JAWAHARLAL COLLEGE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, Affiliated to APJ Abdul Kalam Technological University, Kerala)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(NBA Accredited)

COURSE MATERIAL

CST 204 DATABASE MANAGEMENT SYSTEMS

VISION OF THE INSTITUTION

Emerge as a centre of excellence for professional education to produce high quality engineers and entrepreneurs for the development of the region and the Nation

.

MISSION OF THE INSTITUTION

- To become an ultimate destination for acquiring latest and advanced knowledge in the multidisciplinary domains.

- To provide high quality education in engineering and technology through innovative teaching-learning practices, research and consultancy, embedded with professional ethics.
- To promote intellectual curiosity and thirst for acquiring knowledge through outcome based education.
- To have partnership with industry and reputed institutions to enhance the employability skills of the students and pedagogical pursuits.
- To leverage technologies to solve the real life societal problems through community services.

ABOUT THE DEPARTMENT

- Established in: 2008
- Courses offered: B.Tech in Computer Science and Engineering
- Affiliated to the A P J Abdul Kalam Technological University.

DEPARTMENT VISION

To produce competent professionals with research and innovative skills, by providing them with the most conducive environment for quality academic and research oriented undergraduate education along with moral values committed to build a vibrant nation.

DEPARTMENT MISSION

- Provide a learning environment to develop creativity and problem solving skills in a professional manner.
- Expose to latest technologies and tools used in the field of computer science.
- Provide a platform to explore the industries to understand the work culture and expectation of an organization.
- Enhance Industry Institute Interaction program to develop the entrepreneurship skills.
- Develop research interest among students which will impart a better life for the society and the nation.

PROGRAMME EDUCATIONAL OBJECTIVES

Graduates will be able to

- Provide high-quality knowledge in computer science and engineering required for a computer professional to identify and solve problems in various application domains.

- Persist with the ability in innovative ideas in computer support systems and transmit the knowledge and skills for research and advanced learning.
- Manifest the motivational capabilities, and turn on a social and economic commitment to community services.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

COURSE OUTCOMES

SUBJECT CODE: CS206	
COURSE OUTCOMES	
C211.1	Summarize and exemplify fundamental nature and characteristics of database systems (Cognitive Knowledge Level: Understand)
C211.2	Model real word scenarios given as informal descriptions, using Entity Relationship diagrams. (Cognitive Knowledge Level: Apply)
C211.3	Model and design solutions for efficiently representing and querying data using relational model (Cognitive Knowledge Level: Analyze)
C211.4	Demonstrate the features of indexing and hashing in database applications (Cognitive Knowledge Level: Apply)
C211.5	Discuss and compare the aspects of Concurrency Control and Recovery in Database systems (Cognitive Knowledge Level: Apply)
C211.6	Explain various types of NoSQL databases (Cognitive Knowledge Level: Understand)

PROGRAM SPECIFIC OUTCOMES (PSO)

The students will be able to

- Use fundamental knowledge of mathematics to solve problems using suitable analysis methods, data structure and algorithms.
- Interpret the basic concepts and methods of computer systems and technical specifications to provide accurate solutions.
- Apply theoretical and practical proficiency with a wide area of programming knowledge, design new ideas and innovations towards research.

CO PO MAPPING

Note: H-Highly correlated=3, M-Medium correlated=2,L-Less correlated=1

CO'S	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
C211.1	1	2	2	-	-	-	-	-	-	-	1	-

C211..2	2	2	3	-	-	-	-	-	-	-	-	-
C211..3	2	2	2	-	-	-	-	-	-	-	1	-
C211.4	2	1	2	-	-	-	-	-	-	-	1	-
C211.5	2	2	2	-	-	-	-	-	-	-	1	-
C211..6	2	2	1	-	-	-	-	-	-	-	-	-
C211	**2**	**2**	**2.17**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**

CO PSO MAPPING

CO'S	PSO1	PSO2	PSO3
C211.1	1	2	-
C211.2	1	2	-
C211.3	1	2	-
C211.4	1	-	-
C211.5	-	2	-
C211.6	-	2	-
C211	**1**	**2**	**0**

CONTENT BEYOND SYLLABUS

S:NO	TOPIC
1	Functions, Procedures and HLL interfaces
2	Cost-based query optimization

Reference Materials

Module 1: Introduction & Entity Relationship (ER) Model

Concept & Overview of Database Management Systems (DBMS) - Characteristics of Database system, Database Users, structured, semi-structured and unstructured data. Data Models andSchema - Three Schema architecture. Database Languages, Database architectures and classification.

ER model - Basic concepts, entity set & attributes, notations, Relationships and constraints, cardinality, participation, notations, weak entities, relationships of degree 3.

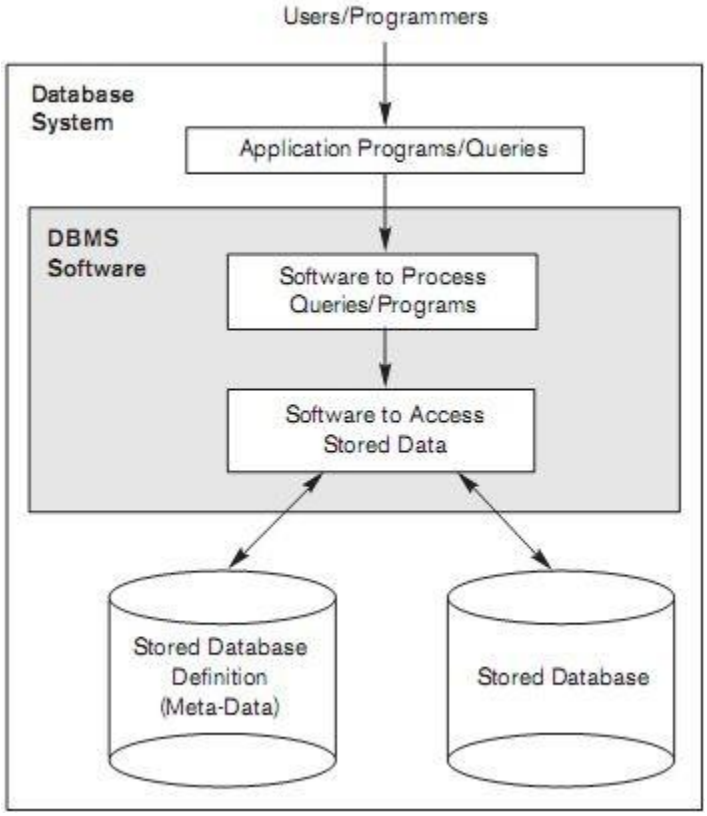
- A database is a collection of related data. Data means, known facts that can be recorded and that have implicit meaning.
- A database management system (DBMS) is a collection of programs that enables users to create and maintain a database. The DBMS is a general-purpose software system that facilitates the processes of

  defining, constructing, manipulating, and sharing databases among various users and applications.
- Defining a database involves specifying the data types, structures, and

  constraints of the data to be stored in the data-base.
- Constructing the database is the process of storing the data on some storage medium that is con-trolled by the DBMS.
- Manipulating a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data.
- Sharing a database allows multiple users and programs to access the

  database simultaneously.
- An application program accesses the database by sending queries or requests for data to the DBMS. A query typically causes some data to be retrieved; a transaction may cause some data to be read and some data to be written into thedatabase.
- Other important functions provided by the DBMS include protecting the database and maintaining it over a long period of time.
- Protection includes system protection against hardware or software malfunction (or crashes) and security protection against unauthorized or malicious access.
- A typical large database may have a life cycle of many years, so the DBMS must be able to maintain the database system by allowing the system to evolve as requirements change over time.

Database System Environment


Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

EXAMLE STUDENT DB

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Characteristics of the Database Approach

- In traditional file processing, each user defines and implements the files needed for a specific software application as part of programming the application.
- Redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date data.
- In the database approach, a single repository maintains data that is defined once and then accessed by various users
- The main characteristics of the database approach versus the file processing approach are the following:

1. Self-describing nature of a database system
2. Insulation between programs and data, and data abstraction
3. Support of multiple views of the data
4. Sharing of data and multiuser transaction processing

1. Self-Describing Nature of a Database System

- A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints.
- This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called meta-data and it describes

the structure of the primary database.

2. Insulation between Programs and Data, and Data Abstraction

- ➢ In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require *changing all programs* that access that file.
- ➢ DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. This property is called program-data independence.
- ➢ An operation (also called a function or method) is specified in two parts.
  - Interface
    - ✓ The interface (or signature) of an operation includes the operation name and the data types of its arguments (or parameters).
  - Implementation
    - ✓ The implementation (or method) of the operation is specified separately and can be changed without affecting the interface.
- ➢ User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This is termed as program-operation independence.
- ➢ The characteristic that allows program-data independence and program operation independence is called data abstraction.
- ➢ A data model is a type of data abstraction that is used to provide conceptual representation. A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented.

- The data model hides storage and implementation details that are not of interest to most database users.
-

3. Support of Multiple Views of the Data

- A database has many users, each user may require a different perspective or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored.
- A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.

4. Sharing of Data and Multiuser Transaction Processing

- A multiuser DBMS, must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database.
- The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.
- A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly and efficiently.
- A transaction is an executing program or process that includes one or more database accesses, such as reading or updating of database records.
- Each transaction is supposed to execute a logically correct database access if executed in its entirety without interference from other transactions.
- The DBMS must enforce several transaction properties.
    1. Isolation property
        - The isolation property ensures that each transaction

appears to execute in isolation from other transactions, even though hundreds of transactions may be executing concurrently.

2. Atomicity property
   - The atomicity property ensures that either all the database operations in a transaction are executed or none are.

ACTORS ON THE SCENE

- The people whose jobs involve the day-to-day use of a large database are called as the actors on the scene.

1. Database Administrators
2. Database Designers
3. End Users
4. System Analysts and Application Programmers (Software Engineers)

Database Administrators

- In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the database administrator (DBA).
- The DBA is responsible for authorizing access to the database, coordinating and monitoring its use,and acquiring software and hardware resources as needed. The DBA is accountable for problems such as security breaches and poor system response time.

Database Designers

- Database designers are responsible for identifying the data to be stored in the data-base and for choosing appropriate structures to represent

and store this data.

- It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements.
- Database designers typically interact with each potential group of users and develop views of the database that meet the data and processing requirements of these groups.
- Each view is then analyzed and integrated with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups.

End Users

- End users are the people whose jobs require access to the database for querying, updating, and generating reports.
- There are several categories of end users:

1. Casual end users
   - Casual end users occasionally access the database, but they may need different information each time.
   - They use a sophisticated database query language to specify their requests and are typically middle or high level managers or other occasional browsers.

2. Naive or parametric end users
   - Naive or parametric end users make up a sizable portion of database endusers.
   - Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates

called canned transactions that have been carefully programmed and tested.

- The tasks that such users perform are varied:

3. Sophisticated end users
   - Sophisticated end users include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.

4. Standalone users
   - .Standalone users maintain personal databases by using ready made program packages that provide easy-to-use menu based or graphics based interfaces.
   - An example is the user of a tax package that stores a variety of personal financial data for tax purposes.

System Analysts and Application Programmers (Software Engineers)

- System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for standard canned transactions that meet these requirements.
- Application programmers implement these specifications as programs, then they test, debug, document, and maintain these canned transactions. Such analysts and programmers commonly referred to as

software developers or software engineers

WORKERS BEHIND THE SCENE

- The people who work to maintain the database system environment but who are not actively interested in the database contents as part of their daily job are called as theworkers behind the scene.

1. DBMS system designers and implementers
2. Tool developers
3. Operators and maintenance personnel (system administration personnel)

DBMS system designers and implementers

- DBMS system designers and implementers design and implement the DBMS modules and interfaces as a software package.
- A DBMS is a very complex software system that consists of many components, or modules, including modules for implementing the catalog, query language processing, interface processing, accessing and buffering data, controlling concurrency,and handling data recovery and security.

Tool developers

- Tool developers design and implement tools , the software packages that facilitate database modeling and design, database system design, and improved performance.
- Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data

generation.

