Database Management Systems(CS19443)

II YEAR- IV SEMESTER

SYLLABUS

UNIT-I	INTRODUCTION TO DATABASE SYSTEMS	10
Introduction – Purpose of Database Systems - View of Data –Database Architecture - Relational Databases – Database Schema – Keys – Codd's Rule – Relational Algebra – Data Models – Entity Relationship Model – Constraints – Entity Relationship Diagram - Design Issues of ER Model – Extended ER Features – Mapping ER Model to Relational Model.		
UNIT-II	SQL AND QUERY PROCESSING	10
SQL: Data Definition – Domain types – Structure of SQL Queries - Modifications of the database – Set Operations – Aggregate Functions – Null Values – Nested Sub queries – Complex Queries – Views – Joined relations – Complex Queries – PL/SQL: Functions, Procedures, Triggers, Cursors -Embedded SQL – Query Processing – Heuristics for Query Optimization .		
UNIT-III	DEPENDENCIES AND NORMAL FORMS	8
Motivation for Normal Forms – Functional dependencies – Armstrong's Axioms for Functional Dependencies – Closure for a set of Functional Dependencies – Definitions of 1NF-2NF-3NF and BCNF – Multivalued Dependency 4NF - Joint Dependency- 5NF.		
UNIT-IV	TRANSACTIONS	7
Transaction Concept – State – ACID Properties – Concurrency control - Serializability – Recoverability – Locking based protocols –Timestamp Based Protocol - Deadlock handling.		
UNIT-V	NOSQL DATABASE	10
Introduction to NoSQL - CAP Theorem – Data Models - Key-Value Databases - Document Databases- Column Family Stores – Graph Databases –Working of NOSQL Using MONGODB/CASSANDRA.		

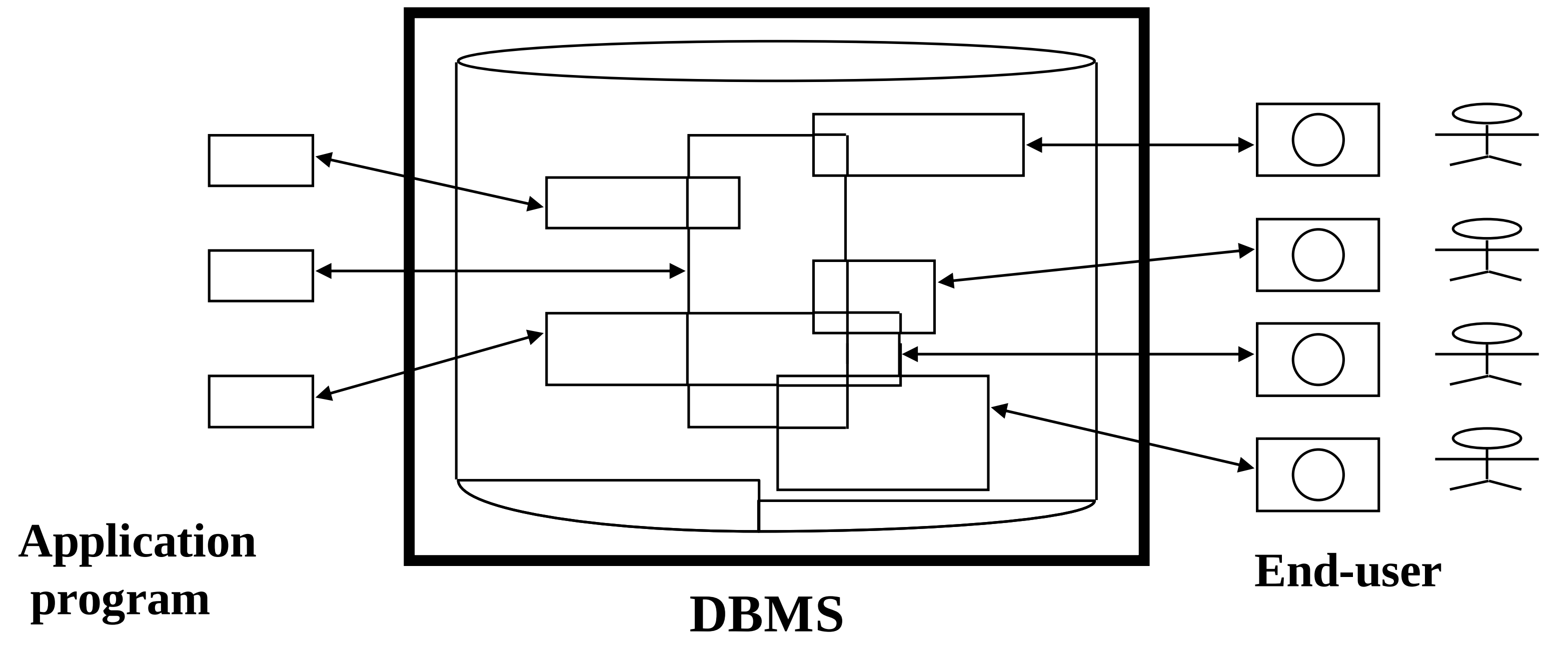
Text Book(s):

1. Abraham Silberschatz, Henry F. Korth and S. Sudharshan, “Database System Concepts”, Seventh Edition, Mc Graw Hill, March 2019.
2.P. J. Sadalage and M. Fowler, "NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence", Addison-Wesley Professional, 2013.

Reference Books(s):

1.“Ramez Elmasri and Shamkant B. Navathe, “Fundamentals of Database Systems”, Seventh Edition, Pearson Education, 2016.
2.C.J.Date, A.Kannan and S.Swamynathan, “An Introduction to Database Systems”, Eighth Edition, Pearson Education, 2006.
3.Atul Kahate, “Introduction to Database Management Systems”, Pearson Education, New Delhi, 2006.
4.Steven Feuerstein with Bill Pribyl,”Oracle PL/SQL Programming”,sixth edition, Publisher: O'Reill 2014.
5. MongoDB: The Definitive Guide, 3rd Edition,by Kristina Chodorow, Shannon Bradshaw,Publisher: O'Reilly Media,2019
6.Cassandra: The Definitive Guide, 2nd Edition,by Eben Hewitt, Jeff Carpenter.Publisher: O'Reilly Media, 2016.

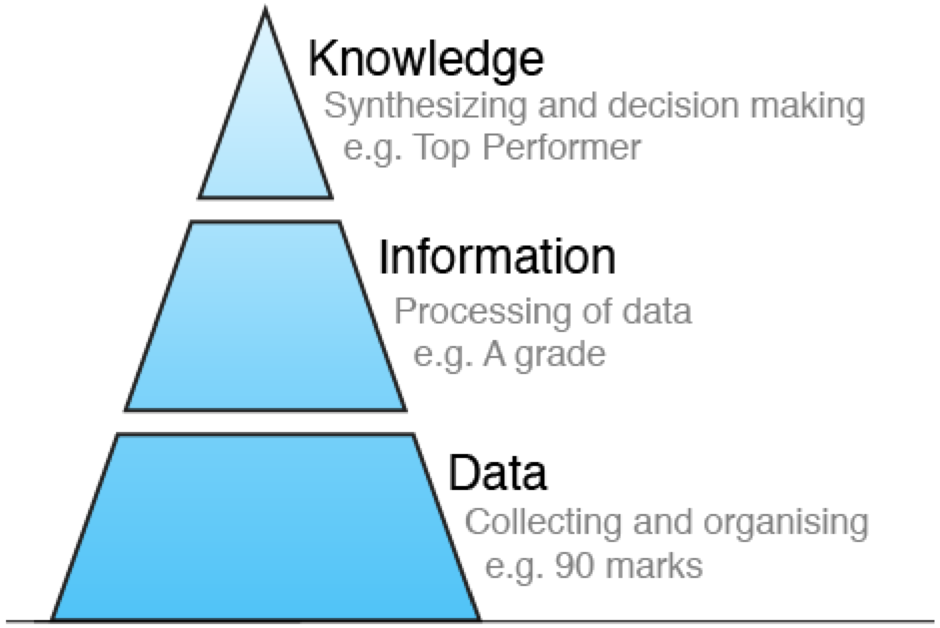
Unit 1
Introduction to Database Management Systems



Basic Definitions

Data:-Raw facts and figures that can be recorded.

Information: Meaningful (processed) data is known as information



Example-

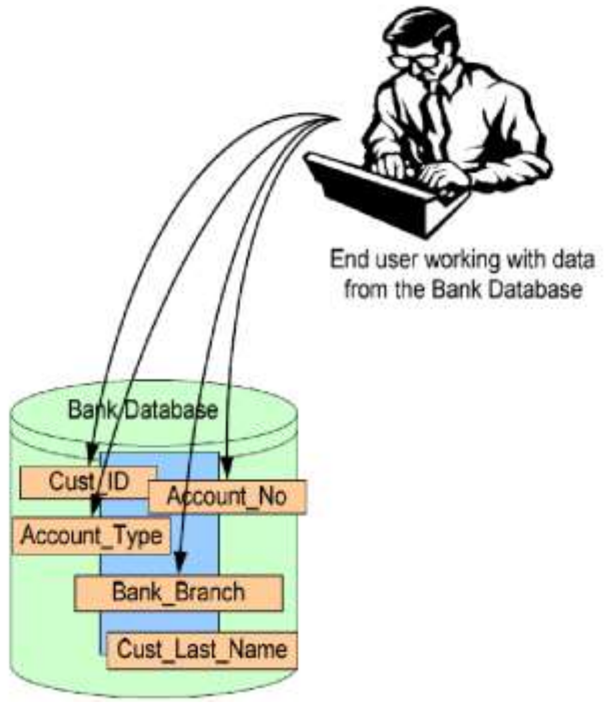
10,971,108

- Data
- Chennai Population in 2020
- Information

Basic Definitions(Cont.)

Database: Collection of inter-related data stored in a secondary storage device, organized meaningfully for a specific purpose.

- Database is a collection of data that contains information relevant to an enterprise.
- Databases are designed to manage large bodies of information.



A simplified Bank Database

Basic Definitions(Cont.)

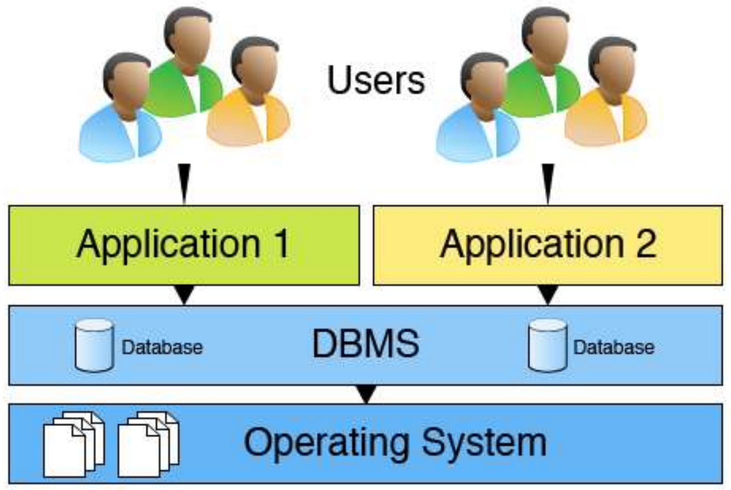
DBMS: Database Management System is a collection of interrelated persistent data and a set of program to access and manage those data.

- Management of data involves both defining structures for storage of information and providing mechanisms for manipulation of information.
- The primary goal of a DBMS is to provide an environment to store, retrieve and modify database information that is both convenient and efficient.

Database System: Database and DBMS collectively known as database system.

Database systems are used to manage collections of data that are:

- Highly valuable
- Relatively large
- Accessed by multiple users and applications, often at the same time.



Database Applications Examples

- **Enterprise Information**
  - Sales: customers, products, purchases
  - Accounting: payments, receipts, assets
  - Human Resources: Information about employees, salaries, payroll taxes.
- **Manufacturing:** management of production, inventory, orders, supply chain.
- **Banking and finance**
  - customer information, accounts, loans, and banking transactions.
  - Credit card transactions
  - Finance: sales and purchases of financial instruments (e.g., stocks and bonds; storing real-time market data)
- **Universities:** registration, grades
- **Airlines:** reservations, schedules

Database Applications Examples (Cont.)

- **Telecommunication:** records of calls, texts, and data usage, generating monthly bills, maintaining balances on prepaid calling cards
- **Web-based services**
  - Online retailers: order tracking, customized recommendations
  - Online advertisements
- **Document databases**
  - A document database is a type of nonrelational database that is designed to store and query data as JSON-like documents.
  - Document databases make it easier for developers to store and query data in a database by using the same document-model format they use in their application code.
- **Navigation systems**: For maintaining the locations of varies places of interest along with the exact routes of roads, train systems, buses, etc.
- **Scientific:** digital libraries, genomics, satellite imagery, physical sensors, simulation data
- **Personal:** Music, photo, & video libraries ,Email archives , File contents (“desktop search”).

Disadvantages of Conventional File-Processing System

- File system is a collection of data .
- In the traditional file approach, each application maintains its own master file and generally, has its own set of transaction files.
- Files are custom-designed for each application and there is little sharing of data among the various applications .
- Application programs are data-dependent. It is impossible to change the physical representation or access techniques without affecting the application.
- This typical file processing system is supported by a conventional operating system. The system stores permanent records in various files ,and it needs different application programs to extract records/ information from the file system.

Purpose of DBMS / Disadvantages of Conventional File-Processing System

Data redundancy and inconsistency

- In file processing, every user group maintains its own files for handling its data-processing applications. Storing the same data multiple times is called data redundancy.
- This redundancy leads to several problems such as storage space is wasted; need to perform a single logical operation like insertion, update and deletion more than one times which may lead to data inconsistency problem.
- *For Example*: The address and phone number of a particular customer may appear in a file that consists of personal information and in saving account records file also. This redundancy leads to data consistency that is, the various copies of the same data may no longer agree.

Purpose of DBMS / Disadvantages of Conventional File-Processing System (Cont.)

Difficulty in accessing data

- File processing environments do not allow needed data to be retrieved in a convenient and efficient manner.
- For Example: Suppose that bank officer needs to find out the names of all customers who live within the city‘s 411027 zip code. The bank officer has now two choices: Either get the list of customers and extract the needed information manually or ask the data processing department to have a system programmer write the necessary application program. Both alternatives are unsatisfactory.

Data isolation

- Because data are scattered in various files and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

Purpose of DBMS / Disadvantages of Conventional File-Processing System (Cont.)

Integrity problems

- The data values stored in database must satisfy certain types of consistency constraints. Developers enforce those constraints in the system by adding appropriate code in the various application programs. When new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.
- *For Example:* The balance of a bank account may never fall below a prescribed amount (say ₹5000).These constraints are enforced in the system by adding appropriate code in the various application programs.

Atomicity problems

- Atomic means the transaction must happen in it’s entirely or not at all. It is difficult to ensure atomicity in a conventional file-processing system.
- Failures may leave database in an inconsistent state with partial updates carried out
- Example: Transfer of funds from one account to another should either complete or not happen at all

Purpose of DBMS / Disadvantages of Conventional File-Processing System (Cont.)

Concurrent-access anomalies

- Many systems allow multiple users to update the data simultaneously to provide overall performance of the system and faster response.
- The result of the concurrent access may leave the account in an incorrect state.
- For Example: Consider bank account A, containing ₹5000. If two customers withdraw funds say ₹500 and ₹1000 respectively from account A at about the same time, the result of the concurrent executions may leave the account in an incorrect (or inconsistent) state. Balance will be ₹4000 instead of ₹3500. To protect against this possibility, the system must maintain some form of supervision.

Security problems

- Enforcing security constraints in an ad hoc manner to the file processing system is difficult. Not every user of the database system should be able to access all the data.
- *For Example*: In a banking system, payroll personnel should be only given authority to see the part of the database that has information about the various bank employees. They do not need access to information about customer accounts

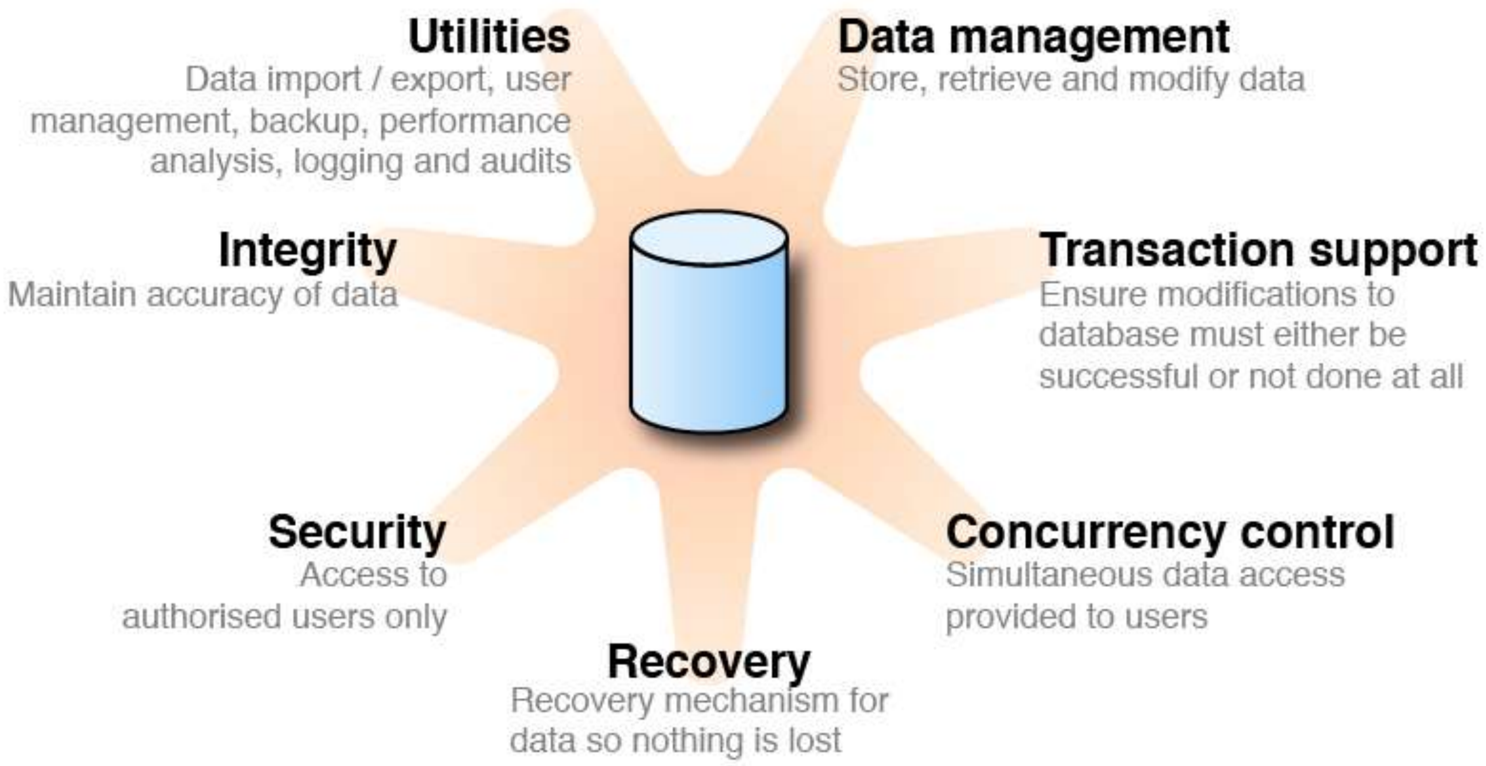
Advantages of a DBMS

- **Improved security -** Database security is the protection of the database from an unauthorized access.
- **Improved data integrity-** Database integrity refers to the validity and consistency of stored data. Integrity is expressed in terms of constraints, which are consistency rules that the database is not permitted to violate.
- **Data consistency -** If a data item is stored more than once and the system is aware of this, the system can ensure that all copies of the item are kept consistent.
- **Improved data accessibility and responsiveness -** Many DBMSs provide query language that allows users to enquire questions and to obtain the required information at their terminals, without any need of separate software.
- **Increased concurrency -**Many DBMSs manage concurrent database access and ensure the data in the database is consistent and valid.
- **Improved backup and recovery services -** Modern DBMSs provide facilities to minimize the amount of processing that is lost following a failure.

Disadvantages of a DBMS

- **Cost of DBMS** -The cost of DBMS varies significantly, depending on the environment and functionality provided
- **Complexity and Size** -The provision of the functionality makes DBMS an extremely complex piece of software. Failure to understand the system can lead to bad design decisions.
- **Higher impact of a failure** -The centralization of resources increases the vulnerability of the system. Since all users and applications rely on the availability of the DBMS, the failure of any component can bring operations to a halt.
- **Cost of conversion** -The cost of converting existing applications to run on the new DBMS and hardware includes the cost of training staff to use these new systems and running of the system.
- **Performance** -The DBMS is written to be more general in order to support applications in all domains. The effect is that some applications may not run as they used to.

Functions of DBMS


Utilities
Data import / export, user management, backup, performance analysis, logging and audits
Data management
Store, retrieve and modify data
Integrity
Maintain accuracy of data
Transaction support
Ensure modifications to database must either be successful or not done at all
Security
Access to authorised users only
Concurrency control
Simultaneous data access provided to users
Recovery
Recovery mechanism for data so nothing is lost

Levels of Abstraction

A major purpose of a database system is to provide users with an abstract view of the data.

A database system must retrieve data efficiently.

The need for efficiency has led designers to use complex data structures to represent data in the database.

