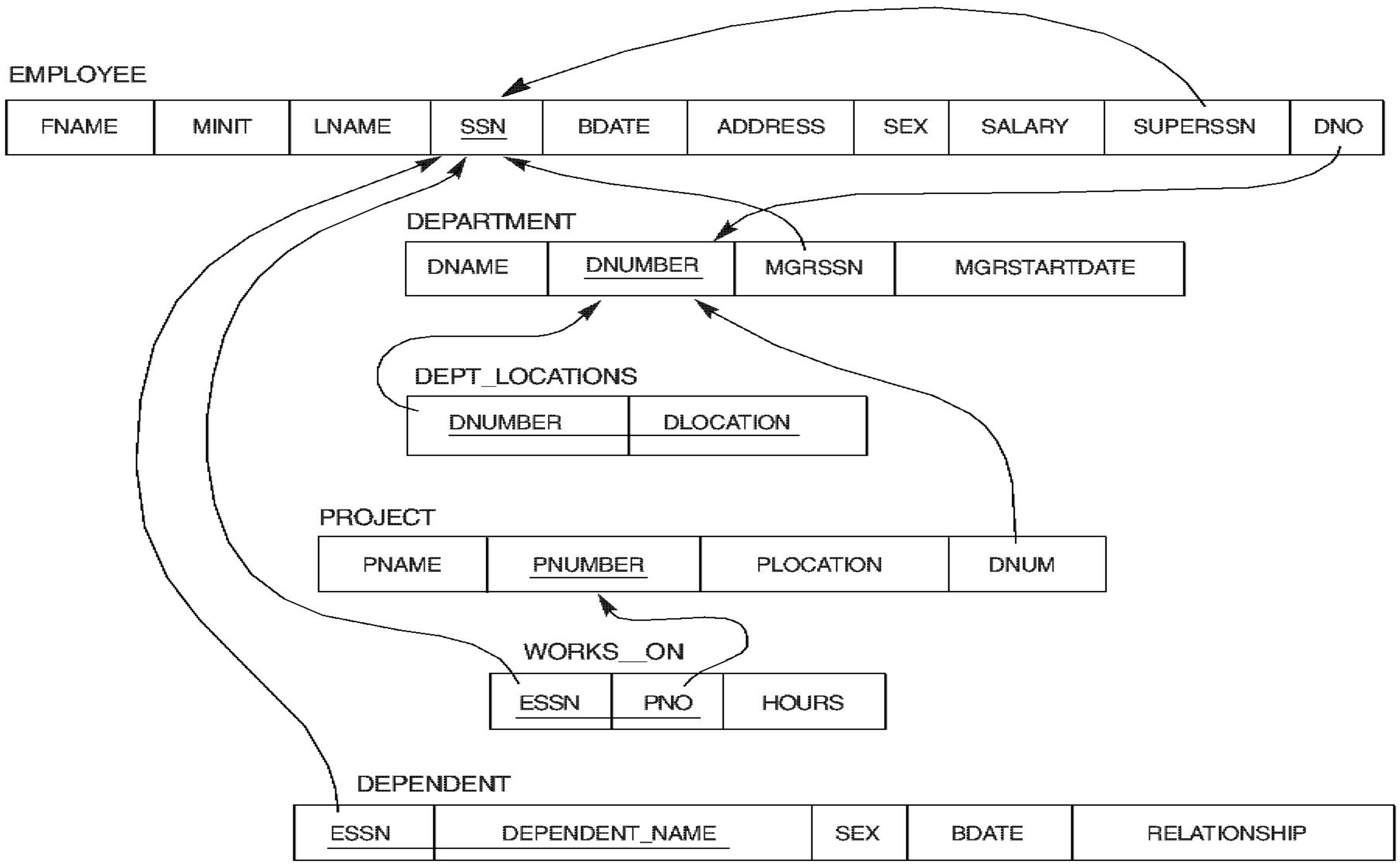
ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

RELATIONAL INTEGRITY CONTRAINTS

Figure 7.7 Referential integrity constraints displayed on the COMPANY relational database schema diagram.



Update Operations on Relations

- Update Operations
  - INSERT a tuple.
  - DELETE a tuple.
  - MODIFY a tuple.
- Integrity constraints should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may *propagate* to cause other updates automatically. This may be necessary to maintain integrity constraints.