Operators and maintenance personnel (system administration personnel)

- Operators and maintenance personnel (system administration personnel) are responsible for the actual running and maintenance of the hardware and software environment for the database system.

ADVANTAGES OF USING THE DBMS

1. Controlling Redundancy
   - Redundancy in storing the same data multiple times leads to several problems.
   a) Duplication of effort
   b) Storage space is wasted
   c) Files that represent the same data may become inconsisten.t
2. Restricting Unauthorized Access
   - A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and to specify account restrictions.
3. Providing Persistent Storage for Program Objects
   - Databases can be used to provide persistent storage for program objects and data structures.
4. Providing Storage Structures and Search Techniques for Efficient Query Processing
   - Database systems must provide capabilities for *efficiently executing queries andupdates.*
   - The database is typically stored on disk, the DBMS must provide specialized data structures and search techniques to speed up

disk search for the desired records. Auxiliary files called indexes are used for this purpose

5. Providing Backup and Recovery
   - ➢ A DBMS must provide facilities for recovering from hardware or software failures. The backup and recovery subsystem of the DBMS is responsible for recovery.
   - ➢ For example, if the computer system fails in the middle of a complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in

     before the transaction started executing.
6. Providing Multiple User Interfaces
   - ➢ Users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces.
   - ➢ These include query languages for casual users, programming language interfaces for application programmers, forms and command codes for parametric users, and menu-driven interfaces

     and natural language interfaces for standalone users.
   - ➢ Both forms-style interfaces and menu-driven interfaces are commonly known asgraphical userinterfaces (GUIs).
7. Representing Complex Relationships among Data
   - ➢ A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficiently.

8. Enforcing Integrity Constraints
   - ➢ Integrity constraints are use to ensure accuracy and consistency of data in DB.
   - ➢ A DBMS should provide capabilities for defining and enforcing these constraints. The simplest type of integrity constraint involves

specifying a data type for each data item.

9. Permitting Inferencing and Actions Using Rules
   - ➢ Some database systems provide capabilities for defining deduction rules for inferencing new information from the stored database facts. Such systems are calleddeductive database systems.

10. Additional Implications of Using the Database Approach

a) Potential for Enforcing Standards.

b) Reduced Application Development Time

c) Flexibility.

d) Availability of Up-to-Date Information

e) Economies of Scale.

DATABASE SYSTEM CONCEPTSAND ARCHITECTURE

- ➢ One fundamental characteristic of the database approach is that it provides some level of data abstraction. Data abstraction refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data.
- ➢ A data modelis a collection of concepts that can be used to describe the structure of a database provides the necessary means to achieve this abstraction.

Categories of Data Models

1. High-level or conceptual data models provide concepts that are close to the way many users perceive data.

2. Low-level or physical data models provide concepts that describe the details of how data is stored on the computer storage media.

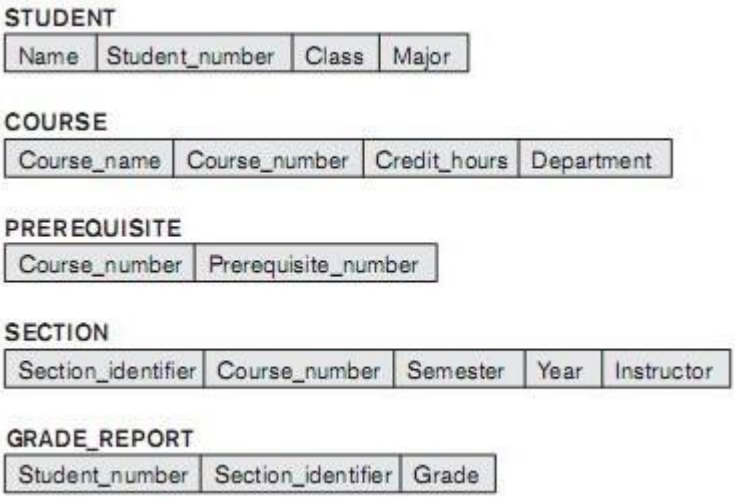
3. Representational(orimplementation)data models,which provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage.

- ➢ Conceptual data models use concepts such as entities, attributes, and relationships.
- ➢ An entity represents a real-world object or concept, such as an employee or a project from the mini world that is described in the database.
- ➢ An attribute represents some property of interest that further describes an entity, such as the employee's name or salary.
- ➢ A relationship among two or more entities represents an association among the entities, for example, a works-on relationship between an employee and a project.
- ➢ Representational or implementation data models are the models used most frequently in traditional commercial DBMSs. They are

1. relational data model,
2. network data model
3. hierarchical data model

Schemas,Instances,andDatabaseState

- ➢ The description of a database is called thedatabaseschema, which is specified during database design and is not expected to change frequently.
- ➢ A displayed schema is called aschema diagram.



- A schema diagram displays only *some aspects* of a schema, such as the names of record types and data items, and some types of constraints.
- The actual data in a database may change quite frequently. The data in the database at a particular moment in time is called a database state or snapshot. It is also called the *current* set of occurrences or instances in the database. Each schema construct has its own *current set* of instances.

The difference between database schema and database state

- When we define a new database the corresponding database state is the *empty state* with no data.
- We get the *initial state* of the database when the database is first populated or loaded with the initial data.
- At any point in time, the database has a *current state*.
- A valid state is, a state that satisfies the structure and constraints specified in the schema.
- The DBMS stores the descriptions of the schema constructs and

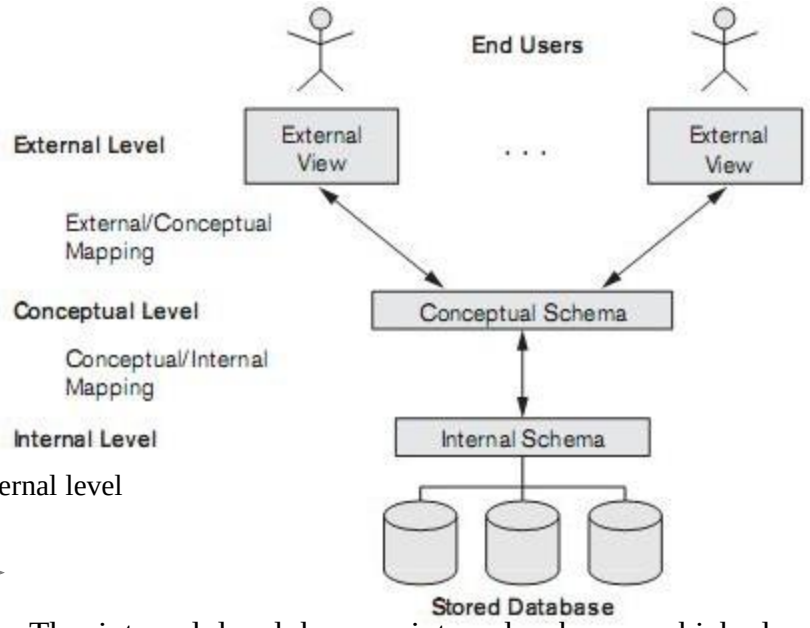
constraints also called themeta-datain the DBMS catalog.

- The schema is not supposed to change frequently ,but it is not uncommon that changes occasionally need to be applied to the schema as the application requirements change. It is called as schema evolution.

Three-Schema Architectureand Data Independence

The Three-Schema Architecture

- The goal of the three-schema architecture is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:



1. Internal level

-
- The internal level has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

2. Conceptual level

- The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints.

3. External or view level

- The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group.
- The processes of transforming requests and results between levels are called mappings.

DATA INDEPENDENCE

- The capacity to change the schema at one level of a database system without having to change the schema at the next higher level.
- Two types of data independence:

1. Logical data independence

- Logical data independence is the capacity to change the conceptual schema without having to change external schemas or application programs.

2. Physical data independence

Physical data independence is the capacity to change the internal schema

without having to change the conceptual schema.

- Data independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only the *mapping* between the two levels is changed.

Database Languages and Interfaces

DBMS Languages

Data definition language (DDL), is used by the DBA and by database designers to define conceptual and internal schemas schemas.

Storage definition language (SDL), is used to specify the internal schema.

View definition language (VDL), to specify user views and their mappings to the conceptual schema.

Data manipulation language (DML) is used for retrieval, insertion, deletion, and modification of the data.

- SQL relational database language which represents a combination of DDL, VDL, and DML, as well as statements for constraint specification, schema evolution, and other features.
- There are two main types of DMLs.

1. high-level or nonprocedural DML can be used on its own to specify complex database operations concisely.
2. low-level or procedural DML *must* be embedded in a general-purpose programming language. This type of DML typically retrieves individual records or objects from the database and processes each separately. Therefore, it needs to use programming language constructs, such as looping, to retrieve and process each record from a set of records. Low-level DMLs are also called record-at-a-timeDMLs

DBMS Interfaces

- User-friendly interfaces provided by a DBMS may include the following:

1. Menu-Based Interfaces for Web Clients or Browsing.
2. Forms-Based Interfaces.
3. Graphical User Interfaces (GUI)
4. Natural Language Interfaces
5. Speech Input and Output
6. Interfaces for Parametric Users.

DATA MODELING USING THEENTITY-RELATIONSHIP (ER) MODEL

- Entity-Relationship (ER) model, which is a popular high-level conceptual data model, used for the conceptual design of database applications, and many database design tools employ its concepts.
- The diagrammatic notation associated with the ER model, known as ER diagrams.

Entity, Types, Entity Sets, Attributes,and Keys

- An entity may be an object with a physical existence for example, a particular person, car, house, or employee or it may be an object with a conceptual existence.
- Each entity has attributes ,the particular properties that describe it.

Composite versus Simple (Atomic) Attributes

- Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings.

- Attributes that are not divisible are calledsimple or atomic attributes.

Single-Valued versus Multivalued Attributes

- Attributes have a single value for a particular entity; such attributes are called single-valued.

  For example, Age is a single-valued attribute of a person.

- Attribute that can have different *numbers* of *values* for same attributes are called multivalued.

  Eg:College_degrees attribute Stored

versus Derived Attributes

- In some cases, two (or more) attribute values are related ,for example, the Age and Birthdate attributes of a person. For a particular person entity, the value of Age can be determined from the current (today's) date and the value of that person's birthdate. The Age attribute is hence called a derived attribute and is said to be derivable from the birthdate attribute, which is called astored attribute.

NULL Values

- In some cases, a particular entity may not have an applicable value for an attribute.

Complex Attributes

- Composite and multivalued attributes can be nested. }. Such attributes are called complex attributes.

Entity type

- An entity type defines a *collection* (or *set*) of entities that have the same

attributes. Each entity type in the database is described by its name and attributes.

Entity set

- The collection of all entities of a particular entity type in the data-base at any point in time is called an entity set, the entity set is usually referred to using the same name as the entity type.
- An entity type describes the schema or intension for a *set of entities* that share the same structure. The collection of entities of a particular entity type is grouped into an entity set, which is also called the

  extension of the entity type.

Key attributes and entity type

- An important constraint on the entities of an entity type is the key or uniqueness constrainton attributes.
- An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set. Such an attribute is called a key attribute, and its values can be used to identify each entity uniquely.

Composite key

- It must be *minimal*, that is, all component attributes must be included in the composite attribute to have the uniqueness property.

RELATIONSHIP TYPES, RELATIONSHIP SETS, ROLES, AND STRUCTURAL CONSTRAINTS

- A relationship type R among n entity types E_1, E_2, ..., E_n defines a set of associations or a relationship set among entities from these entity types.
- As for the case of entity types and entity sets, a relationship type and its corresponding relationship set are customarily referred to by the *same name, R*. Mathematically, the relationship set R is a set of relationship instances r_i,
- where each r_i associates n individual entities (e_1, e_2, ..., e_n), and each entity e_j in r_i is a member of entity setE_j, 1 f j f n.
- Hence, a relationship set is a mathematical relation on E_1, E_2, ..., E_n; alternatively, it can be defined as a subset of the Cartesian product of the entity sets $E_1 \times E_2 \times ... \times E_n$.
- Each of the entity types E_1, E_2, ..., E_n is said to participate in the relationship type R; similarly, each of the individual entities e_1, e_2, ..., e_n is said to participate in the relationship instance

  $r_i = (e_1, e_2, ..., e_n)$.