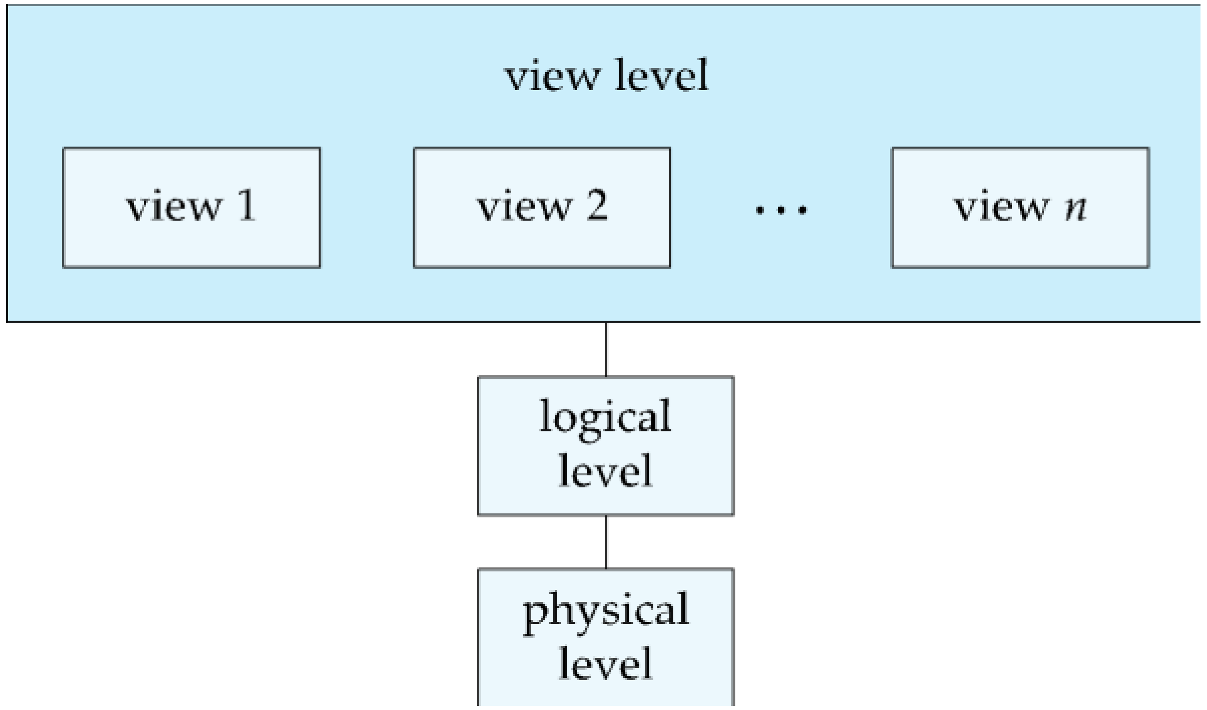
Since many users are not computer trained, developers hide the complexity from users through several levels of abstraction to simplify users’ interactions with the system.

Three levels abstraction of a database system

- Physical / Internal Level: The lowest level of abstraction
- Logical Level: The next-higher level of abstraction
- View Level: The highest level of abstraction

View of Data

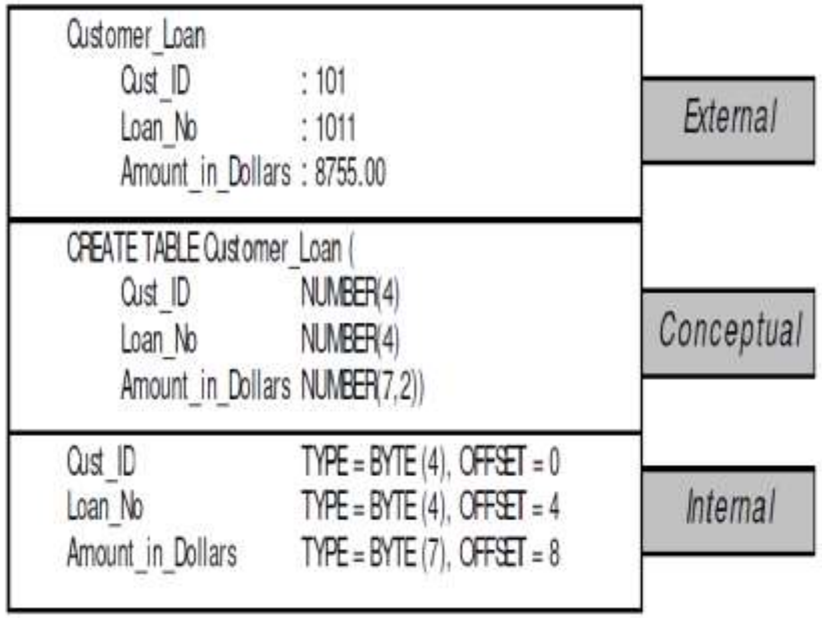
A three-level architecture for a database system



Three-levels architecture

- Physical / Internal Level: The lowest level of abstraction describes *how* the data are stored. This level describes complex low-level data structures in details.
- Logical Level: The next-higher level of abstraction describes *what* data are stored in the database and what relationships exist among those data, although implementation of the simple structures at the logical level does not need to be aware of this complexity.

  Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.
- View Level: The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database.


Customer_Loan
Cust_ID : 101
Loan_No : 1011
Amount_in_Dollars : 8755.00
External
CREATE TABLE Customer_Loan (
Cust_ID NUMBER(4)
Loan_No NUMBER(4)
Amount_in_Dollars NUMBER(7,2))
Conceptual
Cust_ID TYPE = BYTE (4), OFFSET = 0
Loan_No TYPE = BYTE (4), OFFSET = 4
Amount_in_Dollars TYPE = BYTE (7), OFFSET = 8
Internal

Instances and Schemas

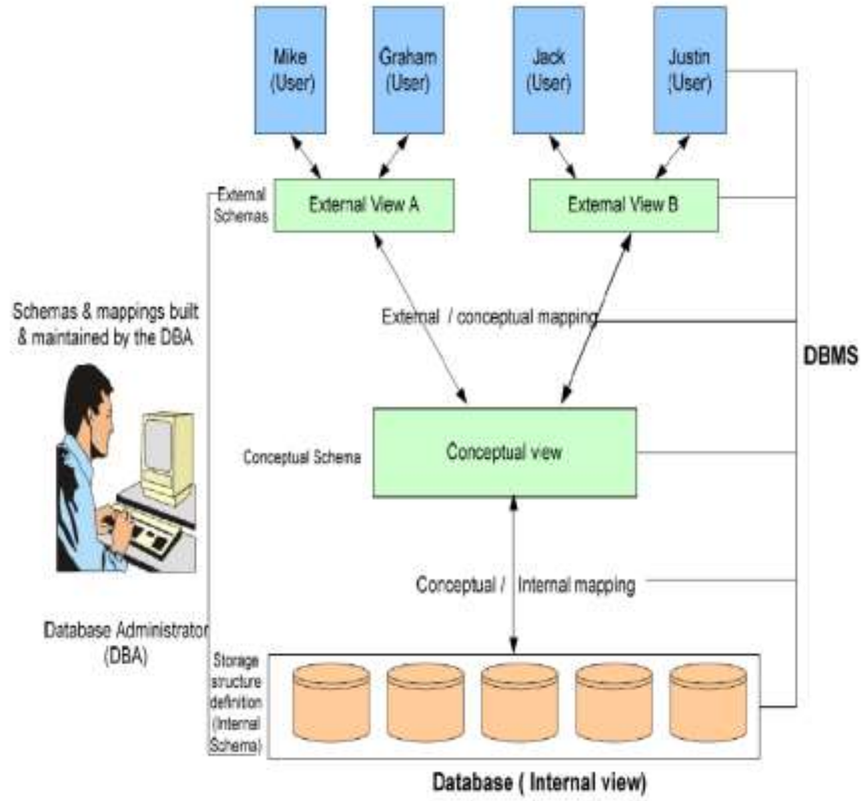
Like types and variables in programming languages

- **Schema-** Overall structure of a database system

  Blueprint of how a database is constructed

  - **Logical Schema** – the overall logical structure of the database
    - It describes the database design at the logical level. The physical schema is hidden beneath the logical schema.
    - Analogous to type information of a variable in a program
  - **Physical schema** – The overall physical structure of the database
  - It describes the database design at the physical level.
  - View schema / Subschema: A database may also have several schemas at the view level called as sub-schemas that describe different views of the database

- **Instance** – The collection of information stored in the database at a particular moment is called an instance of the database.
  - Analogous to the value of a variable



Mike (User)
Graham (User)
Jack (User)
Justin (User)
External Schemas
External View A
External View B
Schemas & mappings built & maintained by the DBA
External / conceptual mapping
DBMS
Conceptual Schema
Conceptual view
Conceptual / Internal mapping
Database Administrator (DBA)
Storage structure definition (Internal Schema)
Database (Internal view)

Data Independence

- The ability to modify a schema definition in one level without affecting a schema definition in the next higher level is called data independence. There are two levels of data independence:

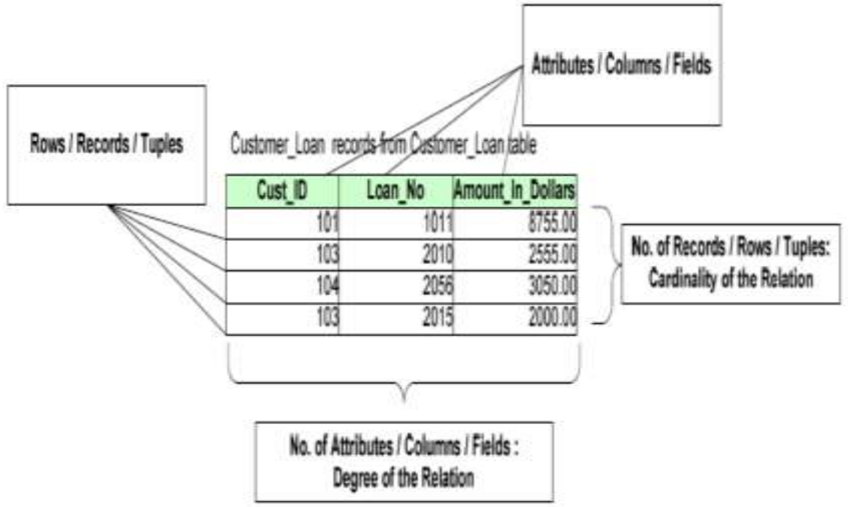
1. **Physical data independence** is the ability to modify the physical schema without causing application programs to be rewritten.
    - Modifications at the physical level are occasionally necessary in order to improve performance.
2. **Logical data independence** is the ability to modify the conceptual schema without causing application programs to be rewritten.
    - Modifications at the conceptual level are necessary whenever the logical structure of the database is altered.

Data Models

- Data Model is a collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- Types:
  1. Relational Model
  2. The Entity-Relationship Model
  3. Object-Based Data Model
  4. Semi-Structured Data Model (XML)
  5. Network Data Model
  6. Hierarchical Data Model

Relational Model

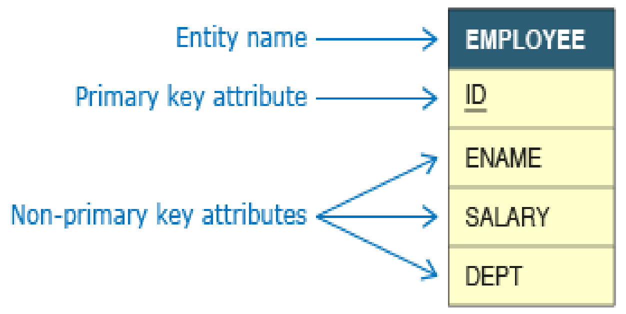
- The relational model uses a collection of tables to represent both data and the relationships among those data.
- It is based on the concept of mathematical relations.
- Each table corresponds to an entity and each row represents an instance of that entity.
- Table are also called relations, related to each other through the sharing of a common entity characteristic.
- The relational data model is widely used data model and a vast majority of current database systems are based on the relational model e.g., Relational DBMS - DB2, MS SQL Server.

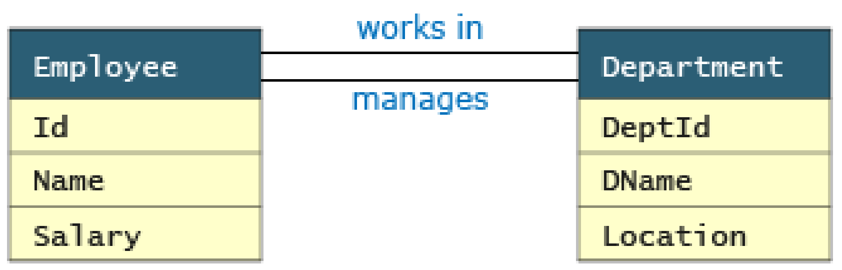

Attributes / Columns / Fields
Rows / Records / Tuples
Customer_Loan records from Customer_Loan table
No. of Records / Rows / Tuples: Cardinality of the Relation
No. of Attributes / Columns / Fields : Degree of the Relation

Cust_ID	Loan_No	Amount_In_Dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

Entity-Relational Model

- The ER Model is based on two components namely ***entity*** and ***relationships***.
- An **entity** is a collection of basic real time objects and an entity is described by a set of attributes that describes associations among data.
- A **relationship** describes associations among these objects/ data. There are three types of relationships exist such as One-to-One, One-to-Many and Many-to-Many.



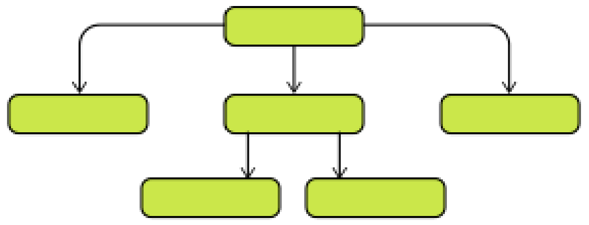


Object-Based Data Model

- In the object-oriented data model (OODM) both data and their relationships are contained in a single structure known as an **object**.
- The object-oriented data model can be seen extending the ER model with notions of encapsulation, methods and object identity. It combines features of the object-oriented data model and relational data model. The OODM is said to be a semantic data model because semantic indicates meaning.
- The OODM is based on the following components:
  - An object is an abstraction of a real-world entity.
  - Attributes describe the properties of an object.
  - A class is a collection of similar objects with shared structure and behaviour. Classes are organized in a class hierarchy.
  - Inheritance is the ability of an object within the class hierarchy.

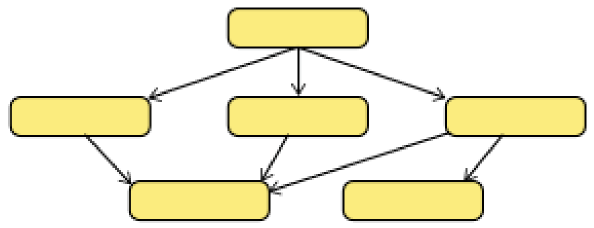
Hierarchical Model

- The hierarchical data model organizes data in a tree structure.
- There is a hierarchy of parent and child data segments.
- This structure implies that a record can have repeating information, generally in the child data segments.
- Data is represented by a collection of records (record types).
- A record type is the equivalent of a table in the relational model, and with the individual records being the equivalent of rows.
- To create links between these record types, the hierarchical model uses parent -child relationships.


Hierarchical
Tree structure

Network Model

- The network model permitted the modeling of many-to-many relationships in data.
- In 1971, the Conference on Data Systems Languages (CODASYL) formally defined the network model.
- Data in the network model is represented by a collection of records and the relationships among data are represented by links (pointers).
- The records in the database are organized as collections of graphs. Example: IDMS.
- IDMS, short for Integrated Database Management System, is primarily a network model database management system for mainframes.



Network
Graph structure

Database Design

- The process of designing the general structure of the database:
- Logical Design – Deciding on the database schema. Database design requires that we find a “good” collection of relation schemas.
  - Business decision – What attributes should we record in the database?
  - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- Physical Design – Deciding on the physical layout of the database

Database Engine

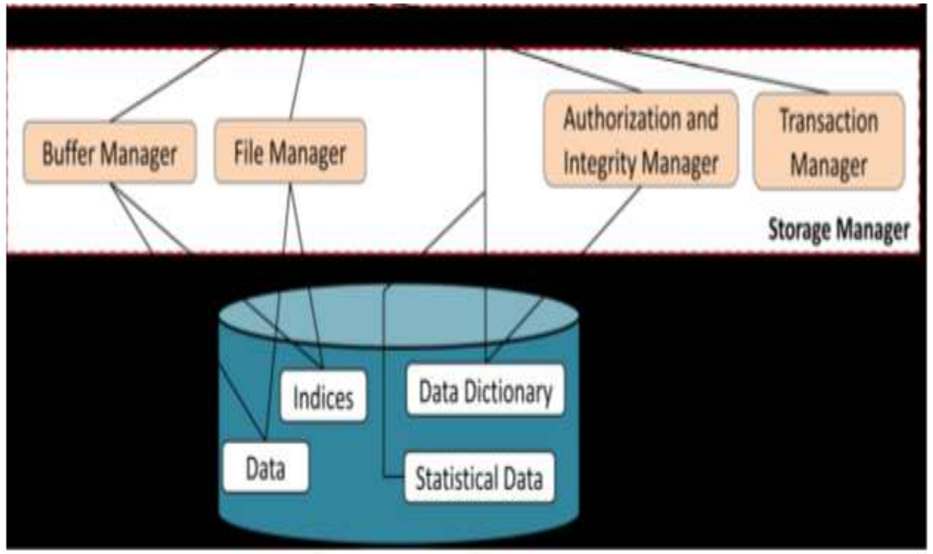
- A database system is partitioned into modules that deal with each of the responsibilities of the overall system.
- The functional components of a database system can be divided into
  - The storage manager,
  - The query processor component,
  - The transaction management component.

Storage Manager

- A program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible to the following tasks:
  - Interaction with the OS file manager
  - Efficient storing, retrieving and updating of data
- The storage manager components include:
  - Authorization and integrity manager- It tests for satisfaction of various integrity constraints and checks the authority of users accessing the data.
  - Transaction manager- It ensures that the database remains in a consistent state despite system failures, and concurrent executions proceed without conflicting.
  - File manager- It manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
  - Buffer manager- It is responsible for fetching data from disk storage into main memory and to decide what data to cache in main memory. It enables the database to handle data sizes that are much larger than the size of the main memory.

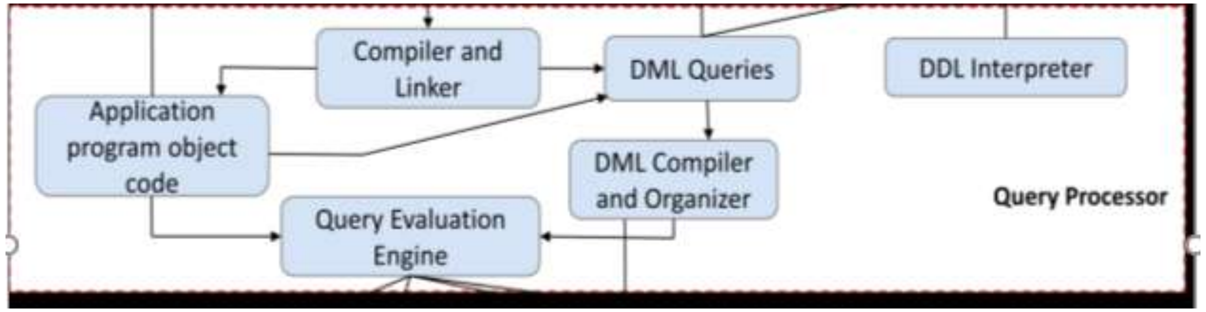
Storage Manager(Cont.)

- The storage manager implements several data structures as part of the physical system implementation:
  - **Data files** -- store the database itself
  - **Data dictionary** -- stores metadata about the structure of the database, in particular the schema of the database. A database system consults the data dictionary before reading and modifying actual data.
  - **Indices** -- can provide fast access to data items. A database index provides pointers to those data items that hold a particular value.