UPDATE OPERATION ON RELATIONS

- In case of integrity violation, several actions can be taken:
  - Cancel the operation that causes the violation (REJECT option)
  - Perform the operation but inform the user of the violation
  - Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
  - Execute a user-specified error-correction routine

EXERCISE

- Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:
- STUDENT(<u>SSN</u>, Name, Major, Bdate)
- COURSE(<u>Course#</u>, Cname, Dept)
- ENROLL(<u>SSN</u>, <u>Course#</u>, <u>Quarter</u>, Grade)
- BOOK_ADOPTION(<u>Course#</u>, <u>Quarter</u>, Book_ISBN)
- TEXT(<u>Book_ISBN</u>, Book_Title, Publisher, Author)
- **Draw a relational schema diagram specifying the foreign keys for this schema.**

DATABASE SCHEMA

- **Database Schema**: The *description* of a database. Includes descriptions of the database structure and the constraints that should hold on the database.
- Can be represented using a diagram and that is known as a *schema diagram*.

- **Schema Diagram**: A diagrammatic display of (some aspects of) a database schema:
- Including, names of record types and data items and some types of constraints

DATABASE SCHEMA

- **Schema Construct**: A component of the schema or an object within the schema, e.g., STUDENT, COURSE.
- An example schema diagram:

Student

Name	StudentID	Course

Module

ModuleName	ModuleID	CreditHours	Department

Grade_Report

studentID	CourseID	Grade

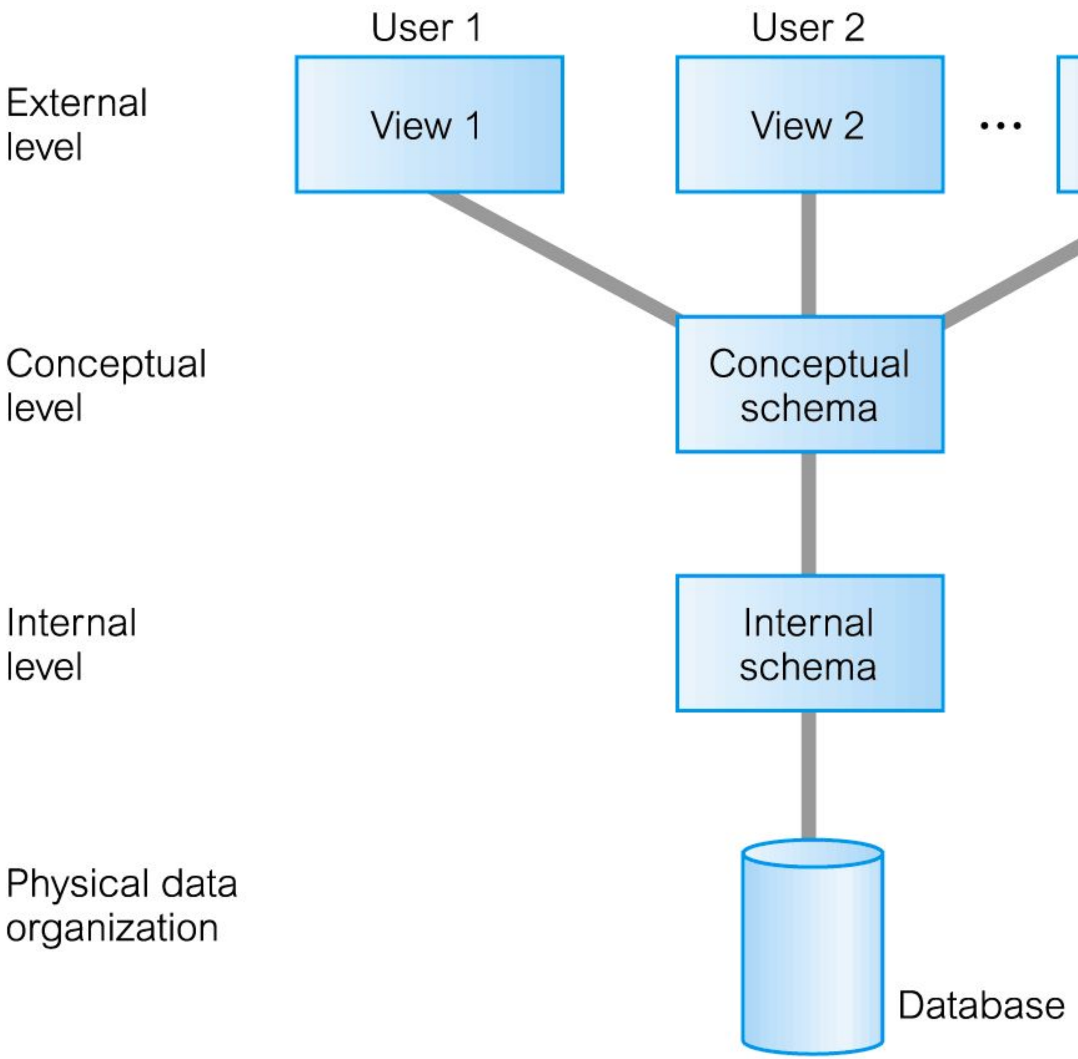
DATABASE INSTANCE/STATE

- **Database Instance/State**: The actual data stored in a database (the content of a database) at a *particular moment in time*.
- For a new database, we define its schema first. At that moment there is no data in the database and it is said to be in an *empty state*.
- **Initial Database State:** Refers to the database when it is loaded with initial data.
- **Valid State:** A state that satisfies the structure and constraints of the database.

DATABASE INSTANCE/STATE

- The **database schema** changes *very infrequently*.
- The **database state** changes *every time the database is updated.*
- **Schema** is also called **intension**, whereas **state** is called **extension**.