Degree of a Relationship Type

- The degree of a relationship type is the number of participating entity types.
- A relationship type of degree two is called binary, and one of degree three is called ternary.

Role name

- The role name signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means.
- The relationship types in which the *same* entity type participates more than once in a relationship type in *different roles* such relationship

types are called recursive relationships.

Constraints on Binary Relationship Types

- Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set.
- Two main types of binary relationship constraints: *cardinality ratio* and *participation*.

Cardinality Ratios for Binary Relationships

- The cardinality ratio for a binary relationship specifies the *maximum* number of relationship instances that an entity can participate in.
- The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1, and M:N.

Participation Constraints and Existence Dependencies

- The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type.
- This constraint specifies the *minimum* number of relationship instances that each entity can participate in, and is sometimes called the minimum cardinality constraint
- There are two types of participation constraints, they are total and partial

1. Total participation

- Total participation is also calledexistence dependency

2. Partial participation

- partial, meaning that *some* or *part of the set of* employee entities are

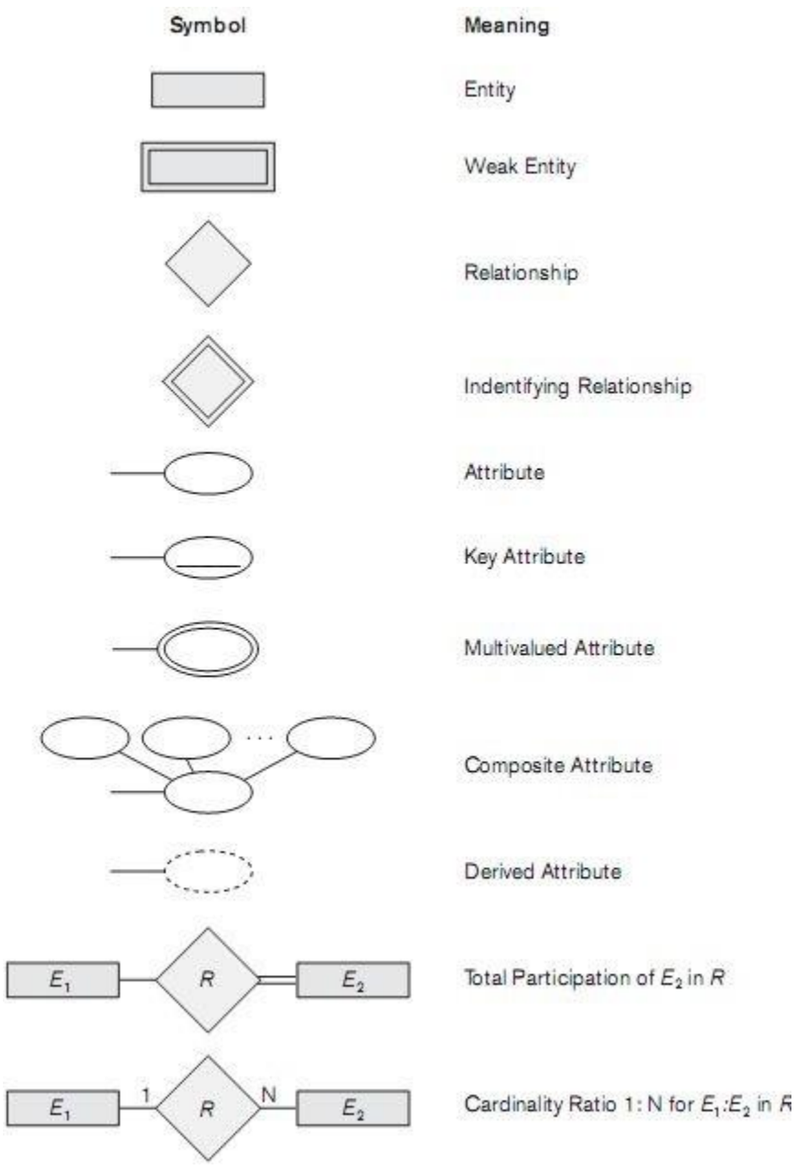
related to some department entity but not necessarily all.

- In ER diagrams, total participation (or existence dependency) is displayed as a *double line* connecting the participating entity type to the relationship,
- partial participation is represented by a*single line*
- The cardinality ratio and participation constraints, are together known as the structural constraintsof a relationship type.

WEAK ENTITY TYPES

- Entity types that do not have key attributes of their own are called weak entity types.
- Weak entity are identified by being related to specific entities from another entity type in combination with one of their attribute values.
- This other entity type is called theidentifying or owner entity type,
- The relationship type that relates a weak entity type to its owner is called the identifying relationshipof the weak entity type.
- A weak entity type always has a *total participation constraint* (existence dependency) with respect to its identifying relationship because a weak entity can not be identified without an owner entity.
- A weak entity type normally has a partial key, which is the attribute that can uniquely identify weak entities that are *related to the same owner entity*.

Design Choices for ER Conceptual Design



Module 2: Relational Model

Structure of Relational Databases - Integrity Constraints, Synthesizing ER diagram to relational schema Introduction to Relational Algebra - select, project, cartesian product operations, join - Equi-join, natural join. query examples, introduction to Structured Query Language (SQL), Data Definition Language (DDL), Table definitions and operations – CREATE, DROP, ALTER, INSERT, DELETE, UPDATE.

Relational Model Concepts

- The relational Model of Data is based on the concept of a Relation.

- A Relation is a mathematical concept based on the **ideas of sets**.
- The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations.

Relation

- RELATION: A table of values
  - A relation may be thought of as a **set of rows**.
  - A relation may alternately be thought of as a **set of columns**.
  - Each row represents a fact that corresponds to a real-world **entity** or **relationship**.
  - Each row has a value of an item or set of items that uniquely identifies that row in the table.
  - Sometimes row-ids or sequential numbers are assigned to identify the rows in the table.
  - Each column typically is called by its column name or column header or attribute name.

Schema of a Relation

- A **Relation** may be defined in multiple ways.
- The **Schema** of a Relation: *R* (A1, A2, An)

Relation schema *R* is defined over **attributes** A1, A2, An

For Example -

CUSTOMER (Cust-id, Cust-name, Address, Phone#)

Here, CUSTOMER is a relation defined over the four attributes Cust-id, Cust-name, Address, Phone#, each of which has **a domain** or **a set of valid values**. For example, the domain of Cust-id is 6 digit numbers.

Tuples

- A **tuple** is **an ordered set of values**
- Each value is derived from an appropriate domain.
- Each row in the CUSTOMER table may be referred to as a tuple in the table and would consist of four values.
- **<632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">**

is a tuple belonging to the CUSTOMER relation.

- **A relation** may be regarded as a ***set of tuples*** (rows).
- **Columns** in a table are also called **attributes** of the relation.
-

Domains

- A **domain** has a logical definition: e.g.,

“USA_phone_numbers” are the set of 10 digit phone numbers valid in the U.S.

- A domain may have a data-type or a format defined for it. The USA_phone_numbers may have a format: (ddd)-ddd-dddd where each d is a decimal digit. E.g., Dates have various formats such as monthname, date, year or yyyy-mm-dd, or dd mm,yyyy etc.
- An attribute designates the **role** played by the domain. E.g., the domain Date may be used to define attributes “Invoice-date” and “Payment-date”.

FORMAL DEFINITIONS

The relation is formed over the cartesian product of the sets; each set has values from a domain; that domain is used in a specific role which is conveyed by the attribute name.

For example, attribute Cust-name is defined over the domain of strings of 25 characters. The role these strings play in the CUSTOMER relation is that of the name of customers.

Formally,

Given R(A1, A2, , An)

r(R) ⊂ dom (A1) X dom (A2) X X dom(An)

R: schema of the relation

r of R: a specific "value" or population of R.

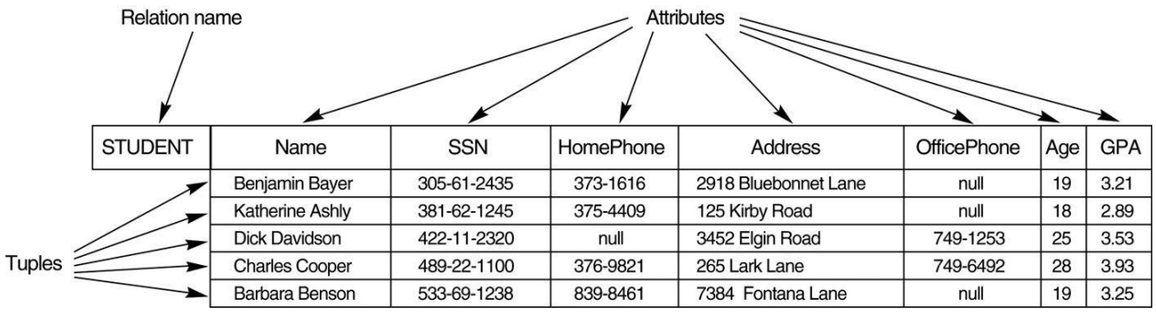
R is also called the intension of a relation r is also called the extension of a relation

Let S1 = {0,1}

Let S2 = {a,b,c} Let R ⊂ S1 X S2

Then for example: r(R) = {<0,a> , <0,b> , <1,c> }

is one possible “state” or “population” or “extension” r of the relation R, defined over domains S1 and S2. It has three tuples.



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

CHARACTERISTICS OF RELATIONS

- Ordering of tuples in a relation r(R): The tuples are *not* considered to be ordered, even though they appear to be in the tabular form.
- Ordering of attributes in a relation schema R (and of values within each tuple): We will consider the attributes in R(A1, A2, ..., An) and the values in t=<v1, v2, ..., vn> to be *ordered* .

(However, a more general *alternative definition* of relation does not require this

ordering).

- Values in a tuple: All values are considered *atomic* (indivisible). A special null value is used to represent values that are unknown or inapplicable to certain tuples.

- component values of a tuple t by t[Ai] = vi (the value of attribute Ai for tuple t).

Similarly, t[Au, Av, ..., Aw] refers to the subtuple of t containing the values of attributes

Au, Av, ..., Aw, respectively.

STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21

Relational Integrity Constraints

- **Constraints are *conditions* that must hold on *all* valid relation instances. There are three main types of constraints:**
  1. **Key constraints**
  2. **Entity integrity constraints**
  3. **Referential integrity constraints Key Constraints**
- Superkey of R: A set of attributes SK of R such that no two tuples *in any valid relation instance r(R)* will have the same value for SK. That is, for any distinct tuples t1 and t2 in r(R), t1[SK] ≠ t2[SK].
- Key of R: A "minimal" superkey; that is, a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey.

Example: The CAR relation schema:

CAR(State, Reg#, SerialNo, Make, Model, Year)

has two keys Key1 = {State, Reg#}, Key2 = {SerialNo}, which are also superkeys. {SerialNo, Make} is a superkey but *not* a key.

- If a relation has *several* candidate keys, one is chosen arbitrarily to be the primary key. The primary key attributes are *underlined*.

Figure 7.4 The CAR relation with two candidate keys: LicenseNumber and EngineSerialNumber.

CAR	LicenseNumber	EngineSerialNumber	Make	Model	Year
	Texas ABC-739	A69352	Ford	Mustang	96
	Florida TVP-347	B43696	Oldsmobile	Cutlass	99
	New York MPO-22	X83554	Oldsmobile	Delta	95
	California 432-TFY	C43742	Mercedes	190-D	93
	California RSK-629	Y82935	Toyota	Camry	98
	Texas RSK-629	U028365	Jaguar	XJS	98

Entity Integrity

- **Relational Database Schema**: A set S of relation schemas that belong to the same database. S is the *name* of the **database**.