Query Processor

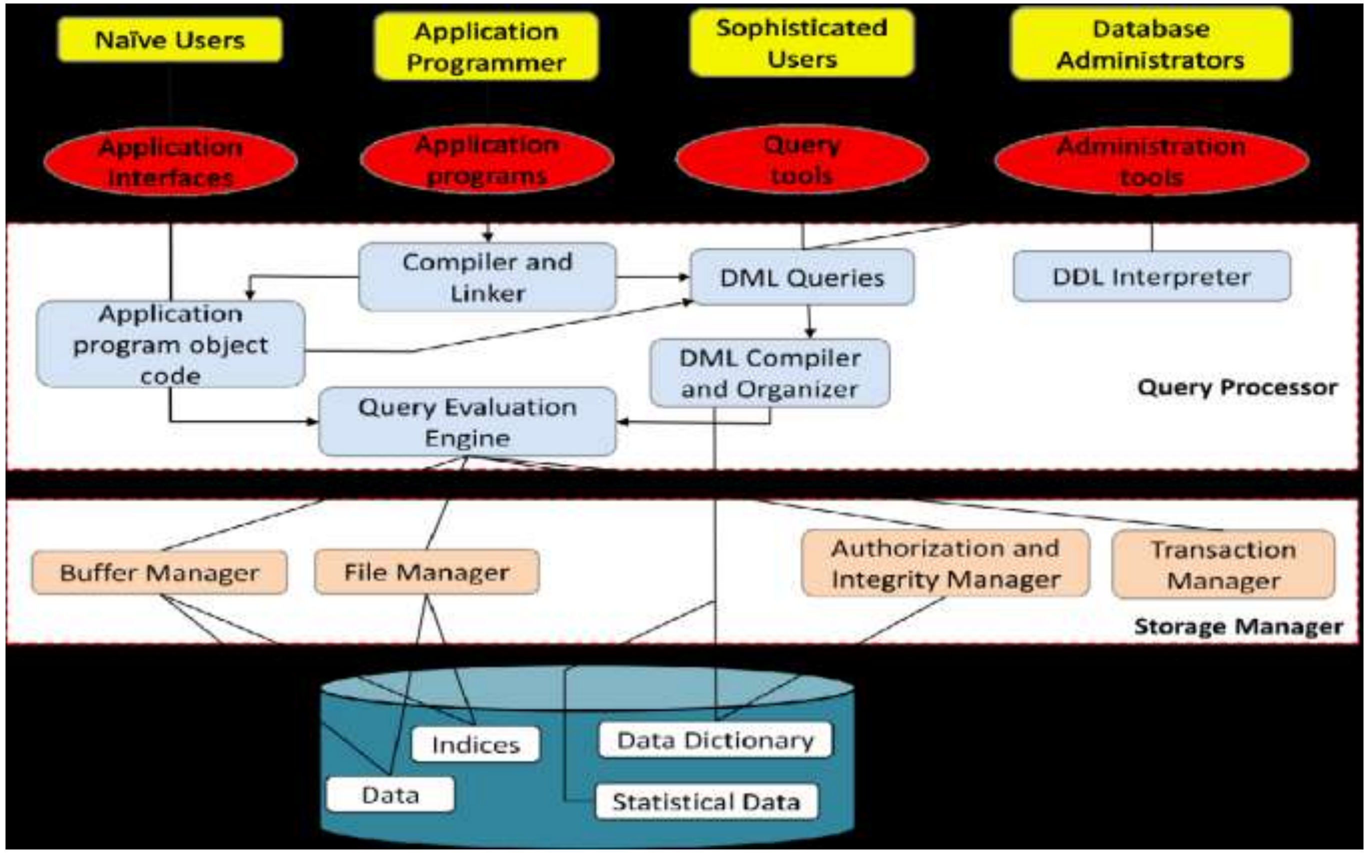
- The major component of a DBMS is Query Processor that transforms queries into a series of low-level instructions.
- It helps the database system to simplify and facilitate access to data.
- The query processor components include:
  - **DDL interpreter** -- interprets DDL statements and records the definitions in the data dictionary.
  - **DML compiler** -- translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
    - The DML compiler performs query optimization; that is, it picks the lowest cost evaluation plan from among the various alternatives.
  - **Query evaluation engine** -- executes low-level instructions generated by the DML compiler.



Transaction Management

- A transaction is a collection of operations that performs a single logical function in a database application
- Transaction-management component ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- Concurrency-control manager controls the interaction among the concurrent transactions, to ensure the consistency of the database.

Database Architecture (Centralized/Shared-Memory)



Database Users

- There are four different types of database-system users, differentiated by the way they expect to interact with the system.
  - Different types of user interfaces have been designed for the different types of users.
- **Naive users:** Naive users interact with the system by invoking one of the application programs that have been written previously.
  - Naive users are typical users of form interface, where the user can fill in appropriate fields of the form.
  - Naive users may also simply read reports generated from the database
- **Application programmers** Application programmers are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces.

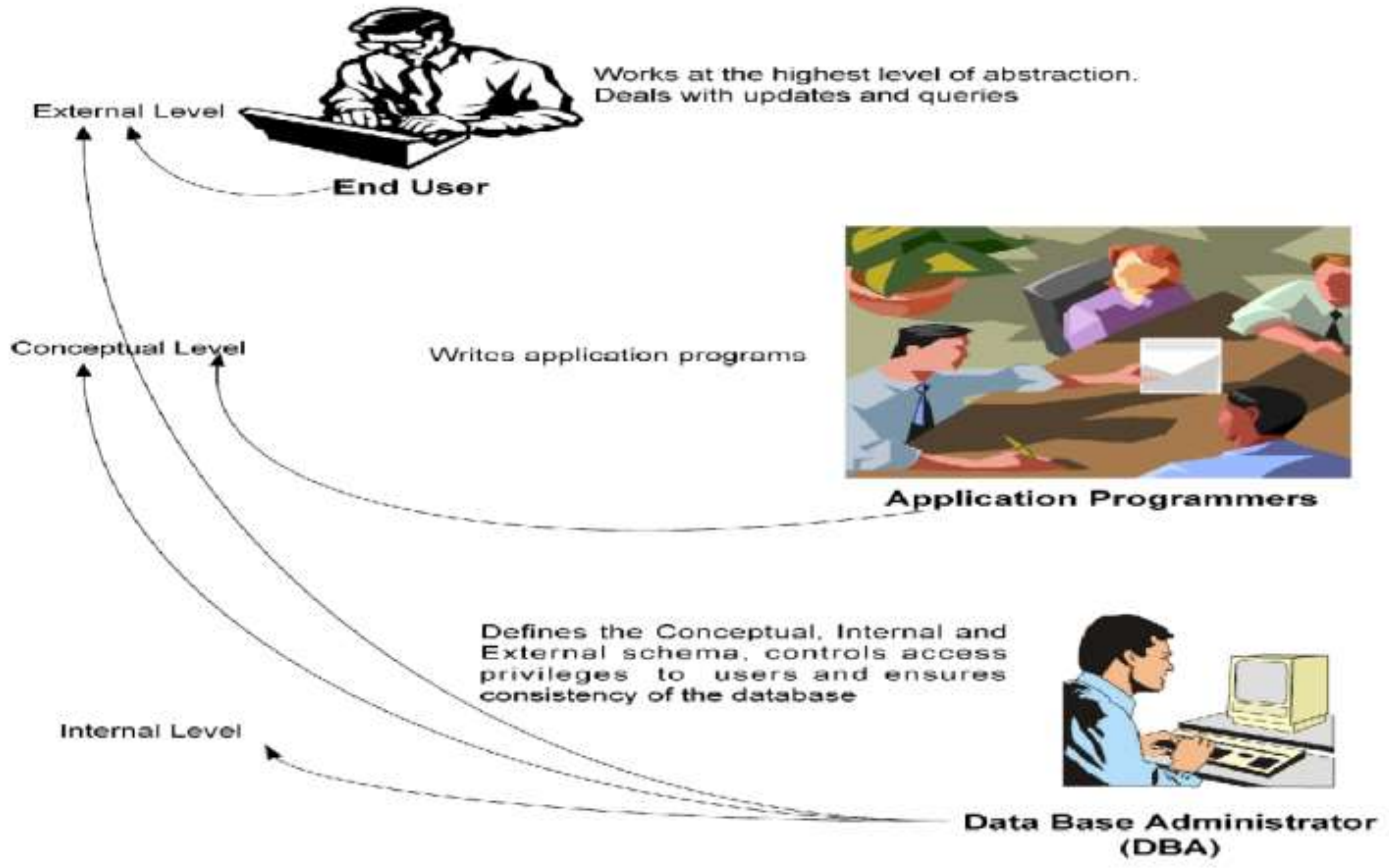
Database Users(Cont.)

- **Sophisticated users:** Sophisticated users interact with the system without writing programs. Instead, they form their requests in a database query language.
  - They submit each such query to a query processor that the storage manager understands.
  - Online analytical processing (OLAP) tools simplify analysis and data mining tools specify certain kinds of patterns in data
- **Specialized users:** Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.
  - The applications are computer-aided design systems, knowledge base and expert systems, systems that store data with complex data types .

Database Administrator

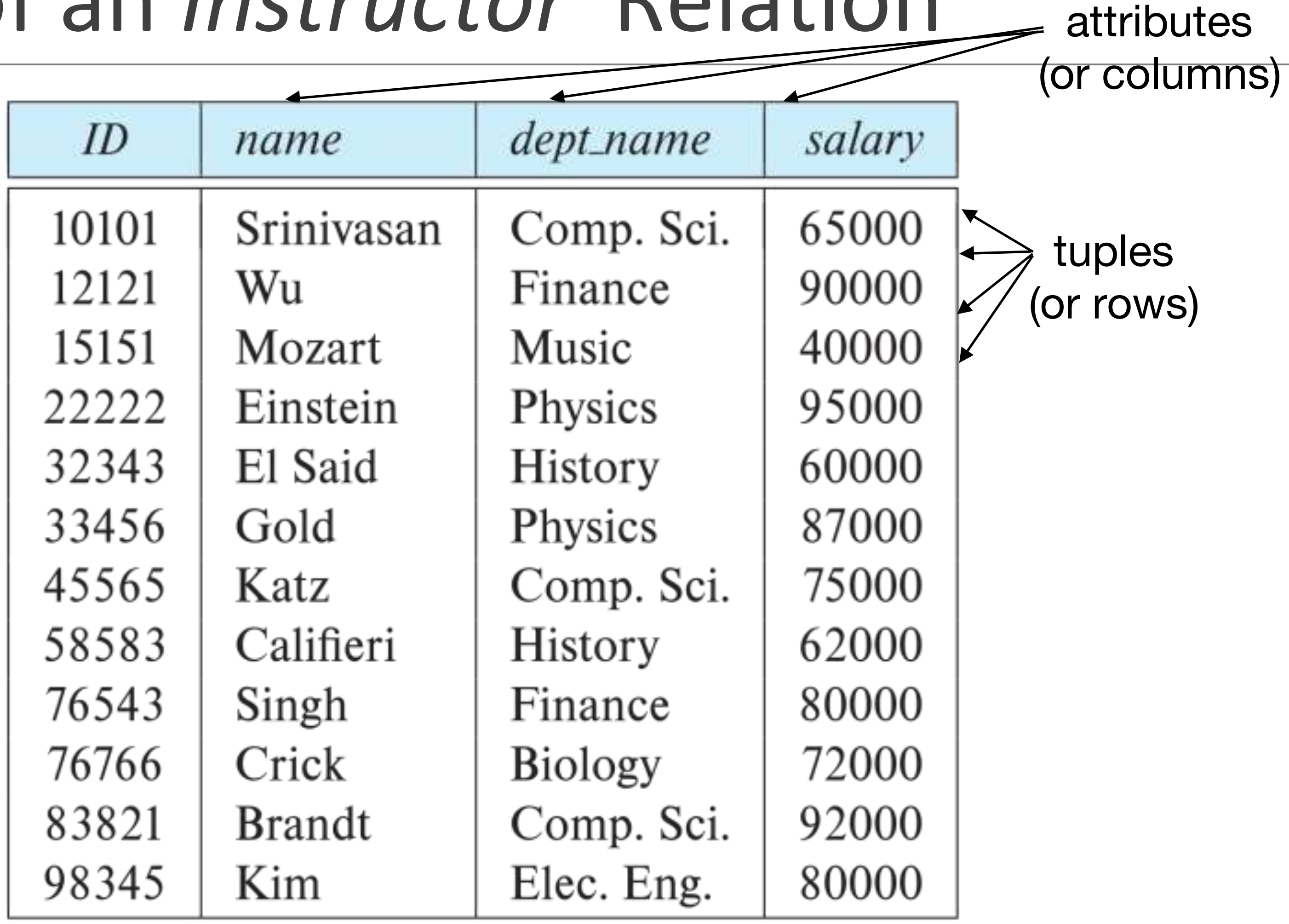
- A person who has central control over the system is called a **database administrator (DBA).** Functions of a DBA include:
  - Schema definition
  - Storage structure and access-method definition
  - Schema and physical-organization modification
  - Granting of authorization for data access
  - Routine maintenance
  - Periodically backing up the database
  - Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required
  - Monitoring jobs running on the database

Users at different level of abstractions


Works at the highest level of abstraction.
Deals with updates and queries
External Level
End User
Conceptual Level
Writes application programs
Application Programmers
Defines the Conceptual, Internal and External schema, controls access privileges to users and ensures consistency of the database
Internal Level
Data Base Administrator (DBA)

Introduction to Relational Model

Example of an *Instructor* Relation


attributes
(or columns)
tuples
(or rows)

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Relation Schema and Instance

$A_1, A_2, \ldots, A_n$ are *attributes*

$R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

Example:

Instructor = (*ID*, *name*, *dept_name*, *salary*)

A relation instance *r* defined over schema *R* is denoted by *r* (*R*).

The current values a relation are specified by a table

An element ***t*** of relation ***r*** is called a *tuple* and is represented by a *row* in a table

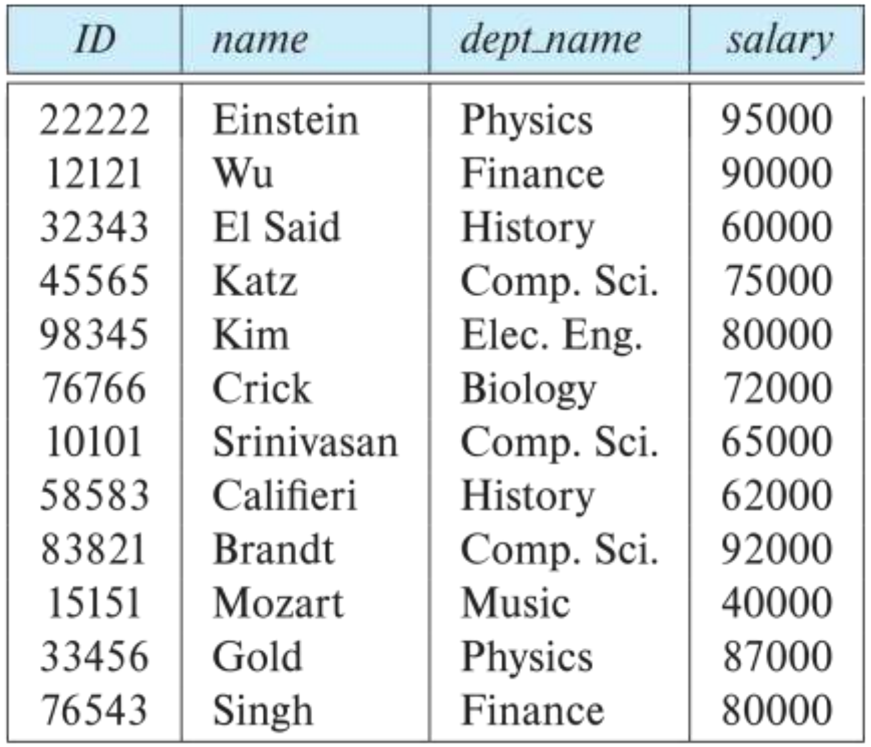
Attributes

- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are required to be **atomic**; that is, indivisible
- The special value ***null*** is a member of every domain. Indicated that the value is "unknown"

Relations are Unordered

Order of tuples is irrelevant (tuples may be stored in an arbitrary order)

Example: *instructor* relation with unordered tuples



ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

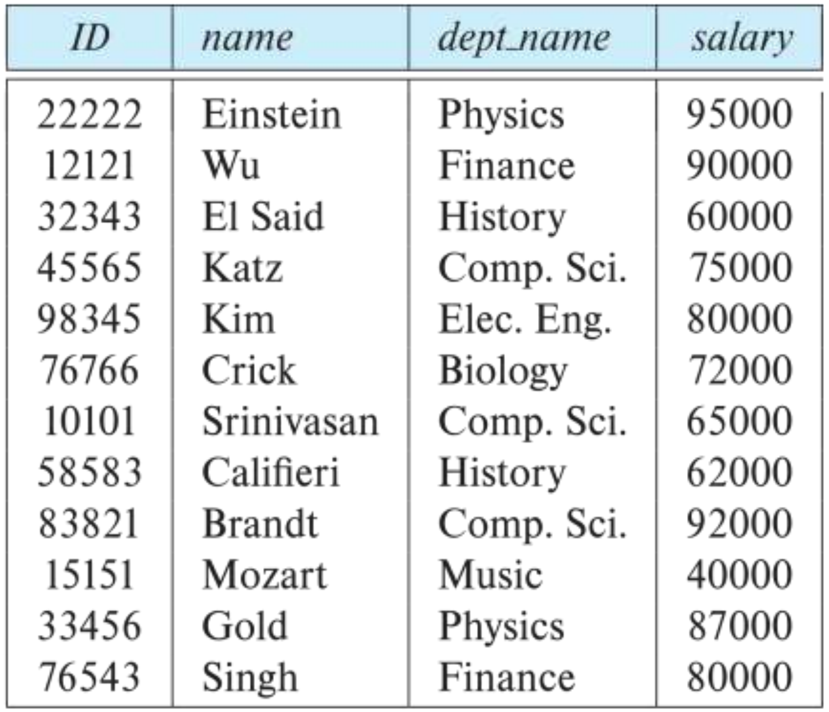
Database Schema

Database schema -- is the logical structure of the database.

Database instance -- is a snapshot of the data in the database at a given instant in time.

Example:

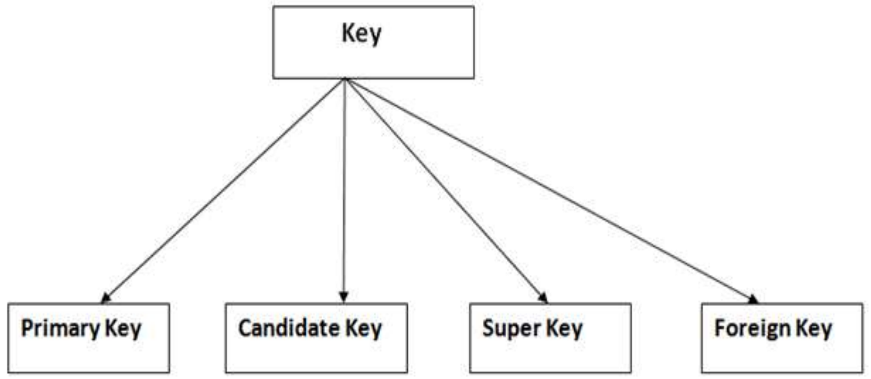
- schema:
  - *instructor* (*ID, name, dept_name, salary*)
- Instance:



ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Keys

- Let K ⊆ R , *K* is a **super key** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - Example: {*ID*} and {ID , name} are both super keys of *instructor.*
- Super key *K* is a **candidate key** if *K* is minimal Example: {*ID*} is a candidate key for *Instructor*
- A candidate key is a set of one or more attributes that can uniquely identify a row in a given table.
- There can be more than one candidate keys in a table.
- Candidate keys are identified during the design phase
- While creating the table, the database designer chooses one candidate key from amongst the several available, to serve as a primary key
- It is preferred to select a candidate key with a minimal number of attributes to function as a primary key



Keys(Cont.)

One of the candidate keys is selected to be the **Primary Key**.

- Which one?

Guidelines to select a primary key:

- Give preference to numeric column(s). The search algorithm performs better when the primary key is numeric
- Give preference to a single attribute.
- The search algorithm gives better output with a single attribute primary key than with a composite attribute primary key
- Give preference to the minimal composite key.
- Example: if the candidate keys are {x1,x2,x3} and {y1,y2}, the composite key{y1,y2} is the minimal composite key and will therefore be chosen as the primary key

Foreign key

Foreign key constraint: If an attribute can only take the values which are present as values of some other attribute, it will be foreign key to the attribute to which it refers.

The relation which is being referenced is called referenced relation and corresponding attribute is called referenced attribute and the relation which refers to referenced relation is called referencing relation and corresponding attribute is called referencing attribute.

Referenced attribute of referencing attribute should be primary key.

Foreign key(Cont.)

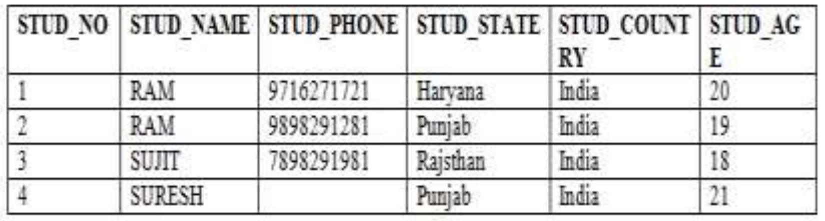
STUD_NO in STUDENT_COURSE is a foreign key to STUD_NO in STUDENT relation.

A foreign key is a set of attributes of a table, the values of which are required to match values of some candidate key in the same or another table.

The constraint that values of a given foreign key must match the values of the corresponding candidate key is known as referential constraint.

A table which has a foreign key referring to its own candidate key is known as self-referencing table

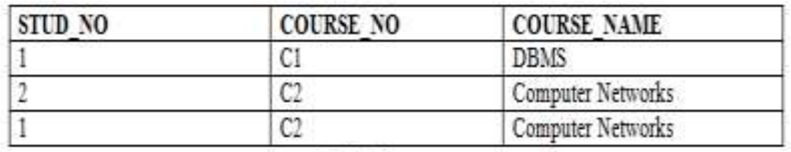
STUDENT



STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE



STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

OVERVIEW OF CODD’s RULE

- Any database which simply has relational data model is not a relational database system (RDBMS).
- There are certain rules for a database to be perfect RDBMS. These rules are developed by Dr. Edgar F Codd in 1985 to define a perfect RDBMS. For a RDBMS to be a perfect RDBMS, it must follow his rules.
- EF Codd has developed 13 rules for a database to be a RDBMS.
- According to him, all these rule help to have perfect RDBMS and hence correct data and relation among the objects in database.
- But none of the database follows all these rules; but obeys to some extent.
- For example, Oracle supports 11.5 of E.F. CODD Rule. The .5 missing was according to E.F. CODD Rule DML operation through a complex view is possible. But in oracle DML operation through a complex view is not always.