THREE-SCHEMA ARCHITECTURE



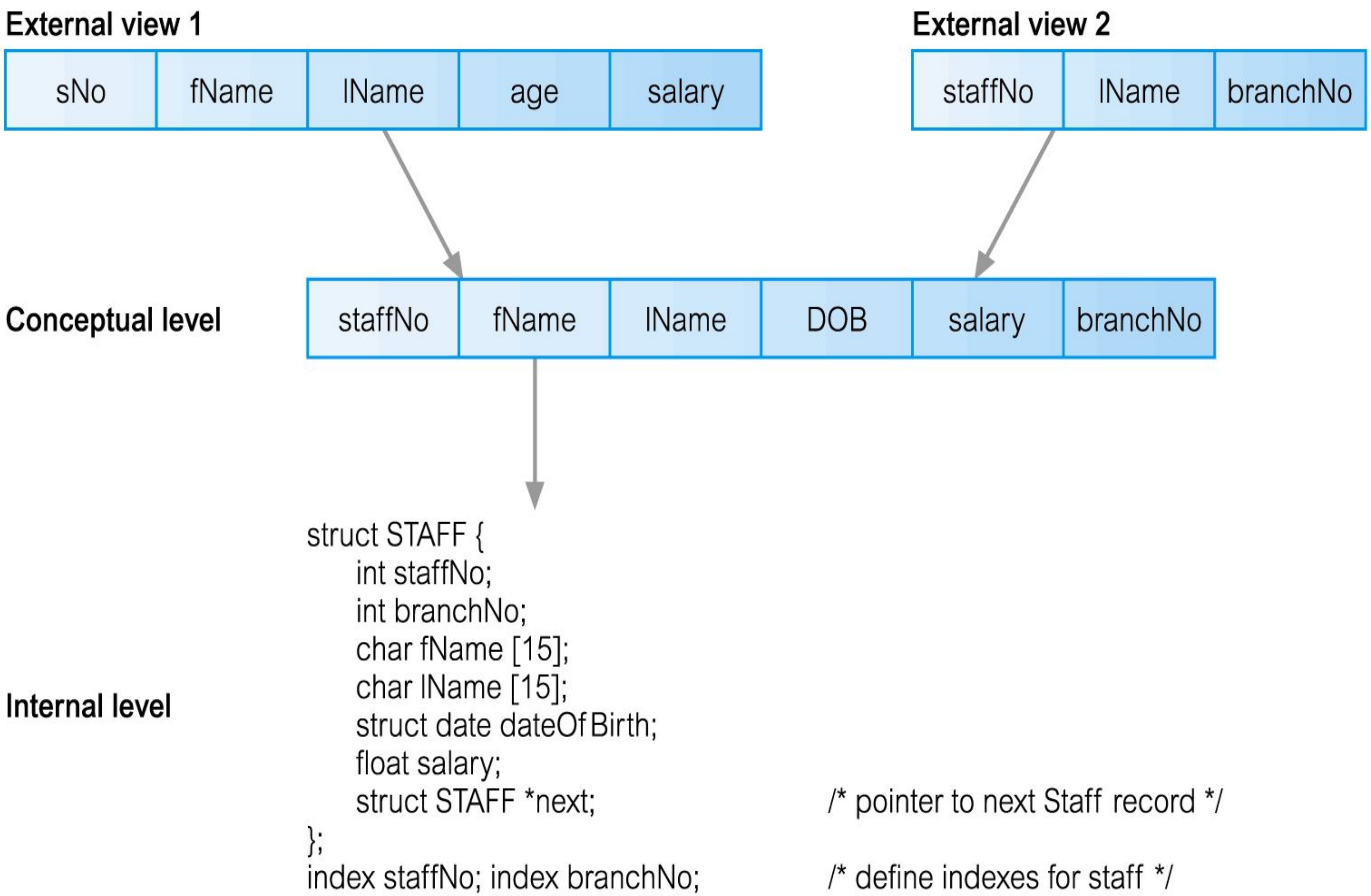
THREE-SCHEMA ARCHITECTURE

- Many *views*, single *conceptual (logical) schema* and *physical schema*
  - Views (External schema) describe how users see the data
  - Conceptual schema defines logical structure
  - Physical schema describes files and indexes used
- Known as the **Three-Schema Architecture**

THREE-SCHEMA ARCHITECTURE

- Defines DBMS schemas at *three levels*:
  - **Internal schema** at the internal level to describe physical storage structures and access paths. Typically uses a *physical* data model.
  - **Conceptual schema** at the conceptual level to describe the structure and constraints for the *whole* database for a community of users. Uses a *conceptual* or an *implementation* data model.
  - **External schemas** at the external level to describe the various user views. Usually uses the same data model as the conceptual level.

THREE-SCHEMA ARCHITECTURE



CONCEPTUAL SCHEMA

- The conceptual schema describes all the relations stored in the database
- Creating a good conceptual schema is not a simple task. It is called conceptual schema design. It involves:
  - Determining the different relations(entities) needed
  - The number of fields for each relation
  - The type of each field
  - The relationship between relations
  - Constraints
  - ...

INTERNAL SCHEMA

- The internal schema specifies how the relations are actually stored in secondary storage devices
- It also specifies *indexes* used to speed up access to the relations
- Decisions about the internal schema depend on:
  - Understanding how the data is going to be accessed
  - The facilities provided by the DBMS

EXTERNAL SCHEMA

- The external schema is a refinement of the conceptual schema
- It allows customized and authorized access to individual users or groups of users
- Every database has one physical and one conceptual schema, but *many* external schemas
- Each view is tailored to a particular group of users