S = {R1, R2, ..., Rn}

- **Entity Integrity**: **The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R).** This is because primary key values are used to *identify* the individual tuples.

t[PK] ≠ null for any tuple t in r(R) for any R

- Note: Other attributes of R may be similarly constrained to disallow null values, even though they are not members of the primary key.

Referential Integrity

- A constraint involving *two* relations (the previous constraints involve a *single* relation).
- Used to specify a *relationship* among tuples in two relations: the **referencing relation** and the **referenced relation**.
- Tuples in the *referencing relation* R1 have attributes FK (called **foreign key** attributes) that reference the primary key attributes PK of the *referenced relation* R2. A tuple t1 in R1 is said to **reference** a tuple t2 in R2 if t1[FK] = t2[PK].
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from R1.FK to R2.PK

Referential Integrity Constraint

Statement of the constraint

The value in the foreign key column (or columns) FK of the the **referencing relation** R1 can be either:

(1) a value of an existing primary key value of the corresponding primary key PK in the **referenced relation** R2,, or..

(2) a null.

In case (2), the FK in R1 should <u>not</u> be a part of its own primary key.

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

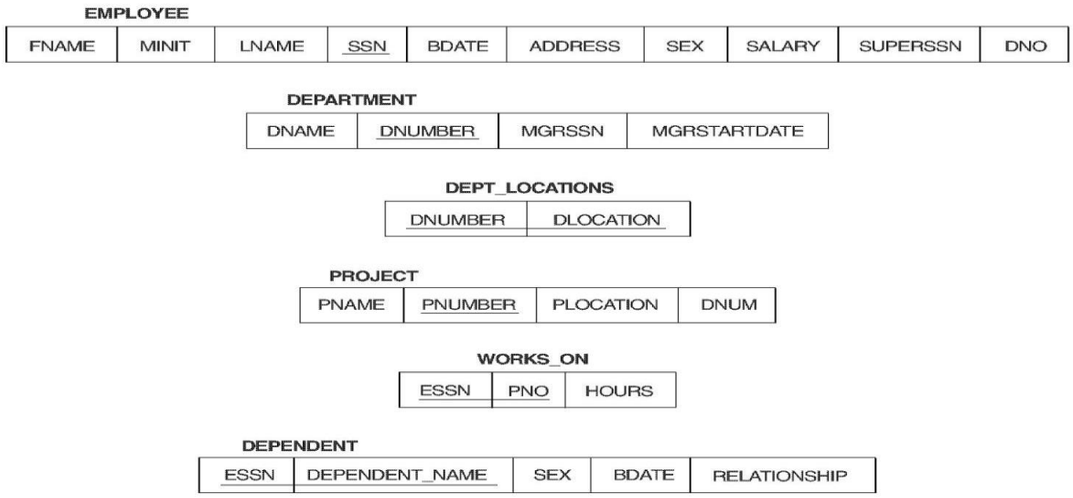


Figure 7.6 One possible relational database state corresponding to the COMPANY schema.

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
John		Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin		Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia		Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer		Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh		Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce		English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad		Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James		Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS

DNUMBER	DLOCATION
	Houston
	Stafford
	Bellaire
	Sugarland

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

WORKS_ON

ESSN	PNO	HOURS
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	null

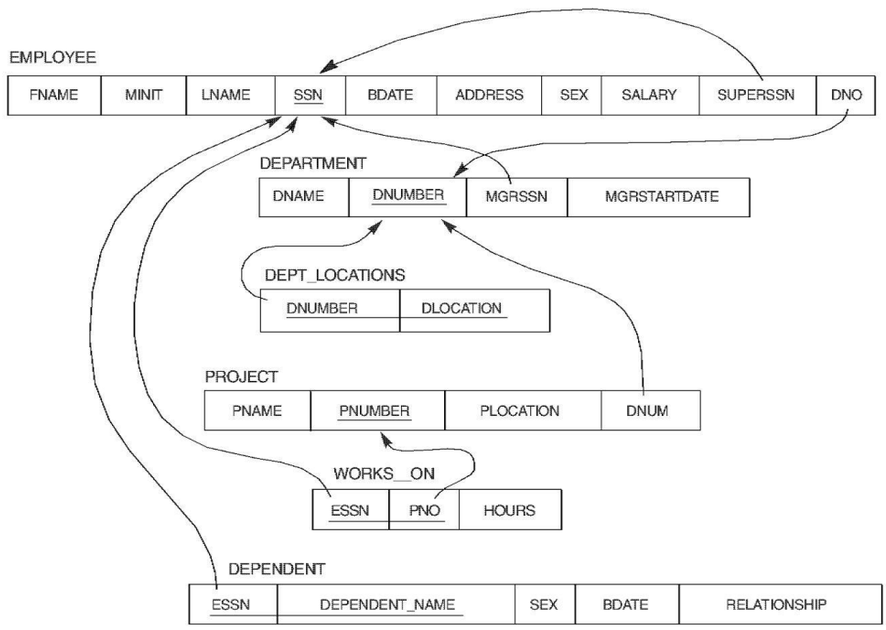
PROJECT

PNAME	PNUMBER	PLOCATION	DNUM
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
333445555	Alice	F	1986-04-05	DAUGHTER
333445555	Theodore	M	1983-10-25	SON
333445555	Joy	F	1958-05-03	SPOUSE
987654321	Abner	M	1942-02-28	SPOUSE
123456789	Michael	M	1988-01-04	SON
123456789	Alice	F	1988-12-30	DAUGHTER
123456789	Elizabeth	F	1967-05-05	SPOUSE

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



Update Operations on Relations

- INSERT a tuple.
- DELETE a tuple.
- MODIFY a tuple.

- Integrity constraints should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may *propagate*to cause other updates automatically. This may be necessary to maintain integrity constraints.
- In case of integrity violation, several actions can be taken: (e.g. Tables: **Employee** and **Work_On**) Cancel the operation that causes the violation (REJECT option)

Perform the operation but inform the user of the violation

Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option) Execute a user-specified error-correction routine

Q. Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:

STUDENT(SSN, Name, Major, Bdate) COURSE(Course#, Cname, Dept) ENROLL(SSN, Course#, Quarter, Grade)

BOOK_ADOPTION(Course#, Quarter, Book_ISBN) TEXT(Book_ISBN, Book_Title, Publisher, Author)

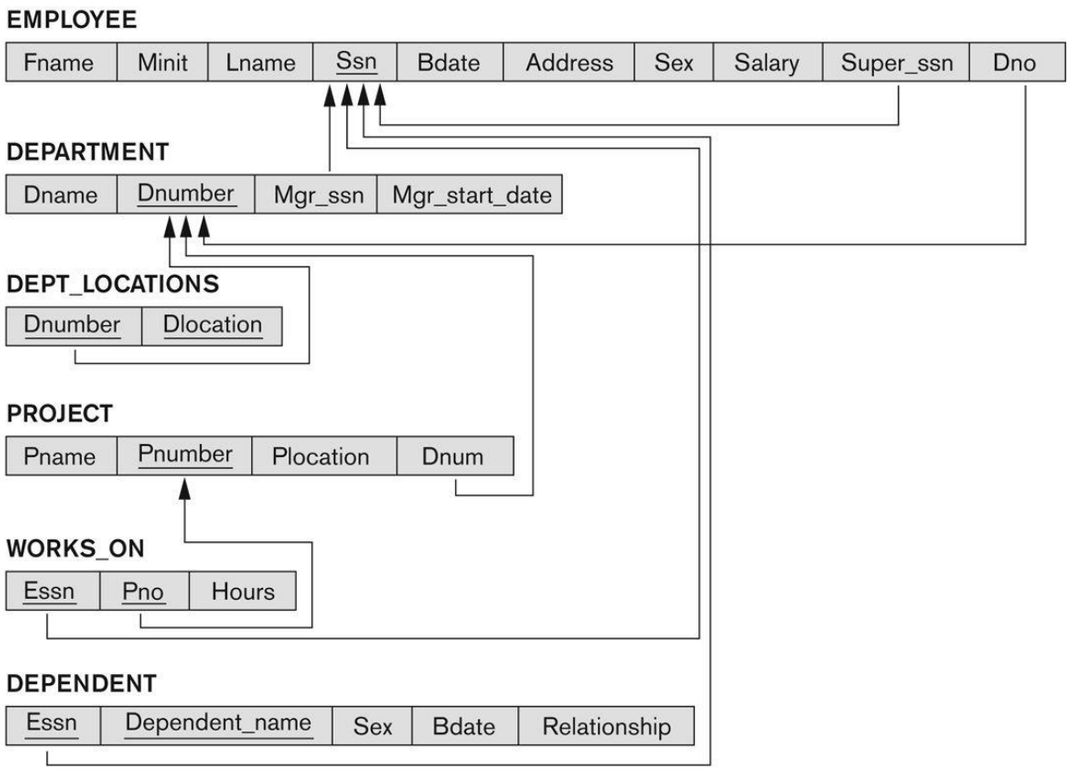
Draw a relational schema diagram specifying the foreign keys for this schema.

Relational Algebra

- Relational algebra is the basic set of operations for the relational model
- These operations enable a user to specify **basic retrieval requests** (or **queries**)
- The result of an operation is a *new relation*, which may have been formed from one or more *input* relations
- This property makes the algebra “closed” (all objects in relational algebra are relations)
- The **algebra operations** thus produce new relations
- These can be further manipulated using operations of the same algebra
- A sequence of relational algebra operations forms a **relational algebra expression**
- The result of a relational algebra expression is also a relation that represents the result of a database query (or retrieval request)
- Relational Algebra consists of several groups of operations

- Unary Relational Operations
  - SELECT (symbol: **s** (sigma))
  - PROJECT (symbol: **p** (pi))
  - RENAME (symbol: **r** (rho))
- Relational Algebra Operations From Set Theory
  - UNION (**È**), INTERSECTION (**Ç**), DIFFERENCE (or MINUS, **–**)
  - CARTESIAN PRODUCT (**x**)
- Binary Relational Operations
  - JOIN (several variations of JOIN exist)
  - DIVISION
- Additional Relational Operations
  - OUTER JOINS, OUTER UNION
  - AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



Unary Relational Operations: SELECT

- The SELECT operation (denoted by s (sigma)) is used to select a *subset* of the tuples from a relation based on a selection condition.
  - The selection condition acts as a filter
  - Keeps only those tuples that satisfy the qualifying condition
  - Tuples satisfying the condition are *selected* whereas the other tuples are discarded (*filtered out*)
- Examples:
  - Select the EMPLOYEE tuples whose department number is 4: s DNO = 4 (EMPLOYEE)
  - Select the employee tuples whose salary is greater than $30,000:

s SALARY > 30,000 (EMPLOYEE**)**

- In general, the *select* operation is denoted by s <selection condition>(R) where
  - the symbol s (sigma) is used to denote the *select* operator
  - the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
  - tuples that make the condition true are selected
    - appear in the result of the operation
  - tuples that make the condition false are filtered out

- discarded from the result of the operation

- **SELECT Operation Properties**

  - The SELECT operation s <selection condition>(R) produces a relation S that has the same schema (same attributes) as R
  - SELECT s is commutative:
  - s <condition1>(s < condition2> (R)) = s <condition2> (s < condition1> (R))
  - Because of commutativity property, a cascade (sequence) of SELECT

operations may be applied in any order:

  - s<cond1>(s<cond2> (s<cond3> (R)) = s<cond2> (s<cond3> (s<cond1> (R)))
  - A cascade of SELECT operations may be replaced by a single

selection with a conjunction of all the conditions:

  - s<cond1>(s< cond2> (s<cond3>(R)) = s <cond1> AND < cond2> AND < cond3>(R)))
  - The number of tuples in the result of a SELECT is less than (or equal

to) the number of tuples in the input relation R The following query results refer to this database state

Figure 5.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Unary Relational Operations: PROJECT

- PROJECT Operation is denoted by **p** (pi)
- This operation keeps certain *columns* (attributes) from a relation and discards the other columns.
    - PROJECT creates a vertical partitioning
        - The list of specified columns (attributes) is kept in each tuple
        - The other attributes in each tuple are discarded
- Example: To list each employee's first and last name and salary, the following is used: pLNAME, FNAME,SALARY(EMPLOYEE)
- The general form of the *project* operation is:

p<attribute list>(R)

    - p (pi) is the symbol used to represent the *project* operation
    - <attribute list> is the desired list of attributes from relation R.
- The project operation *removes any duplicate tuples*
    - This is because the result of the *project* operation must be a *set of tuples*
        - Mathematical sets *do not allow* duplicate elements.
- PROJECT Operation Properties
    - The number of tuples in the result of projection p<list>(R) is always less or equal to the

number of tuples in R

- If the list of attributes includes a *key* of R, then the number of tuples in the result of PROJECT is *equal* to the number of tuples in R

- PROJECT is *not* commutative
  - p <list1> (p <list2> (R)) = p <list1> (R) as long as <list2> contains the attributes in <list1>

Figure 6.1
Results of SELECT and PROJECT operations. (a) $\sigma_{(Dno=4\ \textbf{AND}\ Salary>25000)\ OR\ (Dno=5\ \textbf{AND}\ Salary>30000)}(EMPLOYEE)$. (b) $\pi_{Lname,\ Fname,\ Salary}(EMPLOYEE)$. (c) $\pi_{Sex,\ Salary}(EMPLOYEE)$.