Codd’s Rule 0

- This is the foundational Rule.
- This rule states that any database system should be relational in nature and must have a management system to be RDBMS.
- That means a database should be a relational by having the relation / mapping among the tables in the database. They have to be related to one another by means of constraints/ relation. There should not be any independent tables hanging in the database.
- RDBMS is management system – that means it should be able to manage the data, relation, retrieval, update, delete, permission on the objects. It should be able handle all these administrative tasks without affecting the objectives of database. It should be performing all these tasks by using query languages.

Codd’s Rule 1

- Information Rule

  All information in the database is to be represented in one and only one way.

  Data must be stored in a table in the form of rows and columns.

  The order of rows and columns in the table should not affect the meaning of the table.

  Each cell should have single data.

  There should not be any group/range of values separated by comma, space or hyphen (Normalized data).

- This rule is satisfied by all the databases.

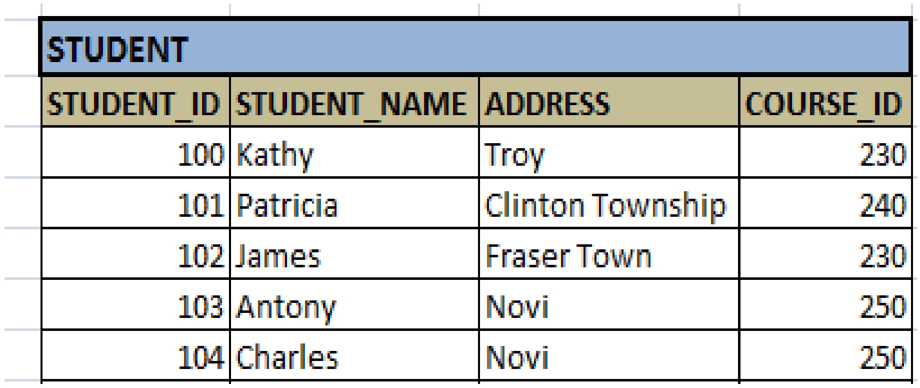
  For example:

  Order of storing personal details about ‘Ram’ and ‘Hari’ in PERSON table should not have any difference. There should be flexibility of storing them in any order in a row. Similarly, storing Person name first and then his address should be same as storing address and then his name. It does not make any difference on the meaning of table.

Codd's Rule 2

Guaranteed access Rule.

- Each unique piece of data should be accessible by, table name + primary key(row) + attribute(column).
- All data are uniquely identified and accessible via this identity.
- When combination of these 3 is used, it should give the correct result.
- Any column/ cell value should not be directly accessed without specifying the table and primary key.
- Most RDBMS do not make the definition of the primary key mandatory and are deficient to that extent .



STUDENT			
STUDENT_ID	STUDENT_NAME	ADDRESS	COURSE_ID
100	Kathy	Troy	230
101	Patricia	Clinton Township	240
102	James	Fraser Town	230
103	Antony	Novi	250
104	Charles	Novi	250

Address of Kathy

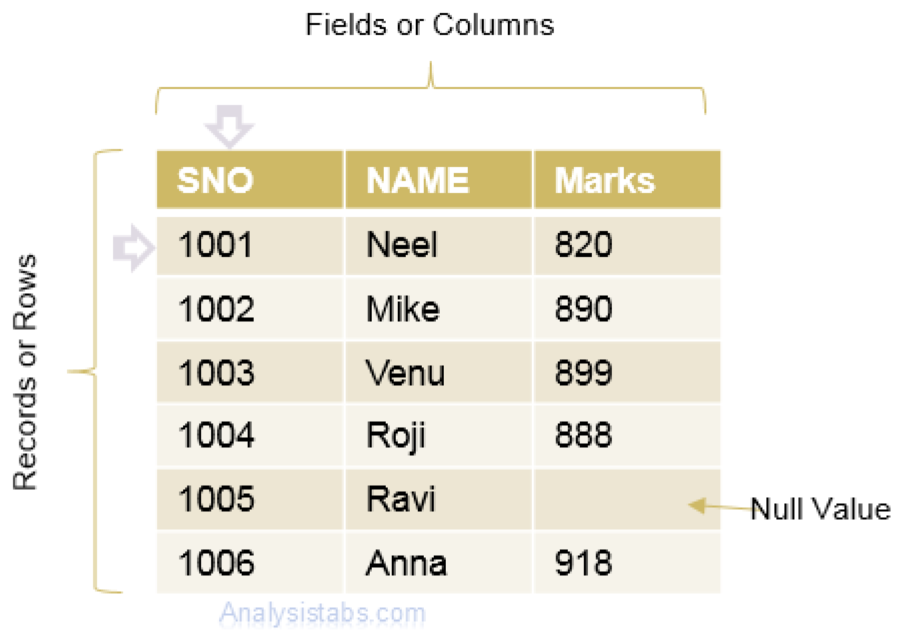
STUDENT + STUDENT_ID (100) + ADDRESS is the right way of getting any cell value.

Codd's Rule 3

Systematic Treatment of NULL

- This rule states about handling the NULLs in the database.
- As database consists of various types of data, each cell will have different datatypes. If any of the cell value is unknown, or not applicable or missing, it cannot be represented as zero or empty.
- It will be always represented as NULL. This NULL should be acting irrespective of the datatype used for the cell. When used in logical or arithmetical operation, it should result the value correctly.

Example Table: Student Marks


Fields or Columns
Records or Rows
Null Value
Analysistabs.com

SNO	NAME	Marks
1001	Neel	820
1002	Mike	890
1003	Venu	899
1004	Roji	888
1005	Ravi	
1006	Anna	918

Codd's Rule 4

Active Online Catalog

- This rule illustrates data dictionary.
- Metadata should be maintained for all the data in the database.
- These metadata stored in the data dictionary should also obey all the characteristics of a database.
- We should be able to access these metadata by using same query language that we use to access the database.

Example:

- SELECT * FROM ALL_TAB;

ALL_TAB is the table which has the table definitions that the user owns and has access.

Codd’s Rule 5

Comprehensive Data Sub-Language Rule

- Any RDBMS database should not be directly accessed.
- It should always be accessed by using some strong query language.
- This query language should be able to access the data, manipulate the data and maintain the consistency , atomicity ,and integrity of the database.

Example:

- SQL is a structured query language which support creating tables / views/ constraints/indexes, accessing the records of tables/views (SELECT), manipulating the records by insert/delete/update, provides security by giving different level of access rights (GRANT and REVOKE) and integrity and consistency by using constraints.

Codd’s Rule 6

View Updating Rule

- Views are the virtual tables created by using queries to show the partial view of the table.
- That is views are subset of table, it is only partial table with few rows and columns.
- This rule states that views are also be able to get updated as we do with its table.

Example:

- If a view is formed as join of 3 tables, changes to view should be reflected in base tables.

Codd's Rule 7

High-level insert, update, and delete

- This rule states that insert, update, and delete operations should be supported for any retrievable set rather than just for a single row in a single table.
- It also perform the operation on multiple row simultaneously .
- There must be delete, updating and insertion at each level of operation. Set operation like union, all union , insertion and minus should also support.

Example:

- Suppose employees got 5% hike in a year. Then their salary has to be updated to reflect the new salary. Since this is the annual hike given to the employees, this increment is applicable for all the employees. Hence, the query should not be written for updating the salary one by one for thousands of employee. A single query should be strong enough to update the entire employee's salary at a time.

Codd’s Rule 8

Physical Data Independence

- The ability to change the physical schema without changing the logical schema is called physical data independence.

- This is saying that users shouldn’t be concerned about how the data is stored or how it’s accessed. In fact, users of the data need only be able to get the basic definition of the data they need.

EXAMPLE:

- A change to the internal schema, such as using different file organization or storage structures, storage devices, or indexing strategy, should be possible without having to change the conceptual or external schemas.

Codd’s Rule 9

Logical Data Independence

The ability to change the logical (conceptual) schema without changing the External schema (User View) is called logical data independence.

EXAMPLE:

The addition or removal of new entities, attributes, or relationships to the conceptual schema should be possible without having to change existing external schemas or having to rewrite existing application programs.

Codd’s Rule 10

Integrity Independence

Database should apply integrity rules by using its query languages.

It should not be dependent on any external factor or application to maintain the integrity.

The keys and constraints in the database should be strong enough to handle the integrity.

A good RDBMS should be independent of the frontend application. It should at least support primary key and foreign key integrity constraints.

Example:

Suppose we want to insert an employee for department 50 using an application.

But department 50 does not exists in the system.

In such case, the application should not perform the task of fetching if department 50 exists, if not insert the department and then inserting the employee.

It should all handled by the database.

Codd’s Rule 11

Distribution Independence

- The database can be located at the user server or at any other network.
- The end user should not be able to know about the database servers.
- He should be able to get the records as if he is pulling the records locally.
- Even if the database is located in different servers, the accessibility time should be comparatively less.

Codd’s Rule 12

Non-Subversion Rule

When a query is fired in the database, it will be converted into low level language so that it can be understood by the underlying systems to retrieve the data.

In such case, when accessing or manipulating the records at low level language, there should not be any loopholes that alter the integrity of the database.

If low level access is allowed to a system, it should not be able to subvert or bypass integrity rules to change the data.

This can be achieved by some sort of looking or encryption.

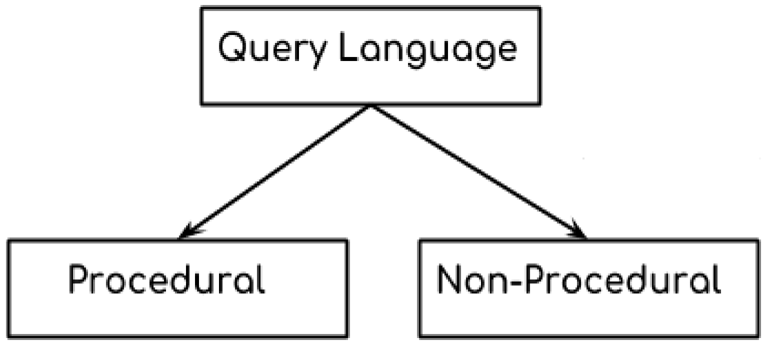
Example:

Update Student’s address query should always be converted into low level language which updates the address record in the student file in the memory.

It should not be updating any other record in the file nor inserting some malicious record into the file/memory.

RELATIONAL QUERY LANGUAGES

- It is the language by which user communicates with the database.
- These relational query languages can be procedural or non-procedural.
- Relational query languages use relational algebra to break the user requests and instruct the DBMS to execute the requests.



RELATIONAL QUERY LANGUAGES(Cont.)

Procedural Query Language

In procedural languages, the program code is written as a sequence of instructions.

User has to specify "what to do" and also "how to do" (step by step procedure).

These instructions are executed in the sequential order. These instructions are written to solve specific problems.

Examples : Relational algebra , FORTRAN, COBOL, ALGOL, BASIC, C and Pascal.

Non-Procedural Query Language

In the non-procedural languages, the user has to specify only "what to do" and not "how to do".

It is also known as an applicative or functional language.

It involves the development of the functions from other functions to construct more complex functions.

Examples : SQL, PROLOG, LISP.

Relational Algebra

A procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.

Six basic operators

- select: σ
- project: ∏
- union: ∪
- set difference: –
- Cartesian product: x
- rename: *ρ*

Select Operation

The **select** operation selects tuples that satisfy a given predicate.

Notation: $\sigma_p(r)$

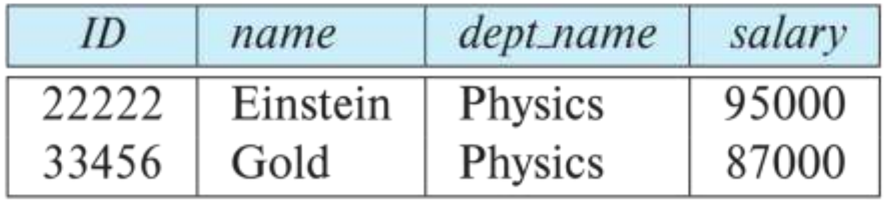
p is called the **selection predicate**

Example: select those tuples of the *instructor* relation where the instructor is in the "Physics" department.

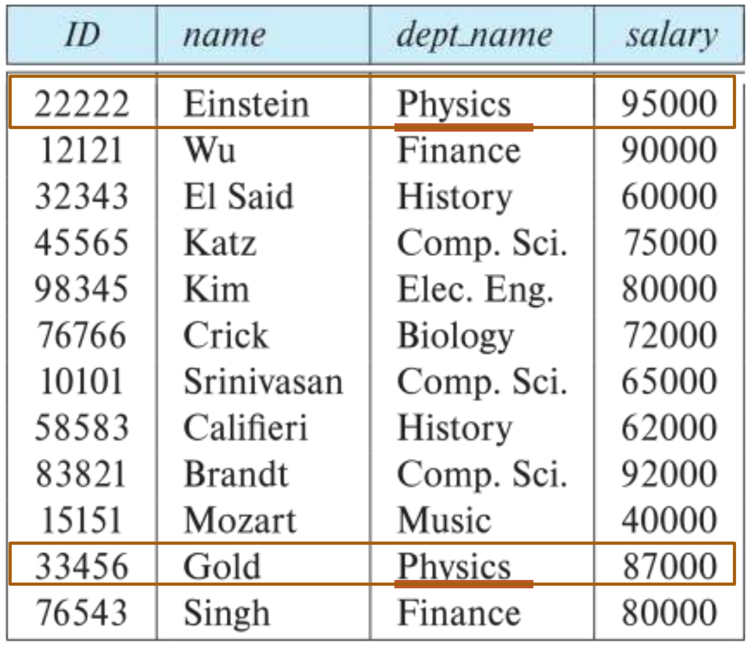
- Query

  $\sigma_{dept\_name=\text{"Physics"}}(instructor)$

- Result



ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000



ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Select Operation (Cont.)

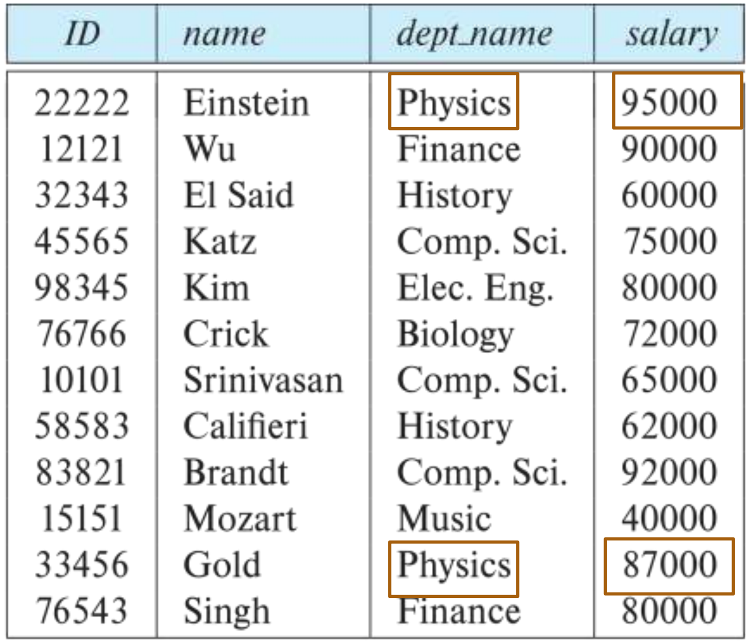
We can use comparison operators like =, ≠, >, ≥. <. ≤ in the selection predicate.

We can combine several predicates into a larger predicate by using the connectives:

∧ (**and**), ∨ (**or**), ¬ (**not**)

Example: Find the instructors in Physics with a salary greater $90,000, we write:

$$\sigma_{\textit{dept\_name}=\text{“Physics”} \wedge \textit{salary} > 90{,}000}(\textit{instructor})$$



ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

ID	name	dept_name	salary
22222	Einstein	Physics	95000

Project Operation

A unary operation that returns its argument relation, with certain attributes left out.

It creates the subset of relation based on the conditions specified. Here, it selects only selected columns/attributes from the relation- vertical subset of relation

Notation:

$$\prod_{A1,A2,A3 \ldots .Ak} (r)$$

where $A_1, A_2, \ldots, A_k$ are attribute names and r is a relation name.

The result is defined as the relation of k columns obtained by erasing the columns that are not listed

Duplicate rows removed from result, since relations are sets

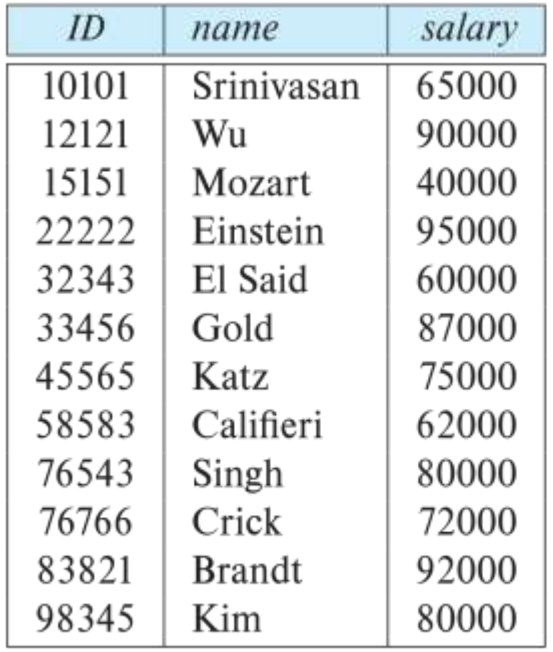
Project Operation Example

Example: eliminate the *dept_name* attribute of *instructor*

Query*:*

$\Pi_{ID,\ name,\ salary}$ (*instructor*)

Result:



ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

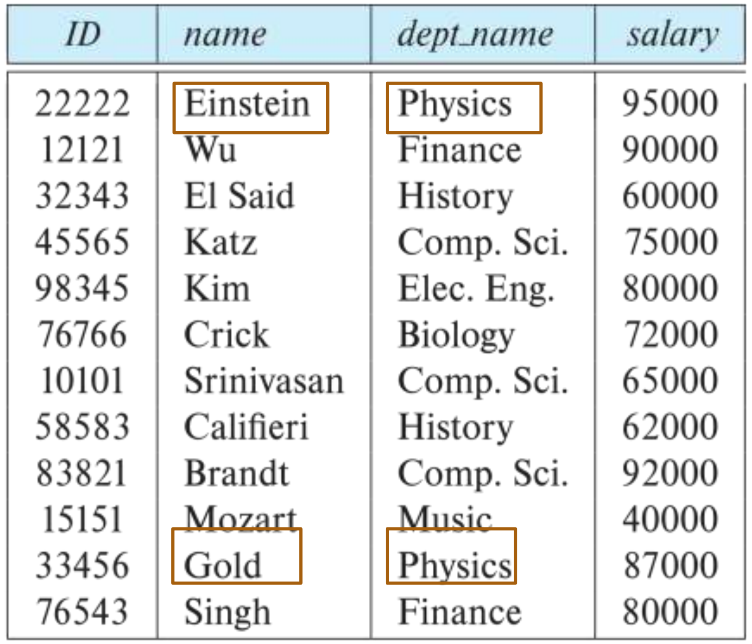
Composition of Relational Operations

The result of a relational-algebra operation is relation and therefore of relational-algebra operations can be composed together into a **relational-algebra expression**.