DATA INDEPENDANCE

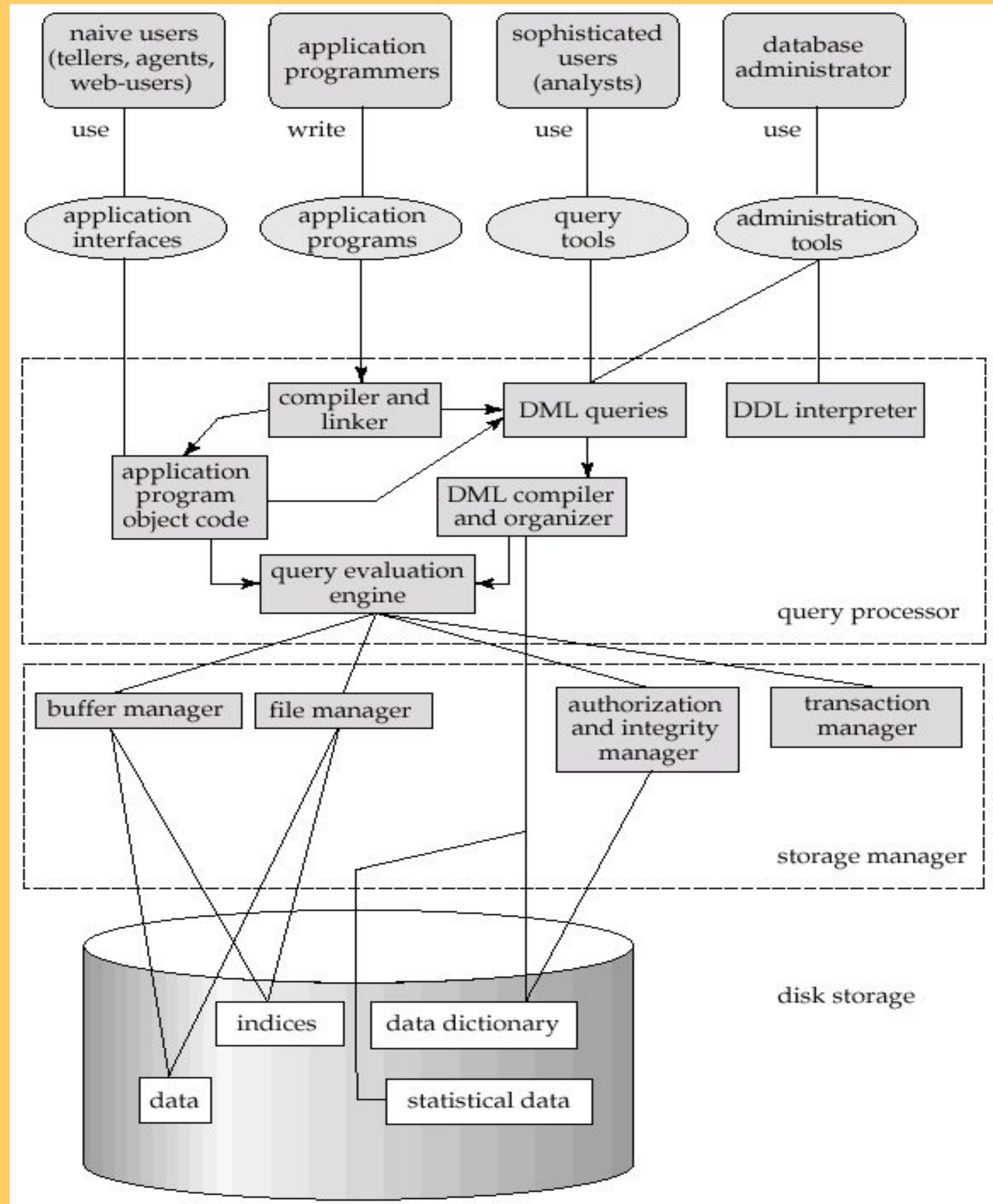
- Applications insulated from how data is structured and stored
- *Logical data independence*: protection from changes in the *logical* structure of data
  - The capacity to change the conceptual schema without having to change the external schemas and their application programs.
- *Physical data independence*: protection from changes in the *physical* structure of data
  - The capacity to change the internal schema without having to change the conceptual schema.
- When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed. The higher-level schemas themselves are *unchanged*.

DBMS LANGUAGES

- **High Level** or **Non-procedural Languages:** e.g., SQL, are *set-oriented* and specify what data to retrieve than how to retrieve. Also called *declarative* languages.
- **Low Level** or **Procedural Languages:** record-at-a-time; they specify *how* to retrieve data and include constructs such as looping.

OVERALL SYSTEM ARCHITECTURE

- A database system is partitioned into modules that deal with each of the responsibilities of the system and is broadly divided into **storage manager** and **query processor** components



STORAGE MANAGER

- A program module that provides the interface between the low-level data stored in the db and the app. Programs and queries submitted to the system.
- It is responsible for the interaction with the file manager.
- It translates the various DML statements into low-level file commands.
- It is responsible for the storing, retrieving, and updating data in the db.

STORAGE MANAGER

- It implements several data structures as part of the physical system implementation.
  - **Data files** which store the data itself.
  - **Data dictionary** which stores the metadata about the structure of the db.
  - **Indices** which provide fast access to data items that hold particular values.