(a)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5

(b)

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

(c)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

■

To retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a select and a project operation

- We can write a *single relational algebra expression* as follows:
  - **p**FNAME, LNAME, SALARY(**s** DNO=5(EMPLOYEE))
- OR We can explicitly show the *sequence of operations*, giving a name to each intermediate

relation:

- DEP5_EMPS ← **s** DNO=5(EMPLOYEE)
- RESULT ← **p** FNAME, LNAME, SALARY (DEP5_EMPS)

Unary Relational Operations: RENAME

- The RENAME operator is denoted by ρ (rho)
- In some cases, we may want to *rename* the attributes of a relation or the relation name or both
  - Useful when a query requires multiple operations
  - Necessary in some cases
  - The general RENAME operation ρ can be expressed by any of the following forms:

ρS (B1, B2, …, Bn)(R) changes both:

the relation name to S, *and*

the column (attribute) names to B1, B1, …..Bn

ρ (R) changes:

s

the *relation name* only to S

ρ(B1, B2, …, Bn)(R) changes:

the *column (attribute) names* only to B1, B1, …..Bn

- For convenience, we also use a *shorthand* for renaming attributes in an intermediate relation:
  - If we write:
    - RESULT ← **p** FNAME, LNAME, SALARY (DEP5_EMPS)
    - RESULT will have the *same attribute names* as DEP5_EMPS (same attributes as EMPLOYEE)
  - If we write:
    - RESULT (F, M, L, S, B, A, SX, SAL, SU, DNO)← **p** FNAME, LNAME, SALARY (DEP5_EMPS)
- The 10 attributes of DEP5_EMPS are *renamed* to F, M, L, S, B, A, SX, SAL, SU, DNO, respectivel

(a)

Fname	Lname	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

(b)
TEMP

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston,TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston,TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble,TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

R

First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

Figure 6.2
Results of a sequence of operations.
(a) $\pi_{Fname, Lname, Salary}(\sigma_{Dno=5}(EMPLOYEE))$.
(b) Using intermediate relations and renaming of attributes.

- **UNION Operation**
  - Binary operation, denoted by È
  - The result of R È S, is a relation that includes all tuples that are either in R or in S or in both R and S
  - Duplicate tuples are eliminated
  - The two operand relations R and S must be “type compatible” (or UNION compatible)
  - R and S must have same number of attributes
  - Each pair of corresponding attributes must be type compatible (have same or compatible domains)

- Example:
  - To retrieve the social security numbers of all employees who either *work in department 5* (RESULT1 below) or *directly supervise an employee who works in department 5* (RESULT2 below)
  - We can use the UNION operation as follows: DEP5_EMPS ← sDNO=5 (EMPLOYEE) RESULT1 ← p SSN(DEP5_EMPS) RESULT2(SSN) ← pSUPERSSN(DEP5_EMPS) RESULT ← RESULT1 È RESULT2
  - The union operation produces the tuples that are in either RESULT1 or RESULT2 or both

Figure 6.3
Result of the UNION operation RESULT ← RESULT1 ∪ RESULT2.

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

- Type Compatibility of operands is required for the binary set operation UNION È, (also for INTERSECTION Ç, and SET DIFFERENCE)
- R1(A1, A2, ..., An) and R2(B1, B2, ..., Bn) are type compatible if:
  - they have the same number of attributes, and

- the domains of corresponding attributes are type compatible (i.e. dom(Ai)=dom(Bi) for i=1, 2, ..., n).

- The resulting relation for R1ÈR2 (also for R1ÇR2, or R1–R2, see next slides) has the same attribute names as the *first* operand relation R1 (by convention)

- INTERSECTION is denoted by ∧
- The result of the operation R ∧ S, is a relation that includes all tuples that are in both R and S
    - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be “type compatible”

- **SET** DIFFERENCE (also called MINUS or EXCEPT) is denoted by –
- The result of R – S, is a relation that includes all tuples that are in R but not in S
    - The attribute names in the result will be the same as the attribute names in R

(a) **STUDENT**

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Figure 6.4
The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.

- The two operand relations R and S must be “type compatible”

- Notice that both union and intersection are *commutative* operations; that is
    - R È S = S È R, and R Ç S = S Ç R
- Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are *associative* operations; that is
    - R È (S È T) = (R È S) È T
    - (R Ç S) Ç T = R Ç (S Ç T)
- The minus operation is not commutative; that is, in general R – S ≠ S – R

CARTESIAN PRODUCT

- CARTESIAN (or CROSS) PRODUCT Operation
    - This operation is used to combine tuples from two relations in a combinatorial fashion.

- Denoted by R(A1, A2, . . ., An) x S(B1, B2, . . ., Bm)
- Result is a relation Q with degree n + m attributes:
    - Q(A1, A2, . . ., An, B1, B2, . . ., Bm), in that order.
- The resulting relation state has one tuple for each combination of tuples—one from R and one from S.
- Hence, if R has nR tuples (denoted as |R| = nR), and S has nS tuples, then R x S will have nR * nS tuples.
- The two operands do NOT have to be "type compatible”
- Generally, CROSS PRODUCT is not a meaningful operation Can become meaningful when followed by other operations Example (not meaningful):

FEMALE_EMPS ← **s** SEX=’F’(EMPLOYEE) EMPNAMES ← **p** FNAME, LNAME, SSN (FEMALE_EMPS) EMP_DEPENDENTS ← EMPNAMES x DEPENDENT

EMP_DEPENDENTS will contain every combination of EMPNAMES and DEPENDENT whether or not they are actually related

Figure 6.5
The CARTESIAN PRODUCT (CROSS PRODUCT) operation.

FEMALE_EMPS

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-07-19	3321Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

ACTUAL_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

Binary Relational Operations: JOIN (denoted by ⋈)

- The sequence of CARTESIAN PRODECT followed by SELECT is used quite commonly to identify and select related tuples from two relations
- A special operation, called JOIN combines this sequence into a single operation
- This operation is very important for any relational database with more than a single relation, because it allows us *combine related tuples* from various relations
- The general form of a join operation on two relations R(A1, A2, . . ., An) and S(B1, B2, . .

., Bm) is:

R ⋈ <join condition>S

- where R and S can be any relations that result from general *relational algebra*

expressions.

DEPT_MGR

Dname	Dnumber	Mgr_ssn	...	Fname	Minit	Lname	Ssn	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...

Figure 6.6
Result of the JOIN operation

EQUIJOIN

- The most common use of join involves join conditions with *equality comparisons* only
- Such a join, where the only comparison operator used is =, is called an EQUIJOIN.

- In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple.

NATURAL JOIN Operation

Another variation of JOIN called NATURAL JOIN — denoted by * — was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.

because one of each pair of attributes with identical values is superfluous

The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, *have the same name* in both relations

If this is not the case, a renaming operation is applied first.

- Example: To apply a natural join on the DNUMBER attributes of DEPARTMENT

and DEPT_LOCATIONS, it is sufficient to write:

- DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS
- Only attribute with the same name is DNUMBER
- An implicit join condition is created based on this attribute:
- DEPARTMENT.DNUMBER=DEPT_LOCATIONS.DNUMBER
- Another example: Q ← R(A,B,C,D) * S(C,D,E)
- The implicit join condition includes *each pair* of attributes with the same name, "AND"ed together:
  - R.C=S.C AND R.D.S.D
- Result keeps only one attribute of each such pair:
  - Q(A,B,C,D,E)

(a)

PROJ_DEPT

Pname	Pnumber	Plocation	Dnum	Dname	Mgr_ssn	Mgr_start_date
ProductX	1	Bellaire	5	Research	333445555	1988-05-22
ProductY	2	Sugarland	5	Research	333445555	1988-05-22
ProductZ	3	Houston	5	Research	333445555	1988-05-22
Computerization	10	Stafford	4	Administration	987654321	1995-01-01
Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

(b)

DEPT_LOCS

Dname	Dnumber	Mgr_ssn	Mgr_start_date	Location
Headquarters	1	888665555	1981-06-19	Houston
Administration	4	987654321	1995-01-01	Stafford
Research	5	333445555	1988-05-22	Bellaire
Research	5	333445555	1988-05-22	Sugarland
Research	5	333445555	1988-05-22	Houston

Figure 6.7
Results of two NATURAL JOIN operations.
(a) PROJ_DEPT ← PROJECT * DEPT.
(b) DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS.

- **DIVISION Operation**

The division operation is applied to two relations

R(Z) ¸ S(X), where X subset Z. Let Y = Z - X (and hence Z = X È Y); that is, let Y be the set of attributes of R that are not attributes of S.

The result of DIVISION is a relation T(Y) that includes a tuple t if tuples tR appear in R with tR

[Y] = t, and with

tR [X] = ts *for every tuple* ts in S.

For a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in

combination with *every* tuple in S.

(a)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(b)

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

Figure 6.8
The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) $T \leftarrow R \div S$.

Table 6.1
Operations of Relational Algebra

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{<\text{selection condition}>}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{<\text{attribute list}>}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{<\text{join condition}>} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{<\text{join condition}>} R_2$, OR $R_1 \bowtie_{(<\text{join attributes 1}>),\ (<\text{join attributes 2}>)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{<\text{join condition}>} R_2$, OR $R_1 *_{(<\text{join attributes 1}>),\ (<\text{join attributes 2}>)} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Relational Calculus

- A **relational calculus** expression creates a new relation, which is specified in terms of variables that range over rows of the stored database relations (in **tuple calculus**) or over columns of the stored relations (in **domain calculus**).
- In a calculus expression, there is *no order of operations* to specify how to retrieve the query result—a calculus expression specifies only what information the result should contain.
  - This is the main distinguishing feature between relational algebra and relational calculus.
  - Relational calculus is considered to be a **nonprocedural** language.
  - This differs from relational algebra, where we must write a *sequence of operations* to specify a retrieval request; hence relational algebra can be considered as a **procedural** way of stating a query.

Tuple Relational Calculus

- The tuple relational calculus is based on specifying a number of tuple variables.
- Each tuple variable usually ranges over a particular database relation, meaning that the variable may take as its value any individual tuple from that relation.
- A simple tuple relational calculus query is of the form
  - {t | COND(t)}

o where t is a tuple variable and COND (t) is a conditional expression involving t.

o The result of such a query is the set of all tuples t that satisfy COND (t).

- Example: To find the first and last names of all employees whose salary is above

$50,000, we can write the following tuple calculus expression:

  - {t.FNAME, t.LNAME | EMPLOYEE(t) AND t.SALARY>50000}
  - The condition EMPLOYEE(t) specifies that the range relation of tuple variable t is EMPLOYEE.
  - The first and last name (PROJECTION pFNAME, LNAME) of each EMPLOYEE tuple t that satisfies the condition t.SALARY>50000 (SELECTION s SALARY >50000) will be retrieved.