Consider the query -- Find the names of all instructors in the Physics department.

$$\Pi_{name}(\sigma_{dept\_name = \text{“Physics”}}\ (instructor))$$

Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

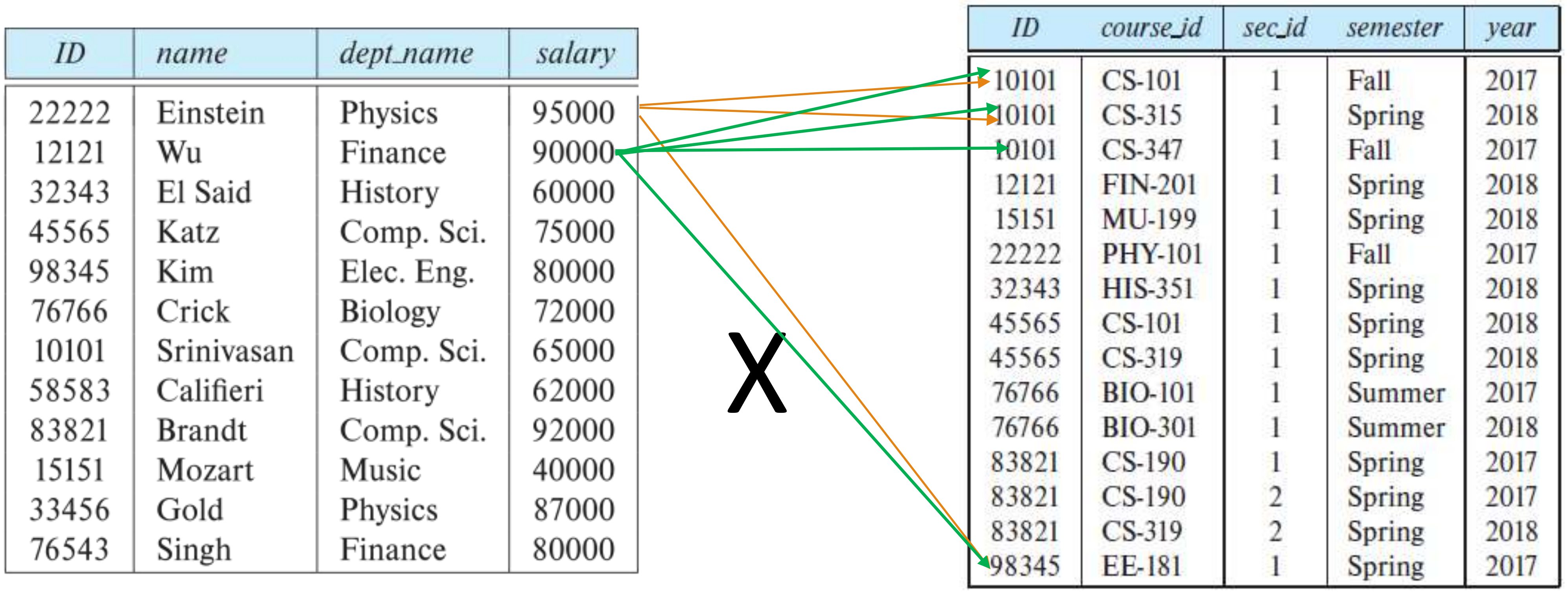


ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

name
Einstein
Gold

Cartesian-Product Operation

- The Cartesian-product operation (denoted by X) allows us to combine information from any two relations.
- R X S returns a relation instance whose schema contains all the fields of R followed by all the fields of S.
- The result of R X S contains tuples <r ,s> (the concatenation of tuples r and s) for each pair of tuples r ϵ R, s ϵ S.
- If relation R has m tuples and relation S has n tuples, then the resultant relation will have m x n tuples.
- For example, if we perform Cartesian product on EMPLOYEE (5 tuples) and DEPT relations (3 tuples), then we will have new tuple with 15 tuples.
- Example: the Cartesian product of the relations *instructor* and t*eaches* is written as: *instructor* X *teaches*
- We construct a tuple of the result out of each possible pair of tuples: one from the *instructor* relation and one from the *teaches* relation
- Since the instructor *ID* appears in both relations, we distinguish between these attribute by attaching to the attribute the name of the relation from which the attribute originally came.
  - *instructor.ID*
  - *teaches.ID*



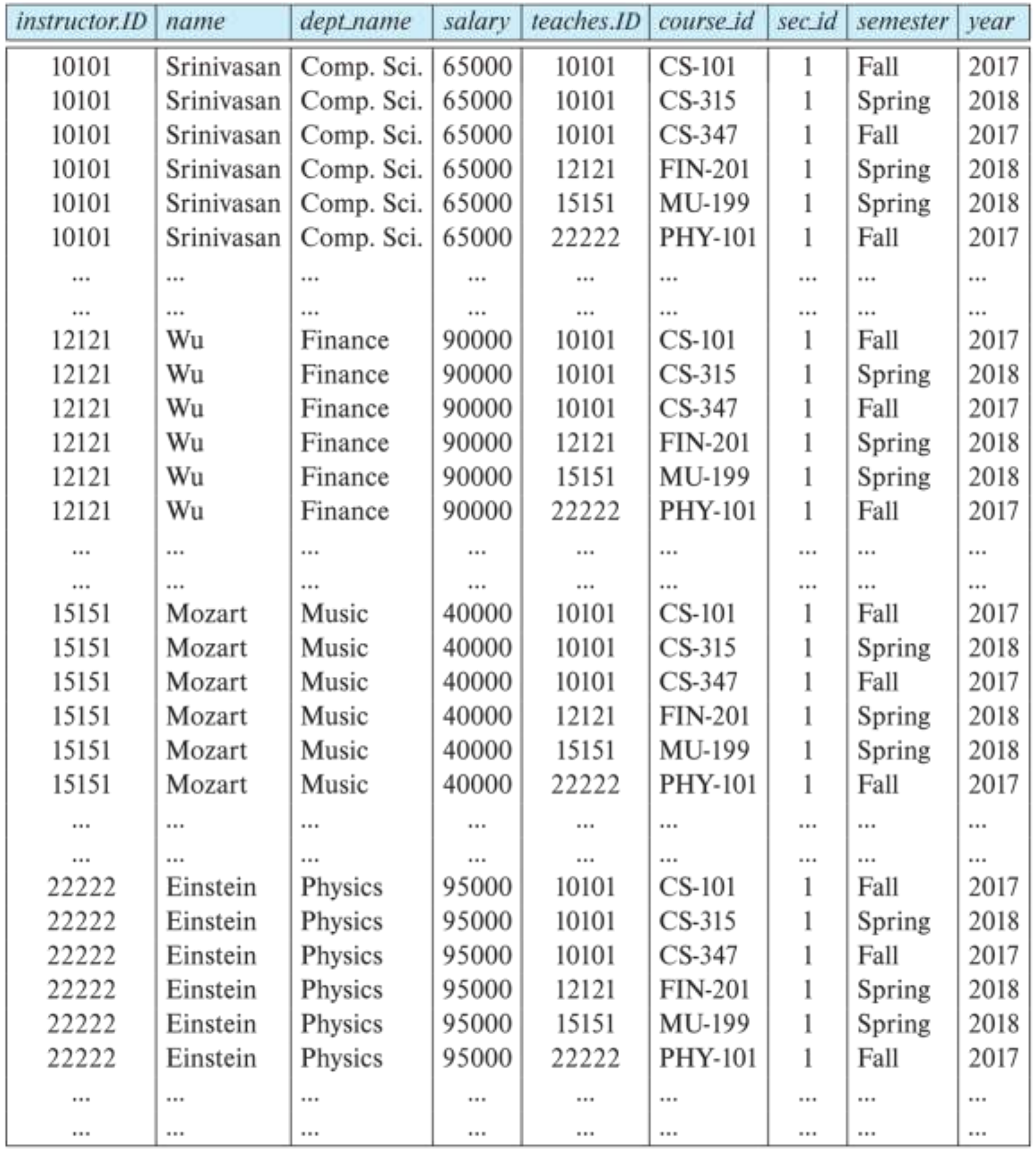
ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Instructor relation

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	2018
45565	CS-101	1	Spring	2018
45565	CS-319	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

The *teaches* relation.

instructor X *teaches*



instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2017
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2018
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2017
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2018
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Join Operation

- The Cartesian-Product

  instructor X *teaches*

  associates every tuple of instructor with every tuple of teaches.
  - Most of the resulting rows have information about instructors who did NOT teach a particular course.

To get only those tuples of "*instructor* X *teaches* " that pertain to instructors and the courses that they taught, we write:

$$\sigma_{\textit{instructor.id} = \textit{teaches.id}} (\textit{instructor} \times \textit{teaches}))$$

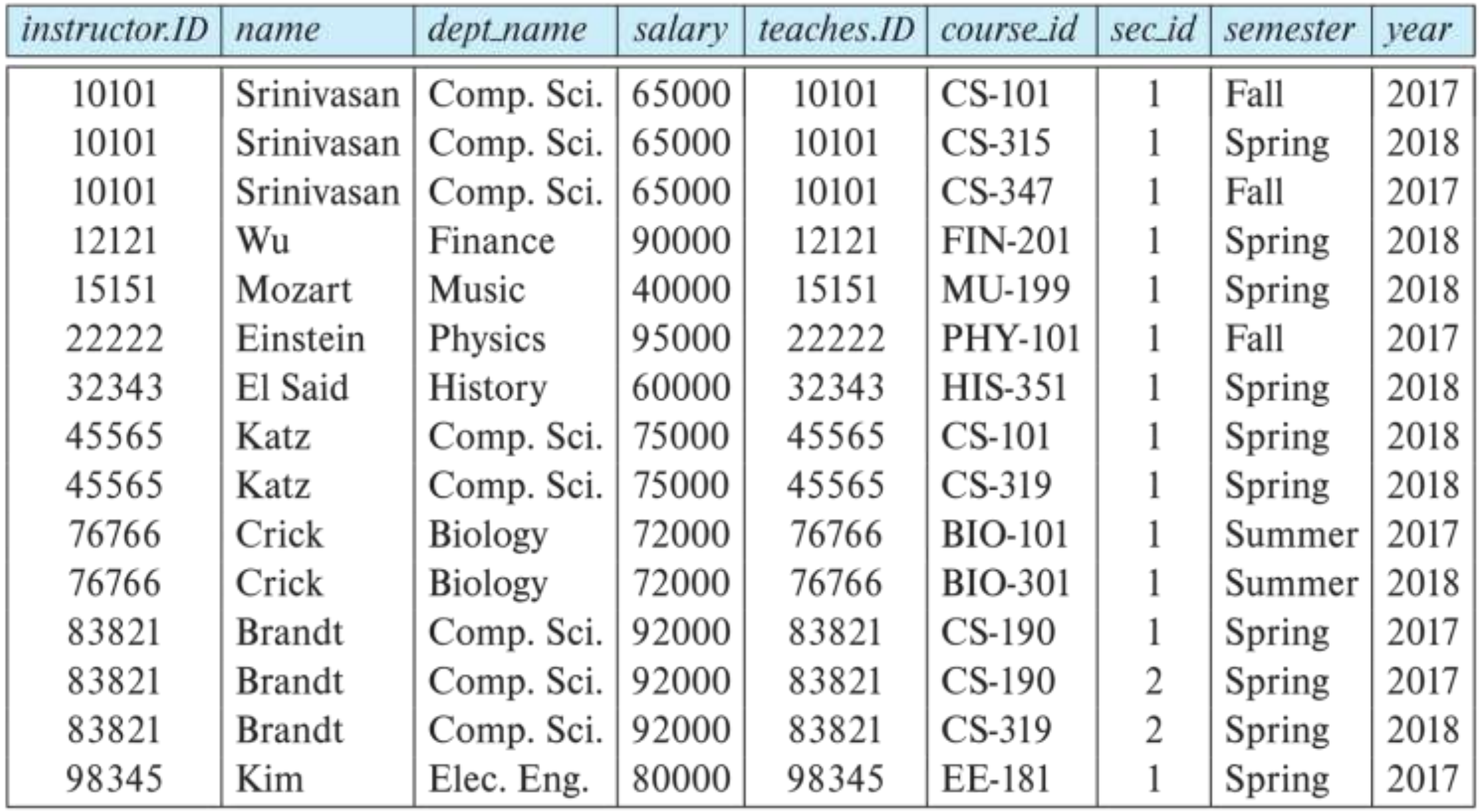
- We get only those tuples of "*instructor* X *teaches*" that pertain to instructors and the courses that they taught.

The result of this expression, shown in the next slide

Join Operation (Cont.)

The table corresponding to:

$\sigma_{\textit{instructor.id} = \textit{teaches.id}}(\textit{instructor} \times \textit{teaches}))$



instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
32343	El Said	History	60000	32343	HIS-351	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-101	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-319	1	Spring	2018
76766	Crick	Biology	72000	76766	BIO-101	1	Summer	2017
76766	Crick	Biology	72000	76766	BIO-301	1	Summer	2018
83821	Brandt	Comp. Sci.	92000	83821	CS-190	1	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-190	2	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-319	2	Spring	2018
98345	Kim	Elec. Eng.	80000	98345	EE-181	1	Spring	2017

Join Operation (Cont.)

The **join** operation allows us to combine a select operation and a Cartesian-Product operation into a single operation.

Consider relations *r* (*R*) and *s* (*S*)

Let "theta" be a predicate on attributes in the schema R "union" S. The join operation r $\bowtie_\theta$ s is defined as follows:

$$r \bowtie_\theta s = \sigma_\theta\ (r \times s)$$

Thus

$$\sigma_{instructor.id = teaches.id}\ (instructor\ \text{x}\ teaches\ ))$$

Can equivalently be written as

$$instructor \bowtie_{Instructor.id = teaches.id} teaches.$$

Union Operation

The union operation allows us to combine two relations

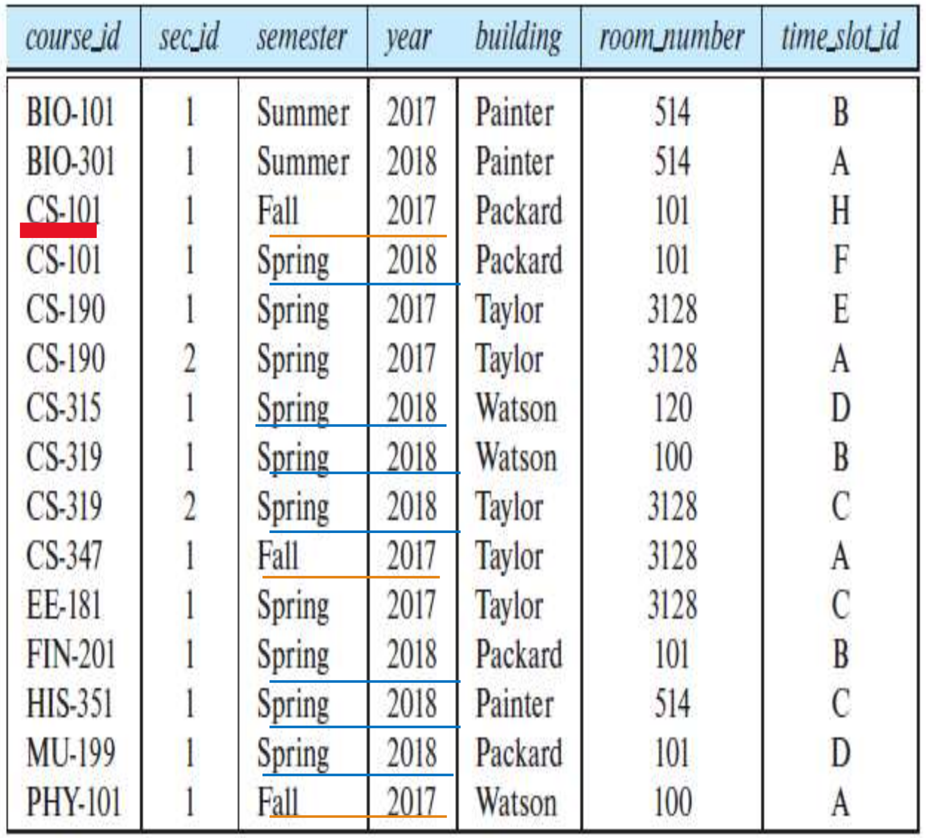
Notation: $r \cup s$

For $r \cup s$ to be valid.

1. r, s must have the *same* **arity** (same number of attributes)
2. The attribute domains must be **compatible** (example: 2nd column of r deals with the same type of values as does the 2nd column of s)

Example: to find all courses taught in the Fall 2017 semester, or in the Spring 2018 semester, or in both

$$\Pi_{course\_id}(\sigma_{semester=\text{"Fall"} \land year=2017}(section)) \cup$$
$$\Pi_{course\_id}(\sigma_{semester=\text{"Spring"} \land year=2018}(section))$$



course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A

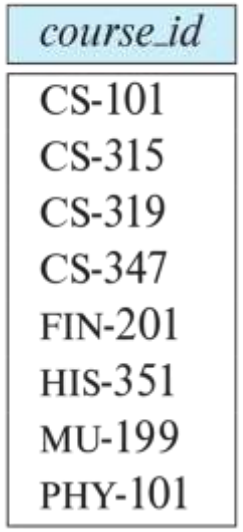
The *section* relation.

Union Operation (Cont.)

Result of:

$\Pi_{course\_id} (\sigma_{semester=\text{“Fall”} \land year=2017} (section)) \cup$

$\Pi_{course\_id} (\sigma_{semester=\text{“Spring”} \land year=2018} (section))$



course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Set-Intersection Operation

The set-intersection operation allows us to find tuples that are in both the input relations.

Notation: $r \cap s$

Assume:

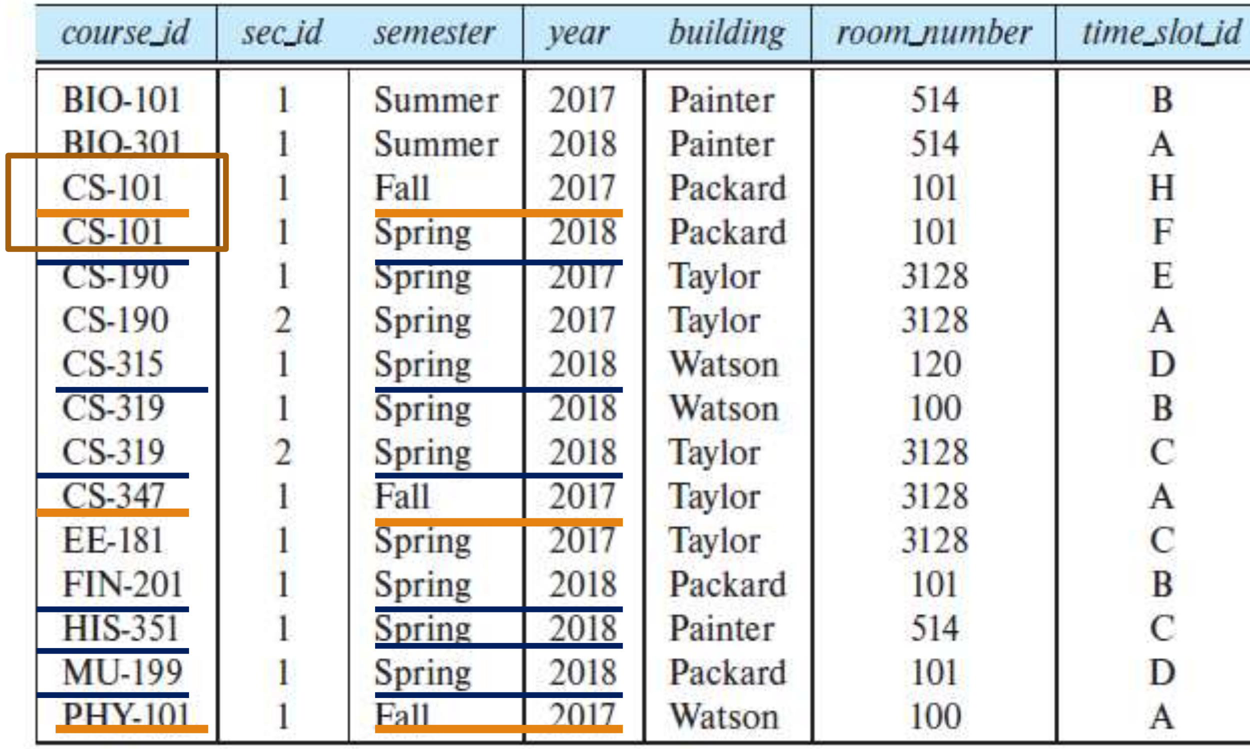
- *r*, *s* have the *same arity*
- attributes of *r* and *s* are compatible

Example: Find the set of all courses taught in both the Fall 2017 and the Spring 2018 semesters.

$$\Pi_{course\_id}(\sigma_{semester=\text{"Fall"}\ \land\ year=2017}(section)) \cap \Pi_{course\_id}(\sigma_{semester=\text{"Spring"}\ \land\ year=2018}(section))$$

- Result

course_id
CS-101



course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A

The *section* relation.

Set-Difference Operation

The set-difference operation allows us to find tuples tha are in one relation but are not in another.

Notation *r* – *s*

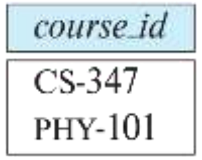
Set differences must be taken between **compatible** relations.

- *r* and *s* must have the same arity
- attribute domains of *r* and *s* must be compatible

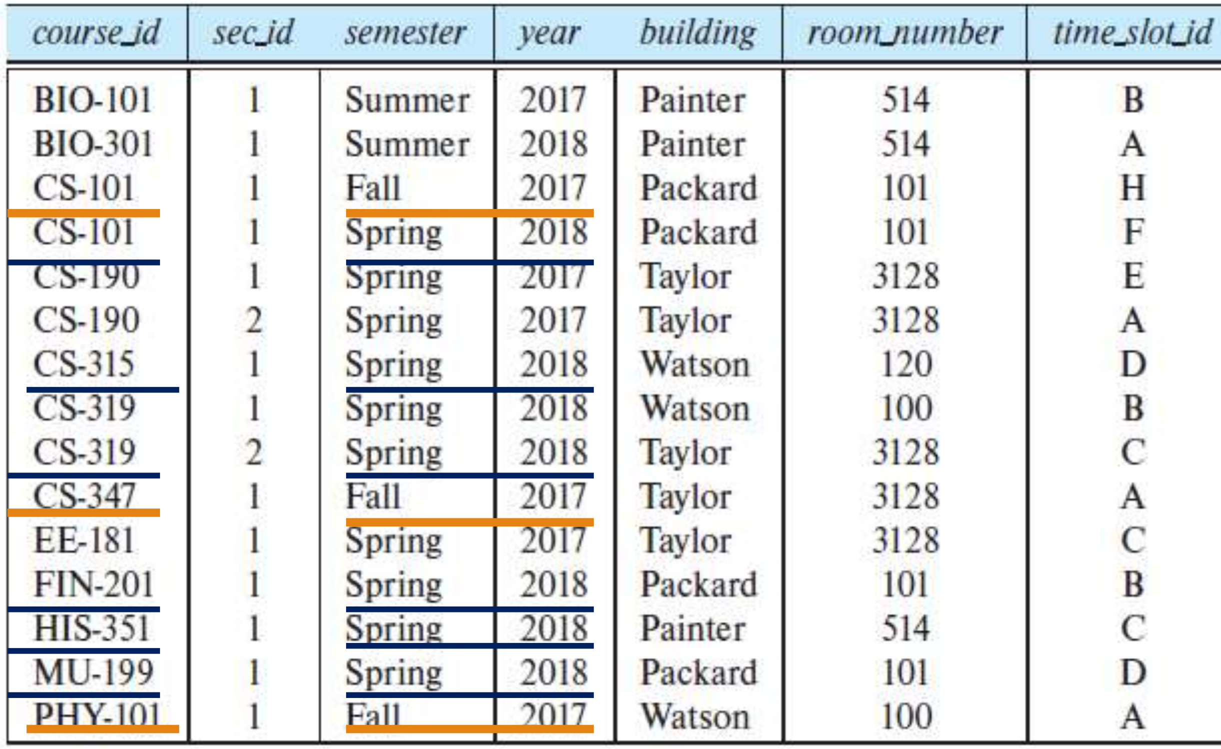
Example: to find all courses taught in the Fall 2017 semester, but not in the Spring 2018 semester

$$\Pi_{course\_id}\ (\sigma_{semester=\text{“Fall”}\ \land\ year=2017}\ (section))\ -$$
$$\Pi_{course\_id}\ (\sigma_{semester=\text{“Spring”}\ \land\ year=2018}\ (section))$$

- Result



course_id
CS-347
PHY-101



course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A

The *section* relation.