QUERY PROCESSOR

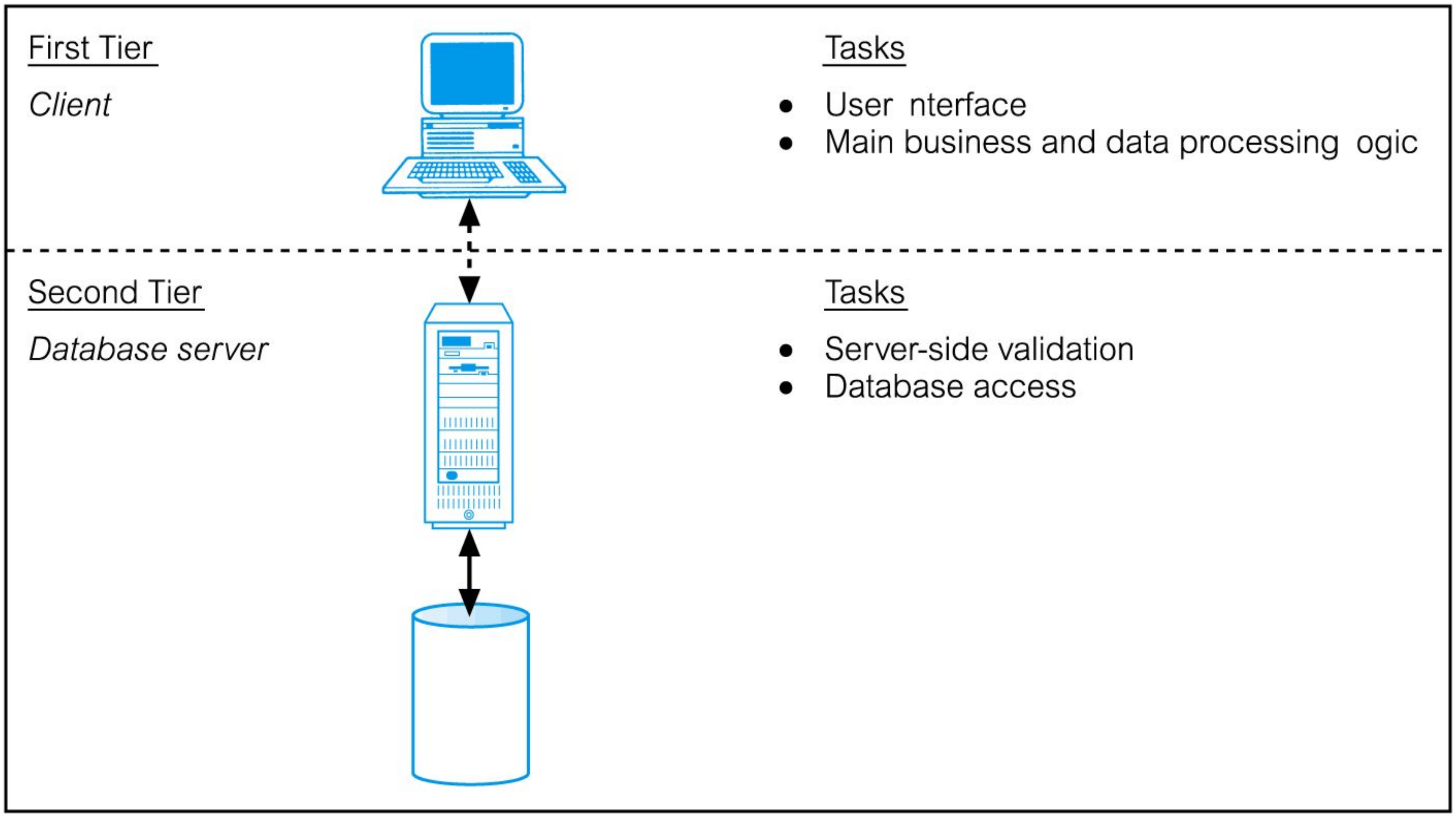
- **DDL interpreter** that interprets DDL statements and records into the definitions in the data dictionary.
- **DML compiler** that translates DML statements in a query language into an evaluation plan consisting of low-level instructions that query evaluation engine understands.
- **Query evaluation engine** which executes low-level instructions generated by the DML compiler.

TWO TIER CLIENT ARCHITECTURE

- Client (tier 1) manages user interface and runs applications.
- Server (tier 2) holds database and DBMS.
- Advantages include:

- wider access to existing databases;
- increased performance;
- possible reduction in hardware costs;
- reduction in communication costs;
- increased consistency.

TWO TIER CLIENT ARCHITECTURE



THREE TIER CLIENT ARCHITECTURE

- Client side presented two problems preventing true scalability:
  - Considerable resources required on client's computer to run effectively.
  - Significant client side administration overhead.

- By 1995, three layers proposed, each potentially running on a different platform.
  - Advantages:
    - Client requiring less expensive hardware.
    - Application maintenance centralized.
    - Easier to modify or replace one tier without affecting others.
    - Separating business logic from database functions makes it easier to implement load balancing.
    - Maps quite naturally to Web environment.

THREE TIER CLIENT ARCHITECTURE