The Existential and Universal Quantifiers

- Two special symbols called quantifiers can appear in formulas; these are the universal quantifier (") and the existential quantifier ($).
- Informally, a tuple variable t is bound if it is quantified, meaning that it appears in an (" t) or ($ t) clause; otherwise, it is free.
- If F is a formula, then so are ($ t)(F) and (" t)(F), where t is a tuple variable.
    - The formula ($ t)(F) is true if the formula F evaluates to true for some (at least one) tuple assigned to free occurrences of t in F; otherwise ($ t)(F) is false.
    - The formula (" t)(F) is true if the formula F evaluates to true for every tuple (in the universe) assigned to free occurrences of t in F; otherwise (" t)(F) is false.
- " is called the universal or “for all” quantifier because every tuple in “the universe of” tuples must make F true to make the quantified formula true.
- $ is called the existential or “there exists” quantifier because any tuple that exists in “the universe of” tuples may make F true to make the quantified formula true.

- The language **SQL** is based on tuple calculus. It uses the basic block structure to express the queries in tuple calculus:
    - SELECT <list of attributes>
    - FROM <list of relations>
    - WHERE <conditions>
- SELECT clause mentions the attributes being projected, the FROM clause mentions the relations needed in the query, and the WHERE clause mentions the selection as well as the join conditions.

SQL syntax is expanded further to accommodate other operations

Another language which is based on tuple calculus is **QUEL** which actually uses the range variables as in tuple calculus. Its syntax includes:

RANGE OF <variable name> IS <relation name> Then it uses

RETRIEVE <list of attributes from range variables> WHERE <conditions>

This language was proposed in the relational DBMS INGRES.

The Domain Relational Calculus

- Another variation of relational calculus called the domain relational calculus, or simply, domain calculus is equivalent to tuple calculus and to relational algebra.
- The language called QBE (Query-By-Example) that is related to domain calculus was developed almost concurrently to SQL at IBM Research, Yorktown Heights, New York.
    - Domain calculus was thought of as a way to explain what QBE does.
- Domain calculus differs from tuple calculus in the type of variables used in formulas:
    - Rather than having variables range over tuples, the variables range over single values from domains of attributes.
- To form a relation of degree n for a query result, we must have n of these domain variables— one for each attribute.
- An expression of the domain calculus is of the form

{ x1, x2, . . ., xn |

COND(x1, x2, . . ., xn, xn+1, xn+2, . . ., xn+m)}

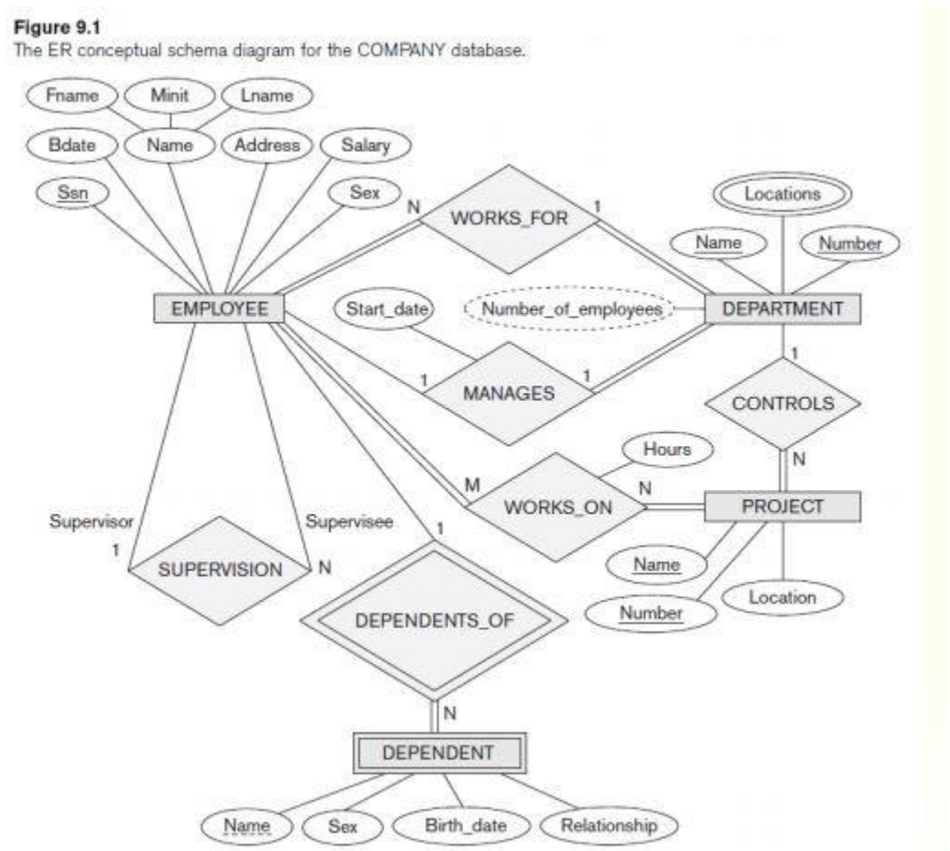
    - where x1, x2, . . ., xn, xn+1, xn+2, . . ., xn+m are domain variables that range over domains (of attributes)

- and COND is a condition or formula of the domain relational calculus.

QBE: A Query Language Based on Domain Calculus

- This language is based on the idea of giving an example of a query using example elements.
- An example element stands for a domain variable and is specified as an example value preceded by the underscore character.
- P. (called P dot) operator (for “print”) is placed in those columns which are requested for the result of the query.
- A user may initially start giving actual values as examples, but later can get used to providing a minimum number of variables as example elements.

Relational Database Design Using ER-to-Relational Mapping



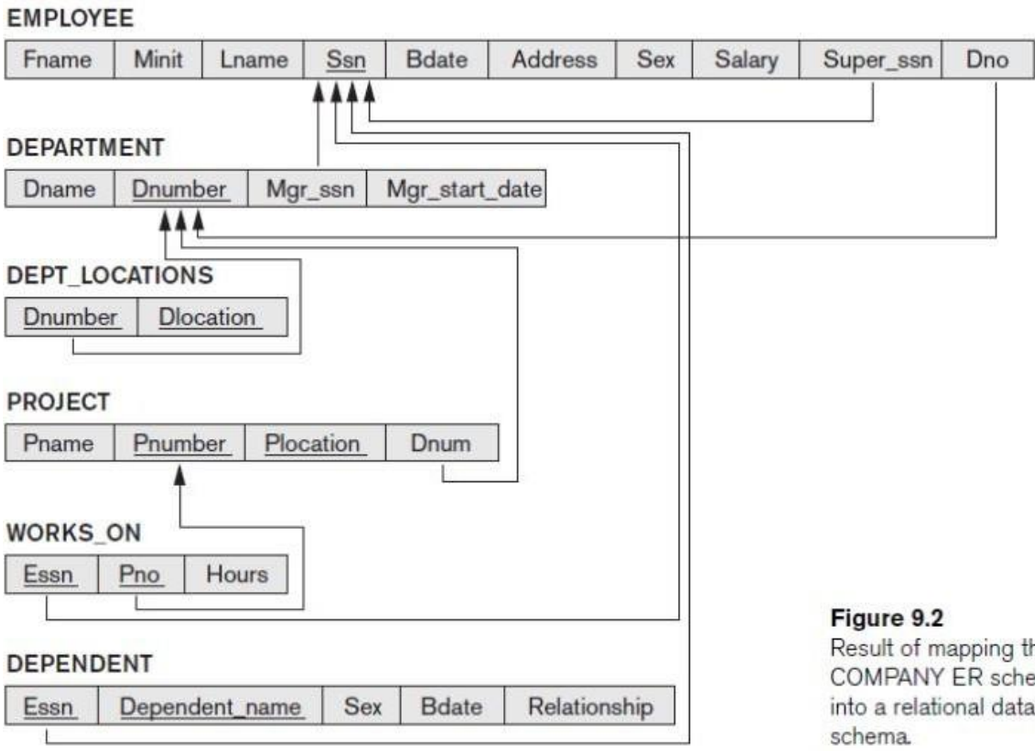


Figure 9.2
Result of mapping the COMPANY ER schema into a relational database schema.

ER-to-Relational Mapping Algorithm

COMPANY database example

Assume that the mapping will create tables with simple single-valued attributes

Step 1: Mapping of Regular Entity Types

For each regular entity type, create a relation R that includes all the simple attributes of E Called entity relations . Each tuple represents an entity instance.

Step 2: Mapping of Weak Entity Types

For each weak entity type, create a relation R and include all simple attributes of the entity type as attributes of R . Include primary key attribute of owner as foreign key attributes of R

Step 3: Mapping of Binary 1:1 Relationship Types

For each binary 1:1 relationship type . Identify relations that correspond to entity types participating in R . Possible approaches: Foreign key approach. Merged relationship approach.Crossreference or relationship relation approach

Step 4: Mapping of Binary 1:N Relationship Types.

For each regular binary 1:N relationship type. Identify relation that represents participating entity type at N-side of relationship type . Include primary key of other entity type as foreign key in S . Include simple attributes of 1:N relationship type as attributes of S

Alternative approach • Use the relationship relation (cross-reference) option as in the third option for binary 1:1 relationships

Step 5: Mapping of Binary M:N Relationship Types . For each binary M:N relationship type.Create a new relation S. Include primary key of participating entity types as foreign key attributes in S. Include any simple attributes of M:N relationship type

Step 6: Mapping of Multivalued Attributes

For each multivalued attribute ,Create a new relation .Primary key of R is the combination of A and K. If the multivalued attribute is composite, include its simple components

Step 7: Mapping of N-ary Relationship Types

For each n-ary relationship type R , Create a new relation S to represent R

- Include primary keys of participating entity types as foreign keys

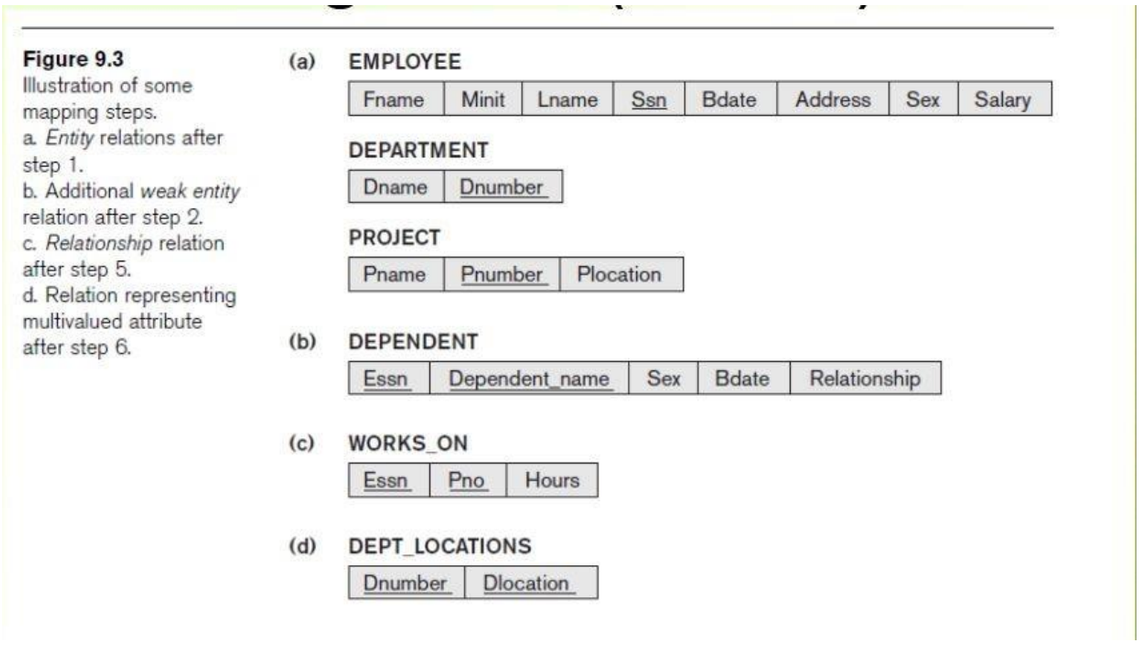


Figure 9.3
Illustration of some mapping steps.
a. *Entity* relations after step 1.
b. Additional *weak entity* relation after step 2.
c. *Relationship* relation after step 5.
d. Relation representing multivalued attribute after step 6.