The Assignment Operation

- It is convenient at times to write a relational-algebra expression by assigning parts of it to temporary relation variables.
- The assignment operation is denoted by $\leftarrow$ and works like assignment in a programming language.
- Example: Find all instructor in the “Physics” and Music department.
  - $Physics \leftarrow \sigma_{dept\_name=\text{“Physics”}}(instructor)$
  - $Music \leftarrow \sigma_{dept\_name=\text{“Music”}}(instructor)$
  - $Physics \cup Music$
- With the assignment operation, a query can be written as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as the result of the query.

The Rename Operation

The results of relational-algebra expressions do not have a name that we can use to refer to them. The rename operator, ρ, is provided for that purpose

The expression:

$$\rho_x (E)$$

returns the result of expression E under the name x

Another form of the rename operation:

$$\rho_{x(A1,A2, .. An)} (E)$$

Equivalent Queries

There is more than one way to write a query in relational algebra.

Example: Find information about courses taught by instructors in the Physics department with salary greater than 90,000

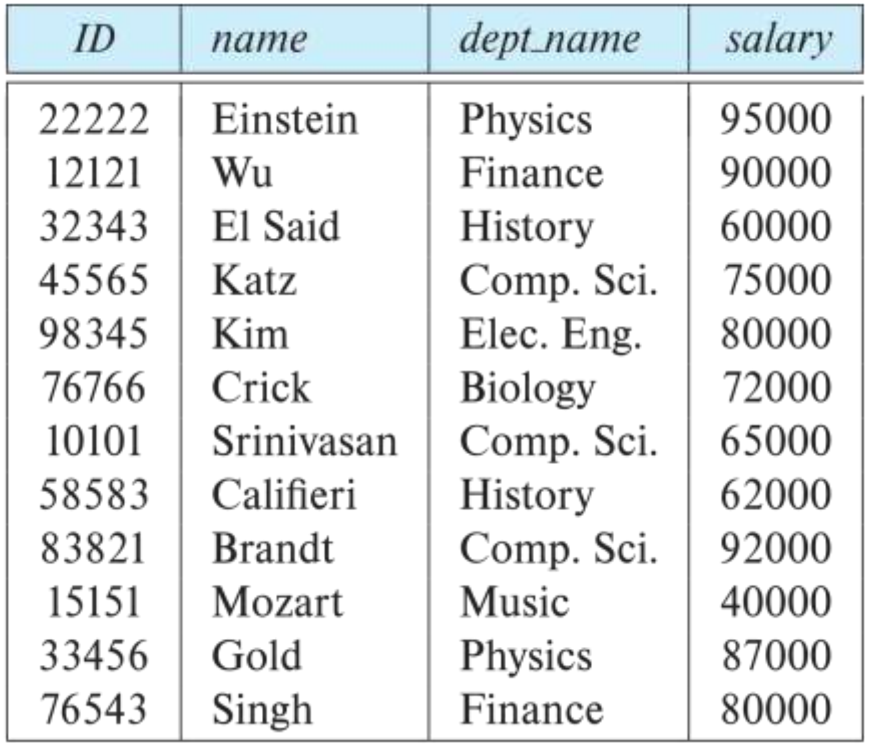
Query 1

$$\sigma_{dept\_name=\text{“Physics”} \wedge salary > 90{,}000}(instructor)$$

Query 2

$$\sigma_{dept\_name=\text{“Physics”}}(\sigma_{salary > 90.000}(instructor))$$

The two queries are not identical; they are, however, equivalent -- they give the same result on any database.



ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Equivalent Queries

There is more than one way to write a query in relational algebra.

Example: Find information about courses taught by instructors in the Physics department

Query 1

$$\sigma_{dept\_name= \text{“Physics”}}(instructor \bowtie_{instructor.ID = teaches.ID} teaches)$$

Query 2

$$(\sigma_{dept\_name= \text{“Physics”}}(instructor)) \bowtie_{instructor.ID = teaches.ID} teaches$$

The two queries are not identical; they are, however, equivalent -- they give the same result on any database.

Design Phases

Initial phase -- characterize fully the data needs of the prospective database users.

Second phase -- choosing a data model

- Applying the concepts of the chosen data model
- Translating these requirements into a conceptual schema of the database.
- A fully developed conceptual schema indicates the functional requirements of the enterprise.
  - Describe the kinds of operations (or transactions) that will be performed on the data.

- Final Phase -- Moving from an abstract data model to the implementation of the database
  - Logical Design – Deciding on the database schema.
    - Database design requires that we find a "good" collection of relation schemas.
    - Business decision – What attributes should we record in the database?
    - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
  - Physical Design – Deciding on the physical layout of the database

Design Alternatives

- In designing a database schema, we must ensure that we avoid two major pitfalls:
  - Redundancy: a bad design may result in repeat information.
    - Redundant representation of information may lead to data inconsistency among the various copies of information
  - Incompleteness: a bad design may make certain aspects of the enterprise difficult or impossible to model.
- Avoiding bad designs is not enough. There may be a large number of good designs from which we must choose.

ER model -- Database Modeling

The ER data mode was developed to facilitate database design by allowing specification of an **enterprise schema** that represents the overall logical structure of a database.

The ER data model employs three basic concepts:

- entity sets,
- relationship sets,
- attributes.

The ER model also has an associated diagrammatic representation, the **ER diagram**, which can express the overall logical structure of a database graphically.

Entity Sets

An **entity** is an object that exists and is distinguishable from other objects.

- Example: specific person, company, event, plant

An **entity set** is a set of entities of the same type that share the same properties.

- Example: set of all persons, companies, trees, holidays

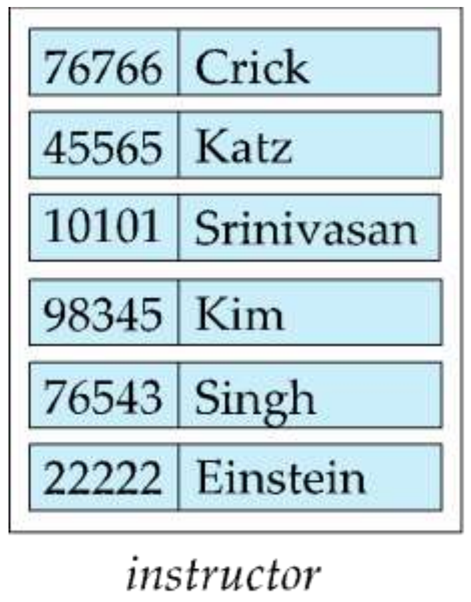
An entity is represented by a set of attributes; i.e., descriptive properties possessed by all members of an entity set.

- Example:

    instructor = (*ID, name, salary*)
    course= (*course_id, title, credits*)

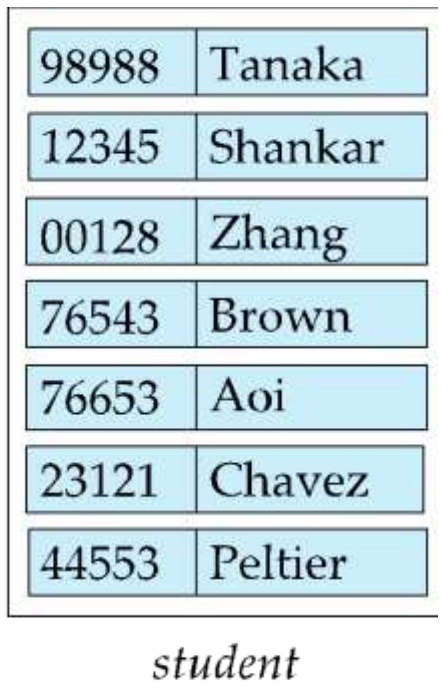
A subset of the attributes form a **primary key** of the entity set; i.e., uniquely identifying each member of the set.

Entity Sets -- *instructor* and *student*



instructor_ID	instructor_name
76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor



student-ID	student_name
98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

Representing Entity sets in ER Diagram

- Entity sets can be represented graphically as follows:
  - Rectangles represent entity sets.
  - Attributes listed inside entity rectangle
  - Underline indicates primary key attributes

Relationship Sets

- A **relationship** is an association among several entities

  Example:

44553 (Peltier)	*advisor*	22222 (Einstein)
student entity	relationship set	*instructor* entity

- A **relationship set** is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

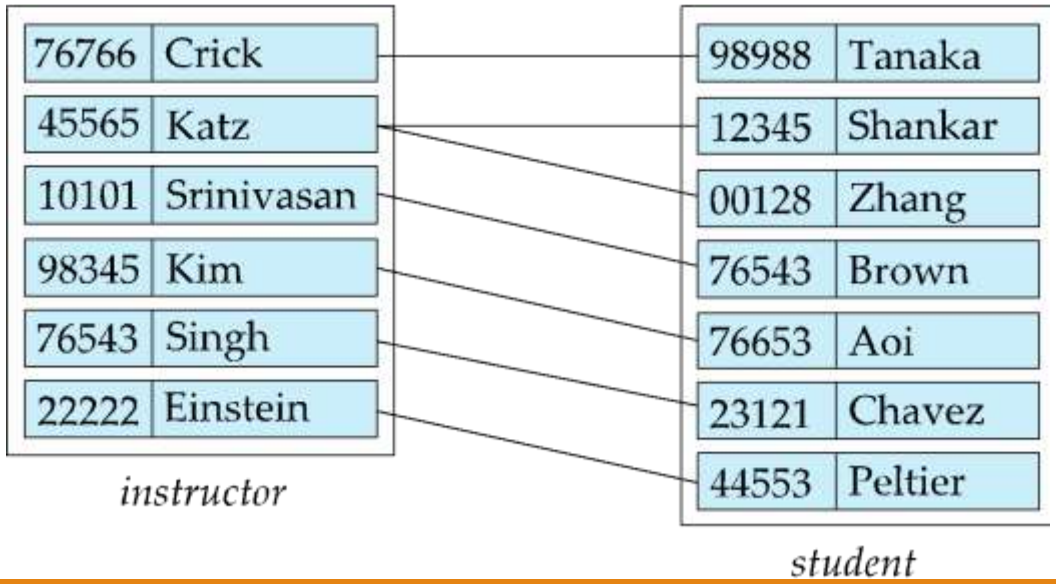
$$\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

  where $(e_1, e_2, \ldots, e_n)$ is a relationship
  - Example:

$$(44553,22222) \in \textit{advisor}$$

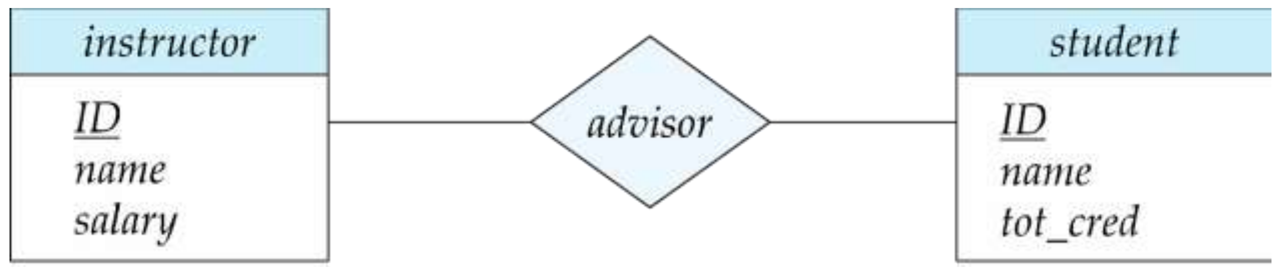
Relationship Sets (Cont.)

- Example: we define the relationship set *advisor* to denote the associations between students and the instructors who act as their advisors.
- Pictorially, we draw a line between related entities.



Representing Relationship Sets via ER Diagrams

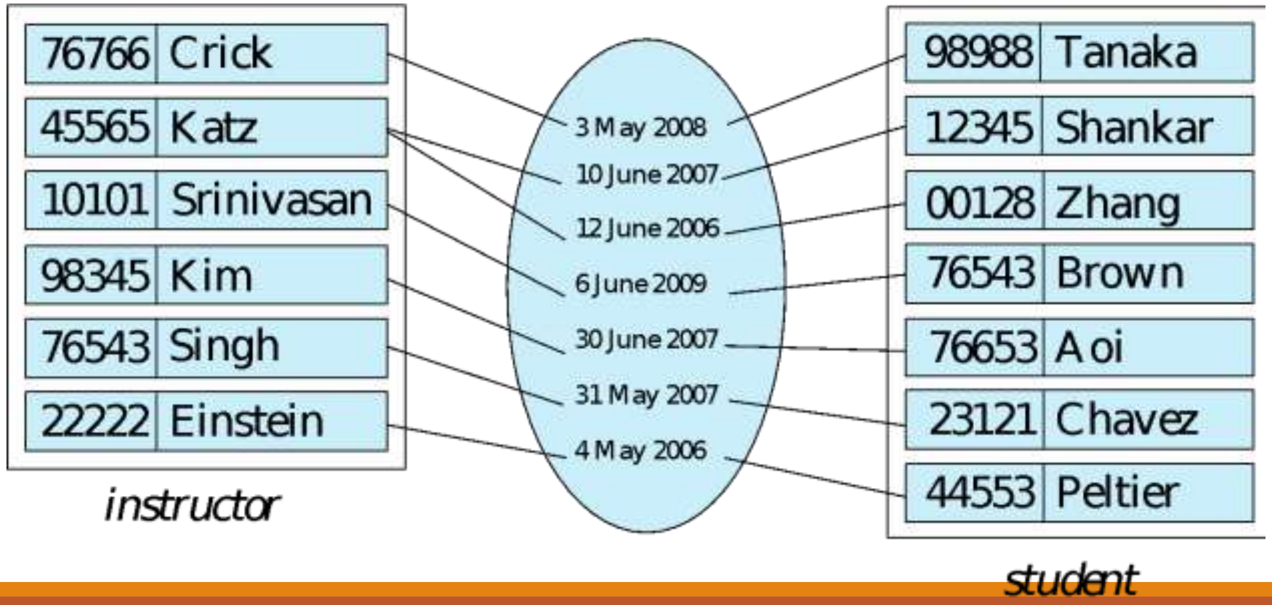
- Diamonds represent relationship sets.



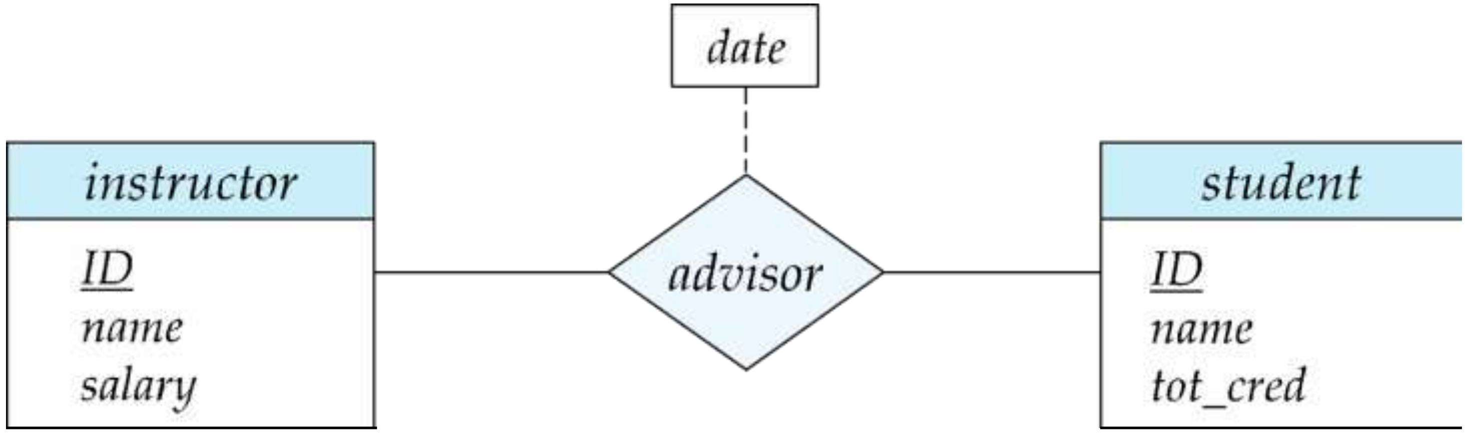
Relationship Sets (Cont.)

An attribute can also be associated with a relationship set.

For instance, the *advisor* relationship set between entity sets *instructor* and *student* may have the attribute *date* which tracks when the student started being associated with the advisor



Relationship Sets with Attributes

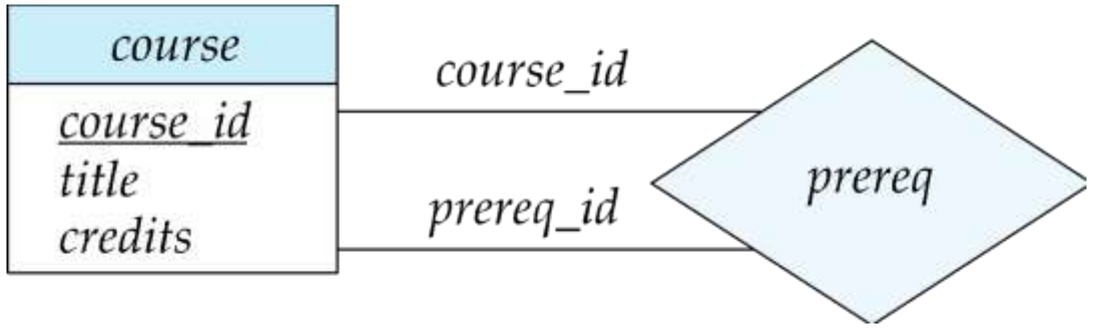


Roles

Entity sets of a relationship need not be distinct

- Each occurrence of an entity set plays a "role" in the relationship

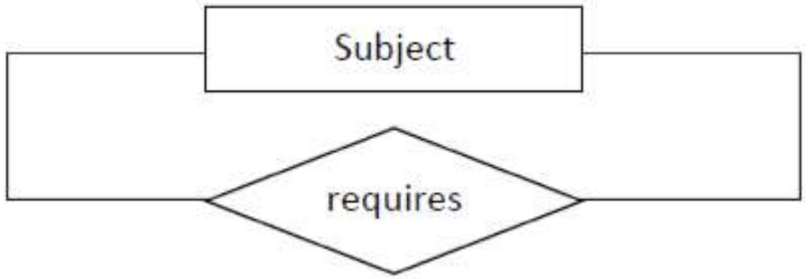
The labels "*course_id*" and "*prereq_id*" are called **roles**.



Degree of a Relationship Set

Unary Relationship

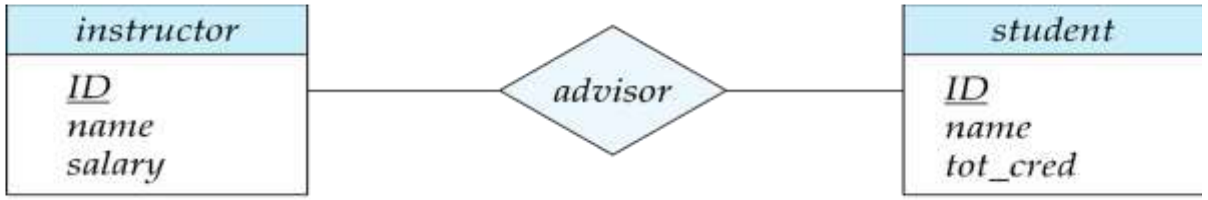
The function that an entity plays in a relationship is called that entity’s role. The same entity set participates in a relationship set more than once in different roles is called ‘recursive relationship’. Recursive relationships are sometimes called ‘unary relationship’.



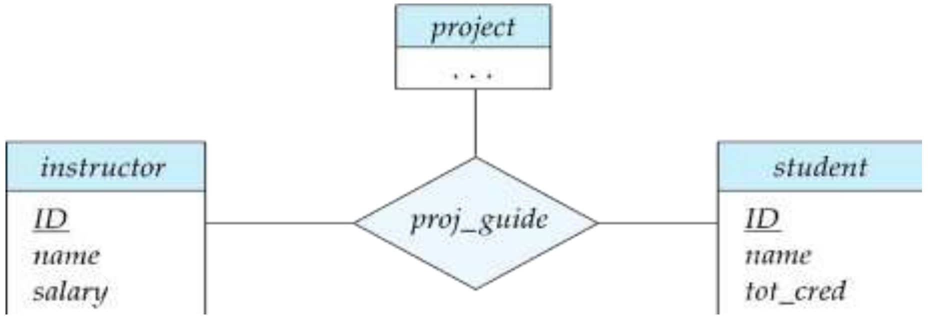
Degree of a Relationship Set

Binary Relationship

The relationship set that involves only two entity sets is called ‘binary relationship set’. A binary relationship exists when two entities are associated.



Ternary Relationship
A ternary relationship exists when there are three entities associated. The entities teacher, subject and student are related using a ternary relationship

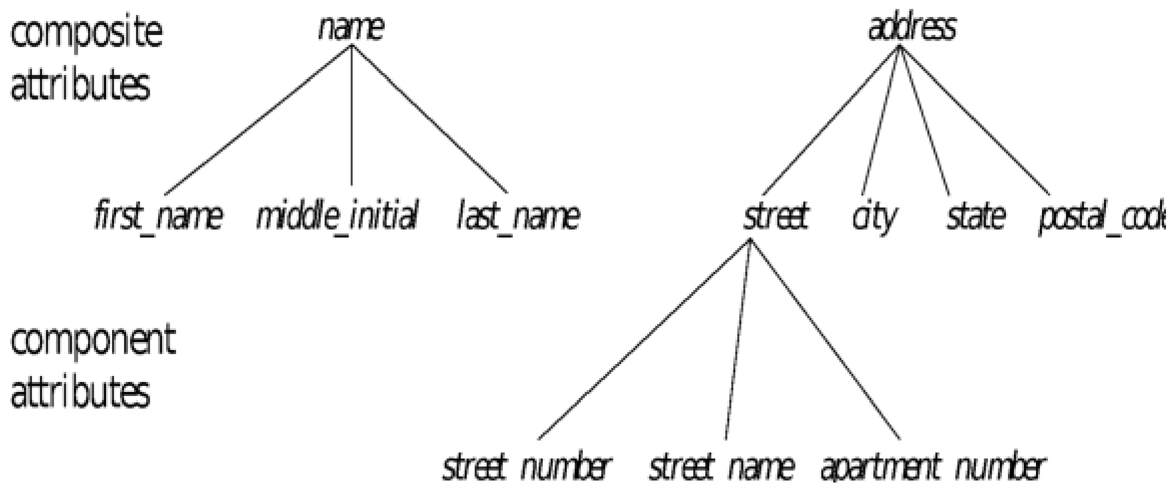


Attributes

- An attribute of an entity set is a function that maps from the entity set into a domain.
- For each attribute, there is a set of permitted values called ‘domain’ or ‘value set’ of that attribute.
- An attribute can be characterized by the following attribute types,

1. Simple and composite attributes.
2. Single valued and multi-valued attributes.
3. Derived attribute.

Composite Attributes



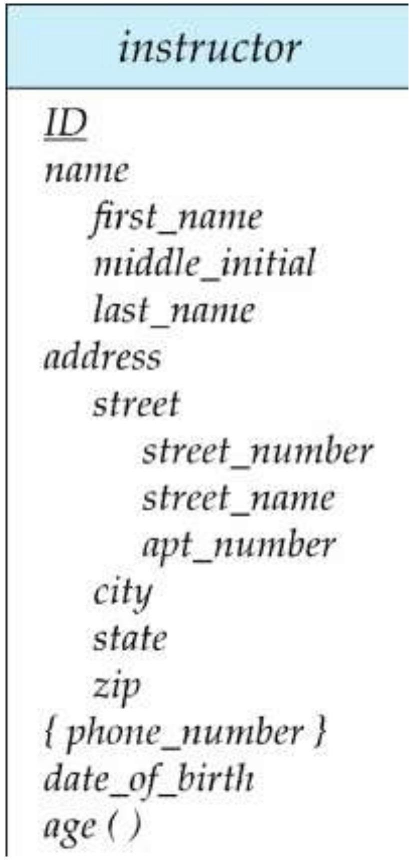
- An attribute composed of a single component with an independent existence is called 'simple attribute or atomic attribute'. Simple attribute cannot be further subdivided into smaller components.

  e.g.: gender of a staff entity.

  An attribute composed of multiple components each with an independent existence is called 'composite attribute'.

  e.g.: The 'address' attribute of the branch entity, can be subdivided into street, city, country and postal code attributes.

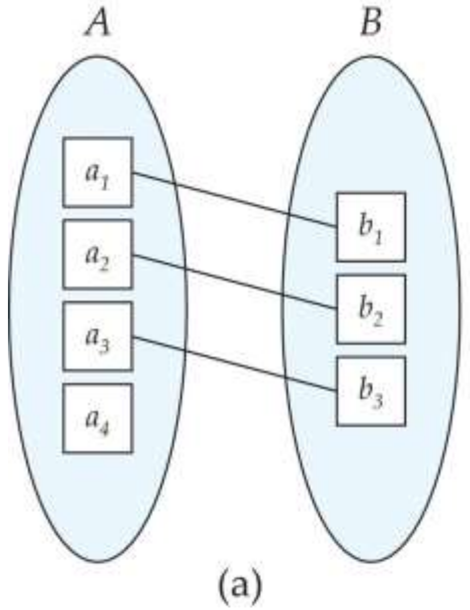
Representing Complex Attributes in ER Diagram



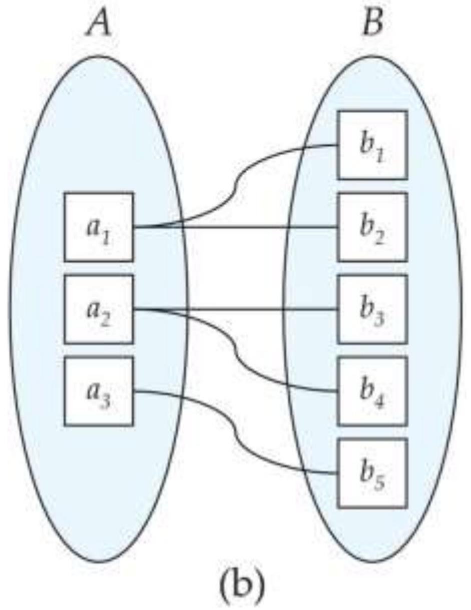
Mapping Cardinality Constraints

- Express the number of entities to which another entity can be associated via a relationship set.
- Most useful in describing binary relationship sets.
- For a binary relationship set the mapping cardinality must be one of the following types:
  - One to one
  - One to many
  - Many to one
  - Many to many

Mapping Cardinalities



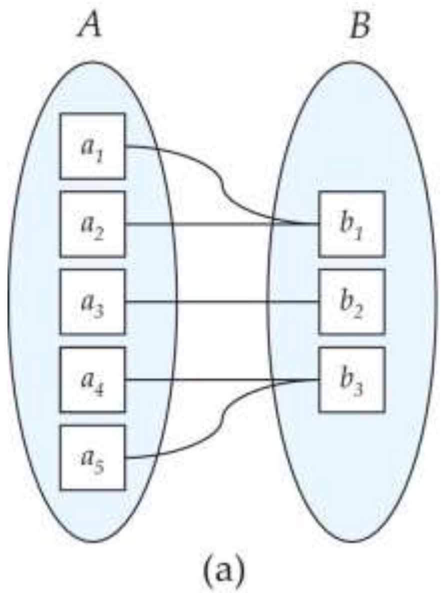
One to one



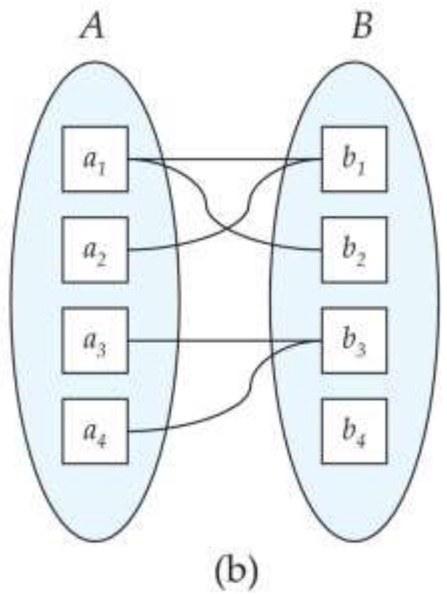
One to many

Note: Some elements in *A* and *B* may not be mapped to any elements in the other set

Mapping Cardinalities



Many to one

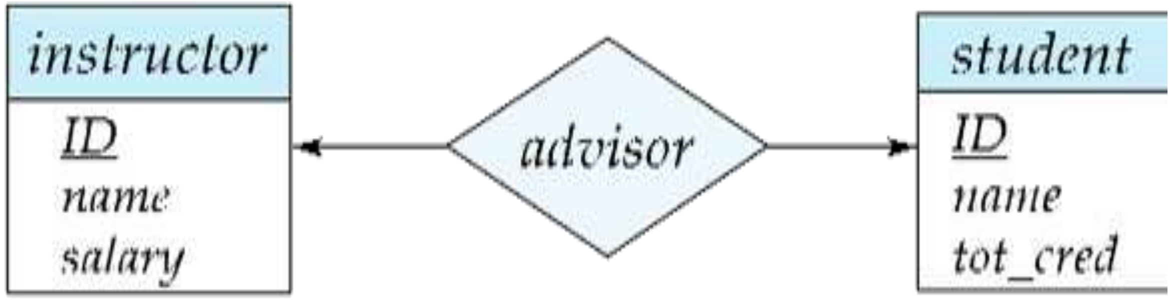


Many to many

Note: Some elements in A and B may not be mapped to any elements in the other set

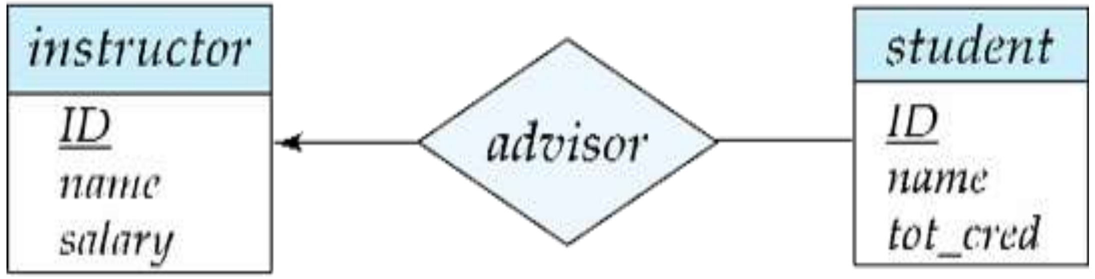
Representing Cardinality Constraints in ER Diagram

- We express cardinality constraints by drawing either a directed line (→), signifying "one," or an undirected line (—), signifying "many," between the relationship set and the entity set.
- One-to-one relationship between an *instructor* and a *student* :
  - A student is associated with at most one *instructor* via the relationship *advisor*
  - A *student* is associated with at most one *department* via *stud_dept*



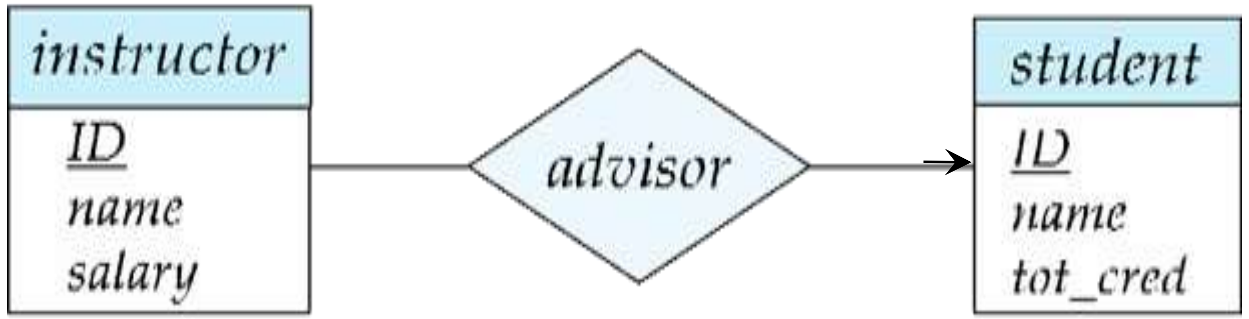
One-to-Many Relationship

- one-to-many relationship between an *instructor* and a *student*
  - an instructor is associated with several (including 0) students via *advisor*
  - a student is associated with at most one instructor via advisor,



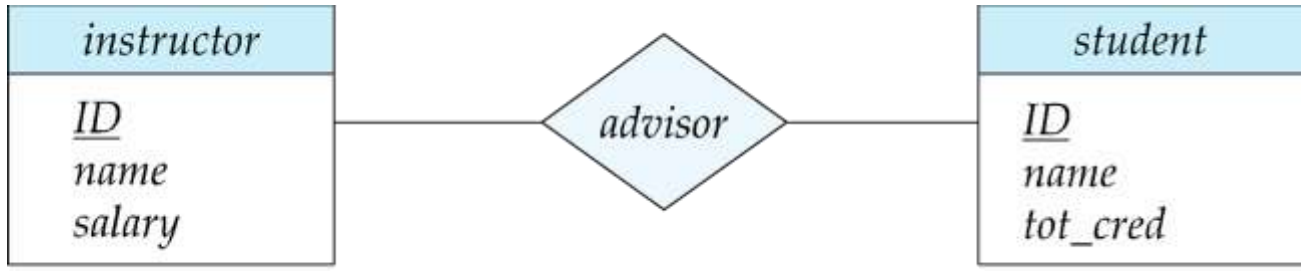
Many-to-One Relationships

- In a many-to-one relationship between an *instructor* and a *student,*
  - an instructor is associated with at most one student via *advisor*,
  - and a student is associated with several (including 0) instructors via *advisor*



Many-to-Many Relationship

- An instructor is associated with several (possibly 0) students via *advisor*
- A student is associated with several (possibly 0) instructors via *advisor*



Total and Partial Participation

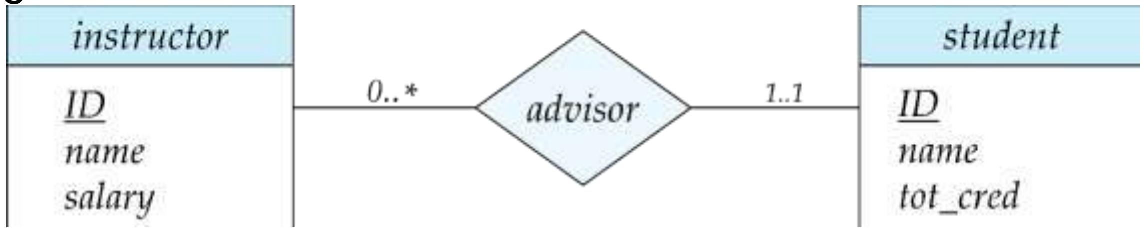
- **Total participation** (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set

  participation of *student* in *advisor* relation is total

  - every *student* must have an associated instructor
- **Partial participation**: some entities may not participate in any relationship in the relationship set
  - Example: participation of *instructor* in *advisor* is partial

Notation for Expressing More Complex Constraints

- A line may have an associated minimum and maximum cardinality, shown in the form *l..h*, where *l* is the minimum and *h* the maximum cardinality
  - A minimum value of 1 indicates total participation.
  - A maximum value of 1 indicates that the entity participates in at most one relationship
  - A maximum value of * indicates no limit.
- Example



  - Instructor can advise 0 or more students. A student must have 1 advisor; cannot have multiple advisors

Cardinality Constraints on Ternary Relationship

- We allow at most one arrow out of a ternary (or greater degree) relationship to indicate a cardinality constraint
- For example, an arrow from *proj_guide* to *instructor* indicates each student has at most one guide for a project
- If there is more than one arrow, there are two ways of defining the meaning.
  - For example, a ternary relationship *R* between *A*, *B* and *C* with arrows to *B* and *C* could mean
    1. Each *A* entity is associated with a unique entity from B and *C* or
    2. Each pair of entities from (*A, B*) is associated with a unique *C* entity, and each pair (*A, C*) is associated with a unique *B*
  - Each alternative has been used in different formalisms
  - To avoid confusion, we outlaw more than one arrow

Primary Key

- Primary keys provide a way to specify how entities and relations are distinguished. We will consider:
  - Entity sets
  - Relationship sets.
  - Weak entity sets

Primary key for Entity Sets

- By definition, individual entities are distinct.
- From database perspective, the differences among them must be expressed in terms of their attributes.
- The values of the attribute values of an entity must be such that they can uniquely identify the entity.
    - No two entities in an entity set are allowed to have exactly the same value for all attributes.
- A key for an entity is a set of attributes that suffice to distinguish entities from each other

Primary Key for Relationship Sets

- To distinguish among the various relationships of a relationship set we use the individual primary keys of the entities in the relationship set.
  - Let *R* be a relationship set involving entity sets E1, E2, .. En
  - The primary key for R is consists of the union of the primary keys of entity sets E1, E2, ..En
  - If the relationship set *R* has attributes a1, a2, .., am associated with it, then the primary key of *R* also includes the attributes a1, a2, .., am
- Example: relationship set “advisor”.
  - The primary key consists of *instructor.ID* and *student.ID*

The choice of the primary key for a relationship set depends on the mapping cardinality of the relationship set.

Choice of Primary key for Binary Relationship

- Many-to-Many relationships.   The preceding union of the primary keys is a minimal superkey and is chosen as the primary key.
- One-to-Many relationships . The primary key of the “Many” side is a minimal superkey and is used as the primary key.
- Many-to-one relationships. The primary key of the “Many” side is a minimal superkey and is used as the primary key.
- One-to-one relationships. The primary key of either one of the participating entity sets forms a minimal superkey, and either one can be chosen as the primary key.

Weak Entity Sets

- A **weak entity set** is one whose existence is dependent on another entity, called its **identifying entity**
- Instead of associating a primary key with a weak entity, we use the identifying entity, along with extra attributes called **discriminator** to uniquely identify a weak entity.
- An entity set that is not a weak entity set is termed a **strong entity set**.
- Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be **existence dependent** on the identifying entity set.
- The identifying entity set is said to **own** the weak entity set that it identifies.
- The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**.
- Note that the relational schema we eventually create from the entity set *section* does have the attribute *course_id*, for reasons that will become clear later, even though we have dropped the attribute *course_id* from the entity set *section.*

Expressing Weak Entity Sets

- In E-R diagrams, a weak entity set is depicted via a double rectangle.
- We underline the discriminator of a weak entity set with a dashed line.
- The relationship set connecting the weak entity set to the identifying strong entity set is depicted by a double diamond.
- Primary key for *section* – (*course_id, sec_id, semester, year*)

E-R Diagram for a University Enterprise

Reduction to Relation Schemas

- Entity sets and relationship sets can be expressed uniformly as *relation schemas* that represent the contents of the database.
- A database which conforms to an E-R diagram can be represented by a collection of schemas.
- For each entity set and relationship set there is a unique schema that is assigned the name of the corresponding entity set or relationship set.
- Each schema has a number of columns (generally corresponding to attributes), which have unique names.

Representing Entity Sets

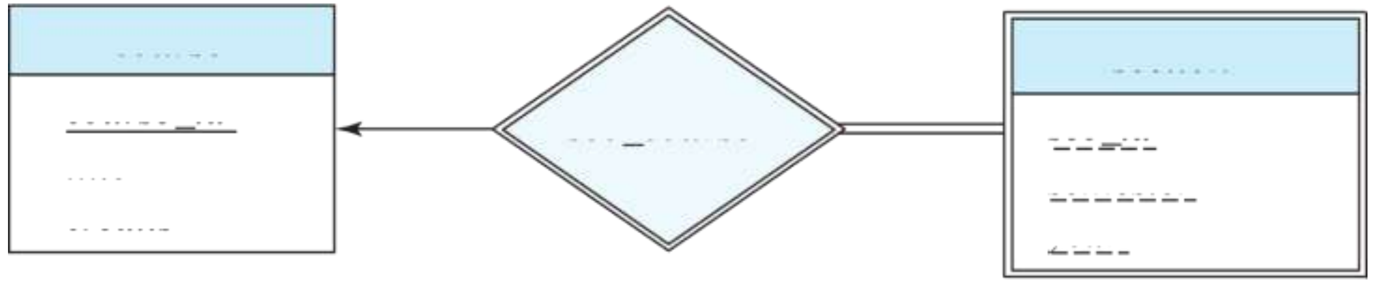
- A strong entity set reduces to a schema with the same attributes

  student(<u>ID</u>, name, tot_cred)

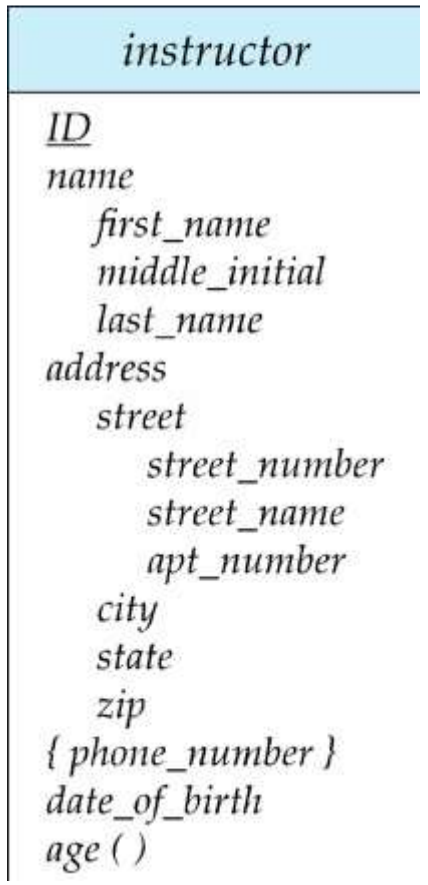
- A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set

  section (<u>course_id, sec_id, sem, year</u>)

- Example



Representation of Entity Sets with Composite Attributes



- Composite attributes are flattened out by creating a separate attribute for each component attribute
  - Example: given entity set *instructor* with composite attribute *name* with component attributes *first_name* and *last_name* the schema corresponding to the entity set has two attributes *name_first_name* and *name_last_name*
    - Prefix omitted if there is no ambiguity (*name_first_name* could be *first_name)*
- Ignoring multivalued attributes, extended instructor schema is
  - *instructor(ID,*
    first_name, middle_initial, last_name,
    street_number, street_name,
    apt_number, city, state, zip_code,
    date_of_birth)