- nclude any simple attributes as attributes

Table 9.1 Correspondence between ER and Relational Models

ER MODEL	RELATIONAL MODEL
Entity type	*Entity* relation
1:1 or 1:N relationship type	Foreign key (or *relationship* relation)
M:N relationship type	*Relationship* relation and *two* foreign keys
n-ary relationship type	*Relationship* relation and *n* foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

Basic SQL

♣ SQL language ♣

Considered one of the major reasons for the commercial success of relational databases SQL -Structured Query Language

Statements for data definitions, queries, and updates (both DDL and DML)

SQL Data Definition and Data Types Terminology:

Table, row, and column used for relational model terms relation, tuple, and attribute

CREATE statement - Main SQL command for data definition SQL schema - Identified by a schema name

Includes an authorization identifier and descriptors for each element

Schema elements include - Tables, constraints, views, domains, and other constructs. Each statement in SQL ends with a semicolon.

CREATE SCHEMA statement

e.g ; CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith';

Specify a new relation Provide name

Specify attributes and initial constraints. Can optionally specify schema:

CREATE TABLE COMPANY.EMPLOYEE ... or CREATE TABLE EMPLOYEE...

The CREATE TABLE Command in SQL

Base tables (base relations) Relation and its tuples are actually created and stored as a file by the DBMS Virtual relations. Created through the CREATE VIEW statement

Attribute Data Types and Domains in SQL Basic data types

- Numeric data types •
- Integer numbers: INTEGER, INT, and SMALLINT
- Floating-point (real) numbers: FLOAT or REAL, and DOUBLE PRECISION

- Character-string data types • Fixed length: CHAR(n), CHARACTER(n) • Varying length: VARCHAR(n), CHAR VARYING (n), CHARACTER VARYING(n)
- Bit-string data types • Fixed length: BIT(n) • Varying length: BIT VARYING(n)
- Boolean data type • Values of TRUE or FALSE or NULL
- DATE data type • Ten positions • Components are YEAR, MONTH, and DAY in the form YYYY-MM-DD

- The SELECT-FROM-WHERE Structure of SQL Queries SELECT <attributes>

FROM <table> WHERE<condition>

EXAMPLES

QUERY Retrieve the birthdate and address of the employee(s) whose name is 'John B. Smith'

SELECT BDATE, ADDRESS FROM EMPLOYEE WHERE FNAME='John' AND MINIT='B' AND LNAME='Smith';

QUERY Retrieve the name and address of all employees who work for the 'Research' department.

SELECT FNAME, LNAME, ADDRESSλ FROM EMPLOYEE, DEPARTMENT WHERE DNAME='Research' AND DNUMBER=DNO;

Unspecified WHERE-Clause and Use of Asterisk (*)

SELECT * FROM EMPLOYEE WHERE DNO=5;

SELECT * FROM EMPLOYEE, DEPARTMENT WHERE DNAME='Research' AND DNO=DNUMBER;

QUERY Select all combinations of EMPLOYEE SSN and DEPARTMENT DNAME in the database.

SELECT * FROM EMPLOYEE, DEPARTMENT;

Tables as Sets in SQL QUERY

Retrieve the salary of every employee and all distinct salary values.

SELECT ALL SALARY FROM EMPLOYEE; SELECT DISTINCT SALARYλ FROM EMPLOYEE;

Substring Comparisons, Arithmetic Operators, and Ordering

QUERY:Retrieve all employees whose address is in Houston, Texas.

SELECT FNAME, LNAME FROM EMPLOYEE WHERE ADDRESS LIKE

'%Houston,TX%';

QUERY Find all employees who were born during the 1950s.

SELECT FNAME, LNAMEλ FROM EMPLOYEE WHERE BDATE LIKE'_ _ 5 ';

QUERY Retrieve all employees in department 5 whose salary is between $30,000 and $40,000. SELECT *FROM EMPLOYEE WHERE (SALARY BETWEEN 30000 AND 40000) AND DNO = 5;

QUERY Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, first name.

SELECT DNAME, LNAME, FNAME, PNAMEλ FROM DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT WHERE DNUMBER=DNO AND SSN=ESSN AND PNO=PNUMBER ORDER BY DNAME, LNAME, FNAME;

Explicit Sets and NULLS in SQL

QUERY Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

SELECT DISTINCT ESSNλ FROM WORKS_ON WHERE PNO IN (1, 2, 3);

QUERY Retrieve the names of all employees who do not have supervisors. SELECT FNAME, LNAME FROM EMPLOYEE WHERE SUPERSSN IS NULL;

Renaming Attributes and Joined Tables

QUERY For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor. S

ELECT E.LNAME AS EMPLOYEE_NAME, S.LNAME AS SUPERVISOR_NAME FROM EMPLOYEE AS E, EMPLOYEE AS S WHERE E.SUPERSSN=S.SSN;

Aggregate Functions and Grouping

QUERY

Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

SELECT SUM (SALARY), MAX (SALARY), MIN (SALARY),λ AVG (SALARY) FROM EMPLOYEE;

QUERY

Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

SELECT SUM (SALARY), MAX (SALARY), MIN (SALARY), AVG (SALARY) FROM EMPLOYEE, DEPARTMENT WHERE DNO=DNUMBER AND DNAME='Research';

QUERY

Count the number of distinct salary values in the database. SELECT COUNT (DISTINCT SALARY) FROM EMPLOYEE;

SELECT LNAME, FNAME FROM EMPLOYEE WHERE (SELECT COUNT (*) FROM DEPENDENT WHERE SSN=ESSN) >= 2;

For each project on which more than two employees work, retrieve the project number, the

project name, and the number of employees who work on the project.

SELECT PNUMBER, PNAME, COUNT (*)λ FROM PROJECT, WORKS_ON WHERE PNUMBER=PNO GROUP BY PNUMBER, PNAME HAVING COUNT (*) > 2;

Insert, Delete, and Update Statements in SQL

INSERT COMMAND

INSERT INTO EMPLOYEE VALUES ('Richard', 'K', 'Marini', '653298653', '1962-12-30','98 Oak Forest,Katy,TX', 'M', 37000, '987654321', 4);

INSERT INTO EMPLOYEE (FNAME, LNAME, DNO) VALUES ('Robert', 'Hatcher', 5);

CREATE TABLE DEPTS_INFO (DEPT_NAME VARCHAR(15), NO_OF_EMPS INTEGER, TOTAL_SAL INTEGER); INSERT INTO DEPTS_INFO (DEPT_NAME,λ NO_OF_EMPS, TOTAL_SAL) SELECT DNAME, COUNT (*), SUM (SALARY) FROM (DEPARTMENT JOIN EMPLOYEE ON DNUMBER=DNO) GROUP BY DNAME;

DELETE COMMAND

DELETE FROM EMPLOYEE WHERE LNAME='Brown'; DELETE FROM EMPLOYEE WHERE SSN='123456789';

The UPDATE Command

UPDATE PROJECT SET PLOCATION = 'Bellaire', DNUM = 5 WHERE PNUMBER=10;

Module 3: SQL DML (Data Manipulation Language), Physical Data Organization

SQL DML (Data Manipulation Language) - SQL queries on single and multiple tables, Nested queries (correlated and non-correlated), Aggregation and grouping, Views, assertions, Triggers, SQL data types. Physical Data Organization - Review of terms: physical and logical records, blocking factor, pinned and unpinned organization. Heap files, Indexing, Singe level indices, numerical examples, Multi-level-indices, numerical examples, B-Trees & B+-Trees (structure only, algorithms not required), Extendible Hashing, Indexing on multiple keys – grid files.

What is relational database design?

The grouping of attributes to form "good" relation schemas

- Two levels of relation schemas:
  - The logical "user view" level
  - The storage "base relation" level
- Normalization is concerned mainly with base relations

Criteria for "good" base relations

1.1 Semantics of the Relation Attributes

GUIDELINE 1: Informally, each tuple should represent one entity or relationship instance.

- Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
- Only foreign keys should be used to refer to other entities
- Entity and relationship attributes should be kept apart as much as possible.

Bottom Line: Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.

1.2 Redundant Information in Tuples and Update Anomalies

- Mixing attributes of multiple entities may cause problems

- Information is stored redundantly wasting storage
- Problems with update anomalies:
  - Insertion anomalies
  - Deletion anomalies
  - Modification anomalies

GUIDELINE 2: Design a schema that does not suffer from the insertion, deletion and update anomalies. If there are any present, then note them so that applications can be made to take them into account.

1.3 Null Values in Tuples

GUIDELINE 3: Relations should be designed such that their tuples will have as few NULL values as possible

- Attributes that are NULL frequently could be placed in separate relations (with the primary key)

- Reasons for nulls:

  a. attribute not applicable or invalid
  b. attribute value unknown (may exist)
  c. value known to exist, but unavailable

1.4 Spurious Tuples

- Bad designs for a relational database may result in erroneous results for certain JOIN operations
- The "lossless join" property is used to guarantee meaningful results for join operations

GUIDELINE 4: The relations should be designed to satisfy the lossless join condition. No spurious tuples should be generated by doing a natural-join of any relations.

Functional Dependencies

- Functional dependencies (FDs) are used to specify *formal measures* of the "goodness" of relational designs
- FDs and keys are used to define **normal forms** for relations
- FDs are **constraints** that are derived from the *meaning* and *interrelationships* of the data attributes
- A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for YX -> Y in R specifies a *constraint* on all relation instances r(R)
- For any two tuples t_1 and t_2 in any relation instance r(R):

  If $t_1[X]=t_2[X]$, *then* $t_1[Y]=t_2[Y]$

- X -> Y holds if whenever two tuples have the same value for X, they *must have* the same value for Y
- FDs are derived from the real-world constraints on

the attributes An FD is a property of the attributes in the

schema R

- The constraint must hold on *every relation instance* r(R)
- If K is a key of R, then K functionally determines all attributes in R (since we never have two distinct tuples with $t_1[K]=t_2[K]$)

Armstrong's inference rules:

A1. (Reflexive) If Y subset-of X, then X -> Y

A2. (Augmentation) If X -> Y, then XZ -> YZ

(Notation: XZ stands for X U Z)

A3. (Transitive) If X -> Y and Y -> Z, then X -> Z

- A1, A2, A3 form a *sound* and *complete* set of inference rules

Some additional inference rules that are useful:

(Decomposition) If X -> YZ, then X -> Y and X -> Z

(Union) If X -> Y and X -> Z, then X -> YZ

(Psuedotransitivity) If X -> Y and WY -> Z, then WX -> Z

The last three inference rules, as well as any other inference rules, can be deduced from A1, A2, and A3 (completeness property)

CLOSURE

Closure of a set F of FDs is the set F^+ of all FDs that can be inferred from F

- Closure of a set of attributes X with respect to F is the set X^+ of all attributes that are functionally determined by X
- X^+ can be calculated by repeatedly applying A1, A2, A3 using

Equivalence of Sets of FDs

- Two sets of FDs F and G are **equivalent** if: every FD in F can be inferred from G, *and* every FD in G can be inferred from F. Hence, F and G are equivalent if $F^+ = G^+$
- Definition: F **covers** G if every FD in G can be inferred from F (i.e., if G subset-of F)

- F and G are equivalent if F covers G and G covers F
- There is an algorithm for checking equivalence of sets of FDs

Minimal Sets of FDs

A set of FDs is **minimal** if it satisfies the following conditions:

(1) Every dependency in F has a single attribute for its RHS.

(2) We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.

(3) We cannot replace any dependency X -> A in F with a dependency Y -> A, where Y proper-subset-of X and still have a set of dependencies that is equivalent to F.