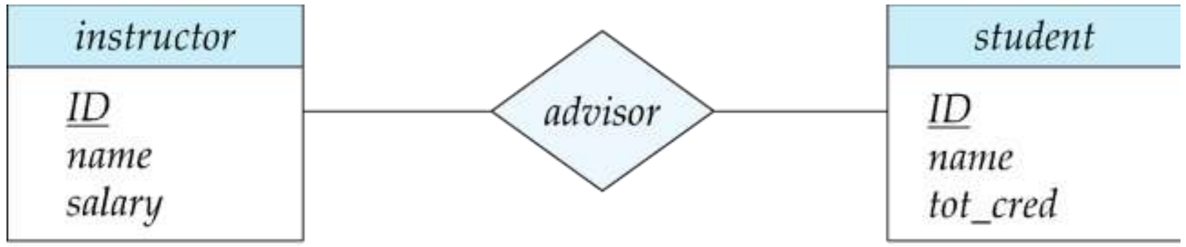
Representation of Entity Sets with Multivalued Attributes

- A multivalued attribute *M* of an entity *E* is represented by a separate schema *EM*
- Schema *EM* has attributes corresponding to the primary key of *E* and an attribute corresponding to multivalued attribute *M*
- Example: Multivalued attribute *phone_number* of *instructor* is represented by a schema:
  inst_phone= (<u>*ID*</u>, <u>*phone_number*</u>)
- Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*
  - For example, an *instructor* entity with primary key 22222 and phone numbers 456-7890 and 123-4567 maps to two tuples:
    (22222, 456-7890) and (22222, 123-4567)

Representing Relationship Sets

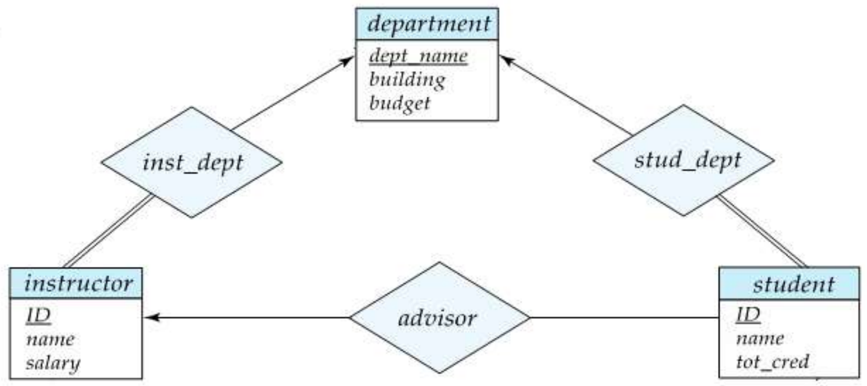
- A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.
- Example: schema for relationship set *advisor*

advisor = (*<u>s_id, i_id</u>*)



Redundancy of Schemas

- Many-to-one and one-to-many relationship sets that are total on the many-side can be represented by adding an extra attribute to the “many” side, containing the primary key of the “one” side
- Example: Instead of creating a schema for relationship set *inst_dept*, add an attribute *dept_name* to the schema arising from entity set *instructor*
- Example

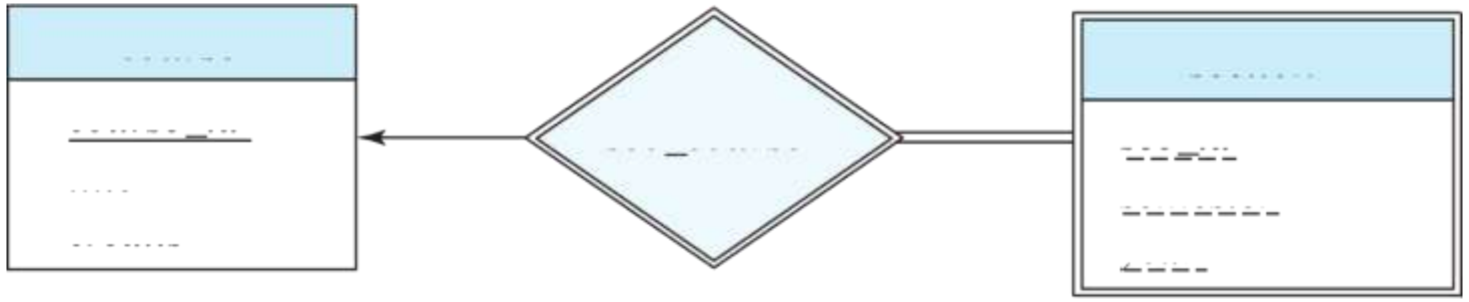


Redundancy of Schemas (Cont.)

- For one-to-one relationship sets, either side can be chosen to act as the “many” side
  - That is, an extra attribute can be added to either of the tables corresponding to the two entity sets
- If participation is *partial* on the “many” side, replacing a schema by an extra attribute in the schema corresponding to the “many” side could result in null values

Redundancy of Schemas (Cont.)

- The schema corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant.
- Example: The *section* schema already contains the attributes that would appear in the *sec_course* schema



Extended E-R Features Specialization

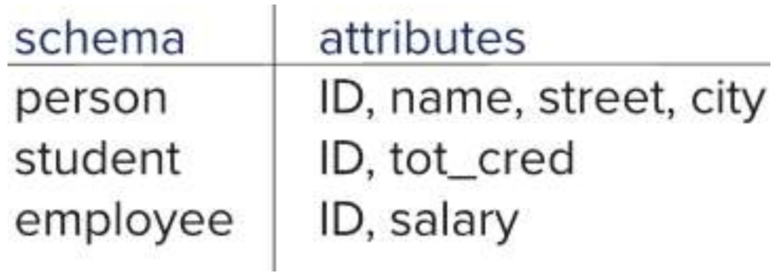
- Top-down design process; we designate sub-groupings within an entity set that are distinctive from other entities in the set.
- These sub-groupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labeled ISA (e.g., *instructor* “is a” *person*).
- **Attribute inheritance** – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked.

Specialization Example

- **Overlapping** – *employee* and *student*
- **Disjoint** – *instructor* and *secretary*
- Total and partial

Representing Specialization via Schemas

- Method 1:
  - Form a schema for the higher-level entity
  - Form a schema for each lower-level entity set, include primary key of higher-level entity set and local attributes

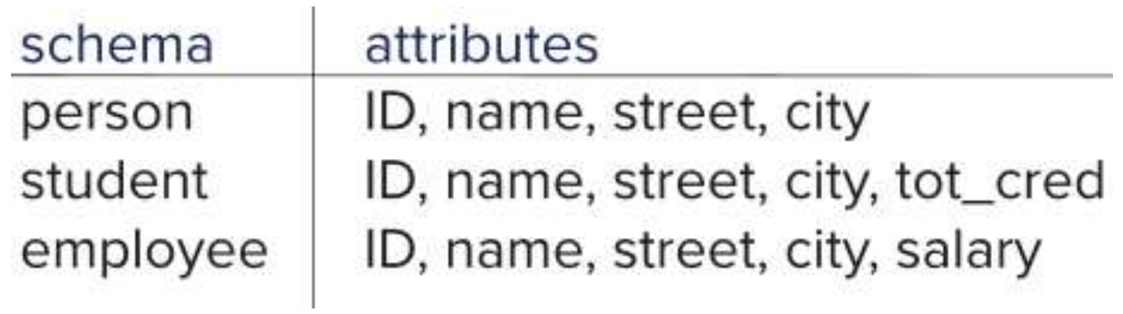


schema	attributes
person	ID, name, street, city
student	ID, tot_cred
employee	ID, salary

  - Drawback: getting information about, an *employee* requires accessing two relations, the one corresponding to the low-level schema and the one corresponding to the high-level schema

Representing Specialization as Schemas (Cont.)

- Method 2:
  - Form a schema for each entity set with all local and inherited attributes



schema	attributes
person	ID, name, street, city
student	ID, name, street, city, tot_cred
employee	ID, name, street, city, salary

  - Drawback: *name, street* and *city* may be stored redundantly for people who are both students and employees

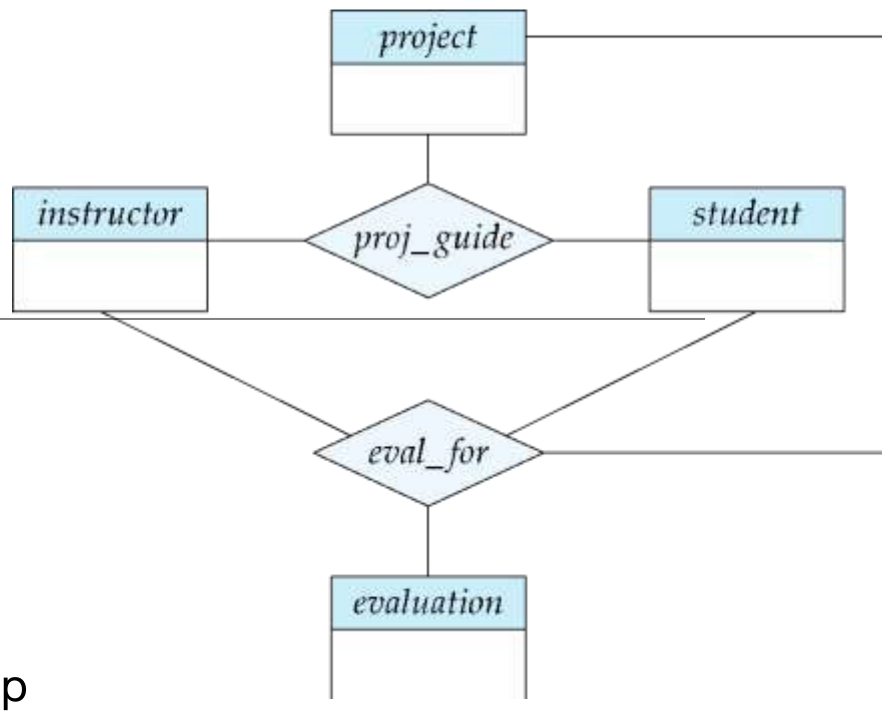
Generalization

- **A bottom-up design process** – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
- The terms specialization and generalization are used interchangeably.

Completeness constraint

- **Completeness constraint** -- specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within a generalization.
  - **total**: an entity must belong to one of the lower-level entity sets
  - **partial**: an entity need not belong to one of the lower-level entity sets
- Partial generalization is the default.
- We can specify total generalization in an ER diagram by adding the keyword **total** in the diagram and drawing a dashed line from the keyword to the corresponding hollow arrow-head to which it applies (for a total generalization), or to the set of hollow arrow-heads to which it applies (for an overlapping generalization).
- The *student* generalization is total: All student entities must be either graduate or undergraduate. Because the higher-level entity set arrived at through generalization is generally composed of only those entities in the lower-level entity sets, the completeness constraint for a generalized higher-level entity set is usually total

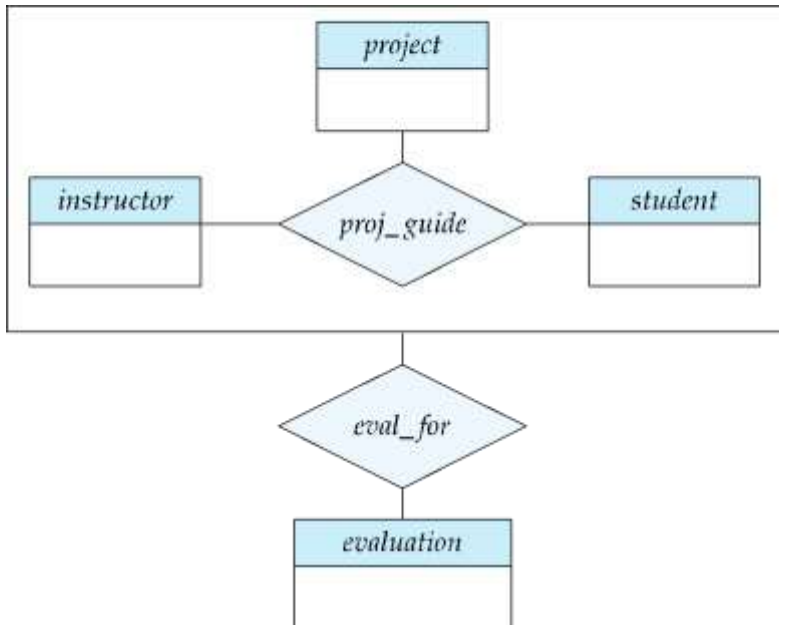
Aggregation



- Consider the ternary relationship *proj_guide*, which we saw earlier
- Suppose we want to record evaluations of a student by a guide on a project
- Relationship sets *eval_for* and *proj_guide* represent overlapping information
  - Every *eval_for* relationship corresponds to a *proj_guide* relationship
  - However, some *proj_guide* relationships may not correspond to any *eval_for* relationships
    - So we can't discard the *proj_guide* relationship
- Eliminate this redundancy via *aggregation*
  - Treat relationship as an abstract entity
  - Allows relationships between relationships
  - Abstraction of relationship into new entity

Aggregation (Cont.)

- Eliminate this redundancy via *aggregation* without introducing redundancy, the following diagram represents:
  - A student is guided by a particular instructor on a particular project
  - A student, instructor, project combination may have an associated evaluation

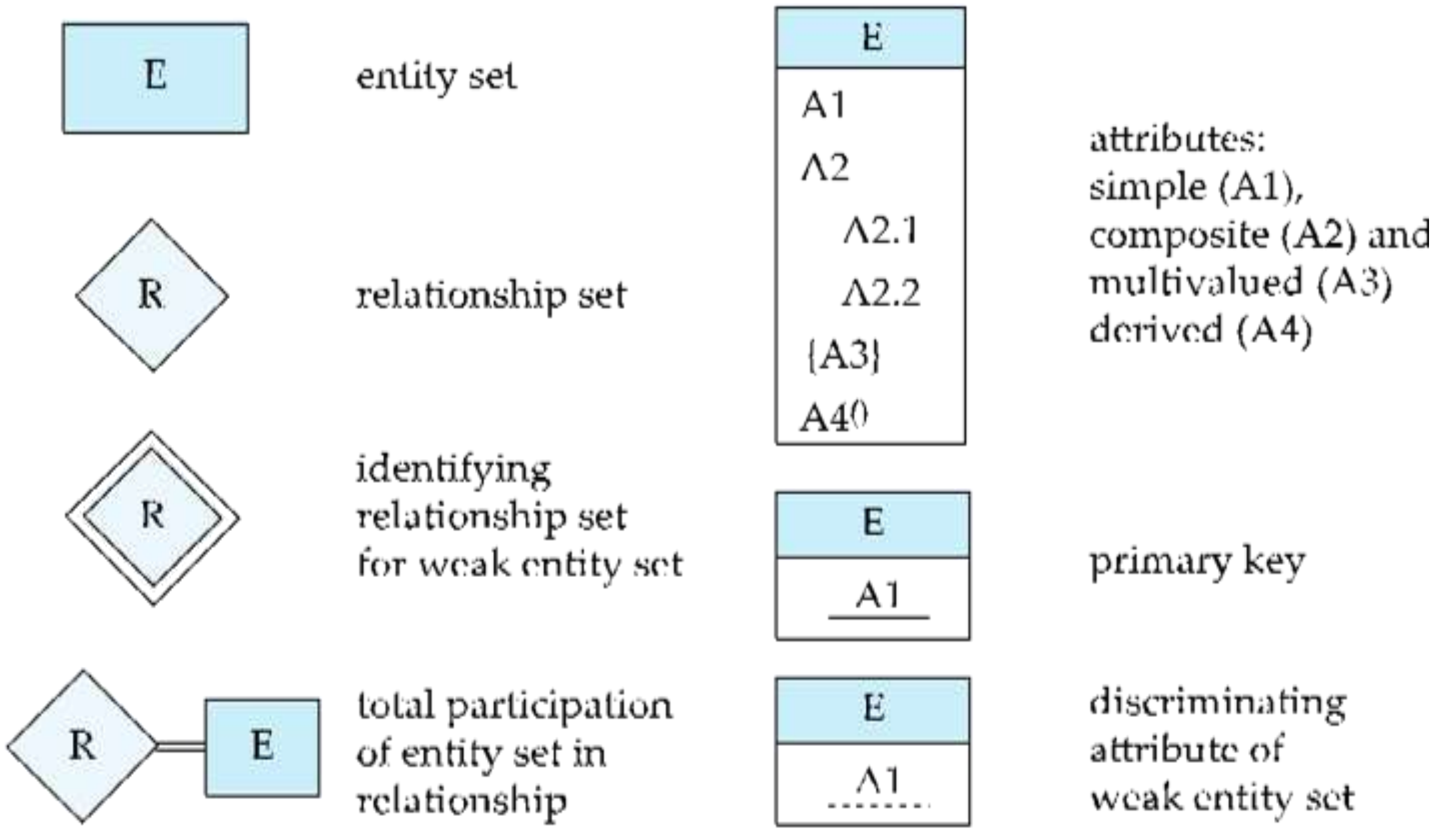


Reduction to Relational Schemas

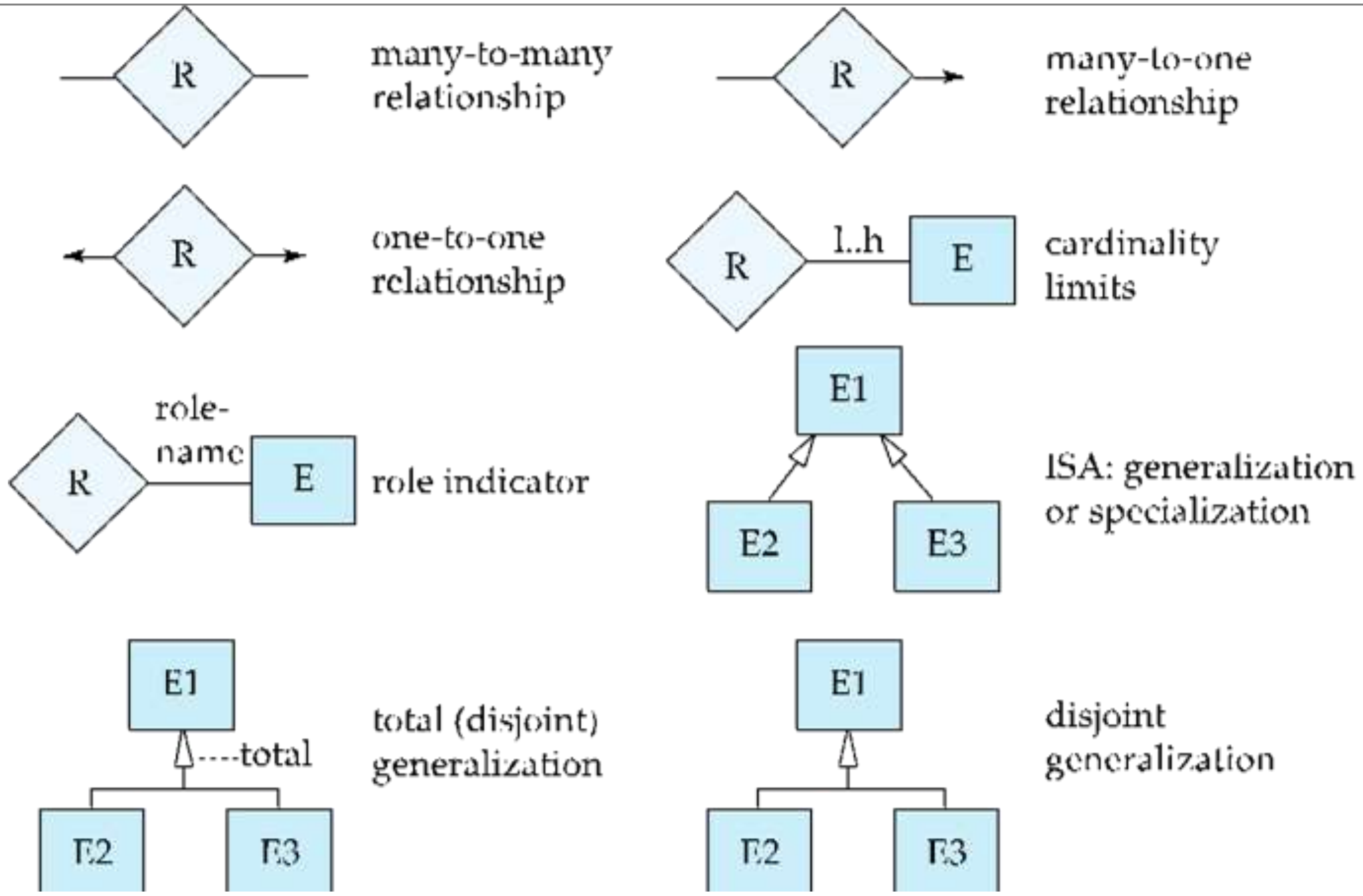
- To represent aggregation, create a schema containing
  - Primary key of the aggregated relationship,
  - The primary key of the associated entity set
  - Any descriptive attributes
- In our example:
  - The schema *eval_for* is:

    eval_for (*s_ID, project_id, i_ID, evaluation_id*)
  - The schema *proj_guide* is redundant.

Summary of Symbols Used in E-R Notation


E
entity set
R
relationship set
R
identifying relationship set for weak entity set
R
E
total participation of entity set in relationship
E
A1
A2
A2.1
A2.2
{A3}
A4()
attributes: simple (A1), composite (A2) and multivalued (A3) derived (A4)
E
A1
primary key
E
A1
discriminating attribute of weak entity set

Symbols Used in E-R Notation (Cont.)



THANK
YOU