- Every set of FDs has an equivalent minimal set
- There can be several equivalent minimal sets
- There is no simple algorithm for computing a minimal set of FDs that is equivalent to a set F of FDs
- Having a minimal set is important for some relational design algorithms
- **Normalization**: Process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations

- **Normal form**: Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form
- 2NF, 3NF, BCNF based on keys and FDs of a relation schema
- 4NF based on keys, MVDs; 5NF based on keys,

-Additional properties may be needed to ensure a good relational design (lossless join, dependency preservation;

The purpose of normalizing data

When we design a database for a relational system, the main objective in developing a logical data model is to create an accurate representation of the data, its relationships and constraints. To achieve this objective, we must identify a suitable set of relations. A technique that we can use to help identify such relations is called normalization. Normalization is a technique for producing a set of relations with desirable properties, given the data requirements of an enterprise. Normalization supports database designers by presenting a series of tests, which can be applied to individual relations so that a relational schema can be normalized to a specific form to prevent the possible occurrence of update anomalies.

FIRST NORMAL FORM

First normal form (1NF) is now considered to be part of the formal definition of a relation in the basic (flat) relational model. It states that:

1. The domain of an attribute must include only atomic (simple, indivisible) values and
2. That the value of any attribute in a tuple must be a single value from the domain of that attribute. Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple. In other words, 1NF disallows relations within relations or relations as attribute values within tuples. The only attribute values permitted by 1NF are single atomic (or indivisible) values.

Consider the DEPARTMENT relation schema, whose primary key is Dnumber, and suppose that we extend it by including the Dlocations attribute. Assuming each department can have a number of locations. This is not in 1NF because Dlocations is not an atomic attribute

SECOND NORMAL FORM

Second normal form (2NF) is based on the concept of full functional dependency. Functional Dependency: The attribute B is fully functionally dependent on the attribute A if each value of A determines one and only one value of B.

Example: PROJ_NUM, PROJ_NAME In this case, the attribute PROJ_NUM is known as the determinant attribute and the attribute PROJ_NAME is known as the dependent attribute.

Generalized Definition: Attribute A determines attribute B (that is B is functionally dependent on A) if all of the rows in the table that agree in value for attribute A also agree in value for attribute B. Fully functional dependency (composite key) If attribute B is functionally dependent on a composite key A but not on any subset of that composite key, the attribute B is fully functionally dependent on A. Partial Dependency: When there is a functional dependence in which the determinant is only part of the

primary key, then there is a partial dependency. For example if (A, B) ◊ (C, D) and B◊ C and (A, B) is the primary key, then the functional dependence B◊C is a partial dependency. {Ssn, Pnumber} →Hours is a full dependency (neither Ssn → Hours nor Pnumber→Hours holds). However, the dependency {Ssn, Pnumber}→Ename is partial because Ssn→Ename holds.

Different anomalies in designing a database, The idea of normalization, Functional dependency, Armstrong's Axioms (proofs not required), Closures and their computation, Equivalence of Functional Dependencies (FD), Minimal Cover (proofs not required). First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), Boyce Codd Normal Form (BCNF), Lossless join and dependency preserving decomposition, Algorithms for checking Lossless Join (LJ) and Dependency Preserving (DP) properties.

normalization

- Normalization is the **process of organizing the data in thedatabase**
- It is used to **minimize the redundancy** from a relation or setof relations
- It is also used to **eliminate the insertion anomaly,updateanomaly and deletion anomaly**
- It divides the larger table into the smaller table and linksthem using relationship

Update anomalies
Deletion anomaliesInsert anomalies

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

Functional dependency

- A functional dependency is an **association between twoattributes of the same relational database table**.
- One of the attributes is called the **determinant** and the otherattribute is called the **determined**
- If **A** is the **determinant** and **B** is the **determined** then we saythat **A functionally determines B** and graphically represent this as **A -> B**

The following table illustrates A ⟹ B:

A	B
1	1
2	4
3	9
4	16
2	4
7	9

The following table illustrates that A does not functionally determine B:

A	B
1	1
2	4
3	9
4	16
3	10

Armstrong's Axioms

- Armstrong's Axiom is a **mathematical notation used to findthe functional dependencies in a database**.
- Conceived by **William W. Armstrong**
- **It is a list of axioms or inference rules** that can beimplemented on any relational database.
- It is denoted by the symbol **F+**.

Various Axioms Rules

A. Primary Rule

Rule 1	**Reflexivity** If A is a set of attributes and B is a subset of A, then A holds B. { A → B }
Rule 2	**Augmentation** If A hold B and C is a set of attributes, then AC holds BC. {AC → BC} It means that attribute in dependencies does not change the basic dependencies.
Rule 3	**Transitivity** If A holds B and B holds C, then A holds C. If {A → B} and {B → C}, then {A → C} A holds B {A → B} means that A functionally determines B.

B. Secondary Rules

Rule 1	**Union** If A holds B and A holds C, then A holds BC. If{A → B} and {A → C}, then {A → BC}
Rule 2	**Decomposition** If A holds BC and A holds B, then A holds C. If{A → BC} and {A → B}, then {A → C}
Rule 3	**Pseudo Transitivity** If A holds B and BC holds D, then AC holds D. If{A → B} and {BC → D}, then {AC → D}

Closure Of Functional Dependency

- The Closure Of Functional Dependency means the **complete set of all possible attributes that can be functionallyderived from given functional dependency**
- If "**F**" is a **functional dependency** then **closure of functional dependency** can be denoted using "**{F}+** ".
- There are **three steps** to calculate closure of functional dependency

Step-1 : Add the attributes which are present on Left Hand Side in the original functional dependency.

Step-2 : Now, add the attributes present on the Right Hand Side of the functional dependency.

Step-3 : With the help of attributes present on Right Hand Side, check the other attributes that can be derived from the other given functional dependencies. Repeat this process until all the possible attributes whichcan be derived are added in the closure.

Example:

Consider the table student_details having (Roll_No, Name,Marks, Location) as theattributes and having two functional dependencies.

FD1 : Roll_No -> Name, Marks FD2 :
Name -> Marks, Location

Step-1: {Roll_no}+ = {Roll_No}

Step-2 : {Roll_no}+ = {Roll_No,Name, Marks}

Step-3 : **{Roll_no}+ = {Roll_No, Marks, Name, Location}**

Step-1 : {Name}+ = {Name}

Step-2 : {Name}+ = {Name, Marks, Location}

Step-3 : Since, we don't have any functional dependency where "Marks or Location".So **{Name}+ = {Name, Marks, Location}**

{Marks}+ = {Marks} and **{Location}+ = { Location}**

Equivalence of Functional Dependencies (FD)

- **Two different sets of functional dependencies for a given relation mayor may not be equivalent.**
- If **FD1 can be derived from FD2**, we can say that FD2 ➵ FD1.
- If **FD2 can be derived from FD1**, we can say that FD1 ➵ FD2.
- If above two cases are true, **FD1=FD2.**

Eg:A relation R(A,B,C,D) having two FD sets FD1 = {A->B, B->C, AB->D} and FD2 = {A->B, B->C, A->C, A->D}

Step 1. **Checking whether all FDs of FD1 are present in FD2**

- A->B YES
- B->C YES
- AB->D YES For set FD2, (AB)+ = {A,B,C,D}.FD2 ➵ FD1 is true.

Step 2. **Checking whether all FDs of FD2 are present in FD1**

- A->B YES
- B->C YES
- A->C YES For set FD1, (A)+ = {A,B,C,D}.
- A->D YES For set FD1, (A)+ = {A,B,C,D}FD1 ➵ FD2 is true.

Step 3. As FD2 ➵ FD1 and FD1 ➵ FD2 both are true FD2 =FD1 is true. These two FD sets are semantically equivalent.

Minimal Cover

- Whenever a user updates the database, the system must check whether any of the functional dependencies are getting violated in this process. If there is a violation of dependencies in the new database state, the system must roll back. Working with a huge set of functional dependencies can cause unnecessary added computational time. This is where the minimal cover comes into play.

There are **4 rules** to find Minimal cover :

1. Break down the RHS of each functional dependency into a single attribute
2. Find redundant fds
3. Minimize LHS.
4. Group the functional dependencies that have common LHS together into a SingleFD .

Q1. Minimal cover of F with dependencies F={BC->ADEF, F->DE} ?

STEP 1: Break down the RHS

BC->A BC->D BC->E BC->F F->D F->E

STEP 2: Find redundant fds

- Assume BC->A is redundant fd and we remove this fd,now try computing (BC)+={BCDEF} but there is no A. so BC->A is not redundant
- BC->D (BC)+={BCAEFD}, D is present so BC->D is redundant
- BC->E (BC)+={BCADFE}, E is present so BC->E is redundant
- BC->F (BC)+={BCADE}, F is not present so BC->F is not redundantso we get

BC->A BC->F F->D F->E

BC->A BC->F F->D F->E

STEP 3: Minimize LHS

BC->A BC->F

- From BC-> A, if we remove B and then we get C->A. By taking closure (C)+={C,A}, there is no B. same way remove C then B->A. By taking closurethere is no C. So it can't be minimize
- BC->F ,it also can't be minimizeso

we get

BC->A BC->F F->D F->E

Step 4: Group the functional dependencies that have common LHS togetherinto a Single FD .

so the minimal cover is

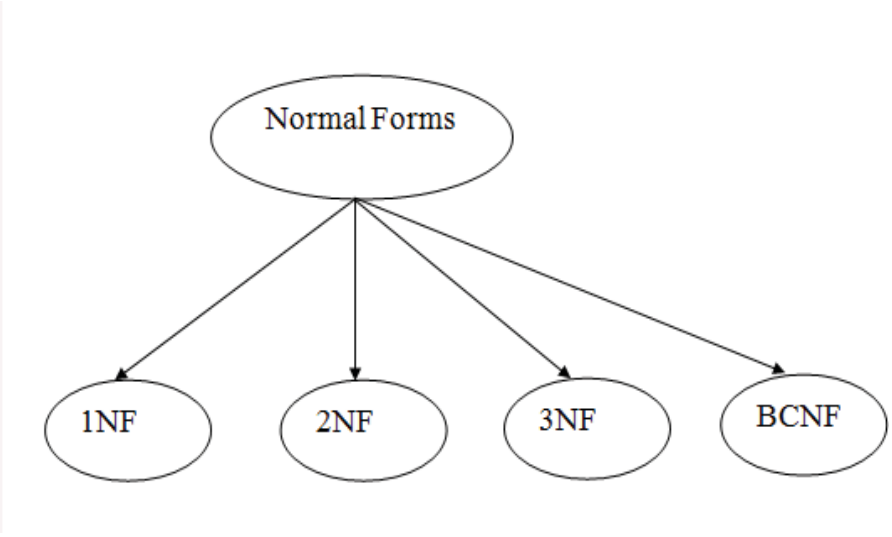
BC->A F F->DE

index

- ❑ First Normal Form (1NF)
- ❑ Second Normal Form (2NF)
- ❑ Third Normal Form (3NF)
- ❑ Boyce Codd Normal Form (BCNF)

normalization

- Normalization is the **process of organizing the data in thedatabase**
- It is used to **minimize the redundancy** from a relation or setof relations
- It is also used to **eliminate the insertion anomaly,updateanomaly and deletion anomaly**
- It divides the larger table into the smaller table and linksthem using relationship



First Normal Form (1NF)

- A relation will be **1NF if it contains an atomic value.**
- It states that an attribute of a table cannot hold multiplevalues. It must hold only single-valued attribute.

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

- In the 2NF, **relational must be in 1NF**.
- In the second normal form, **No non-prime attribute is dependent onthe proper subset of any candidate key of table**

Non-prime attribute: An attribute that is not a part of any candidatekey

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

- **In 3NF,the relation must be in 2NF**
- Transitive functional dependency of non-prime attribute on any superkey should be removed

Transitive functional dependency: **A->B** & **B->C** ,THEN **A->C**

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

{EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_I D, EMP_NAME, EMP_ZIP}. so on

Candidate key: {EMP_ID}

Non-prime attributes: In the given table,all attributes except EMP_ID are non-prime

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Boyce Codd Normal Form (BCNF)

- BCNF is the **advance version of 3NF**.
- A table is in BCNF if every functional dependency X C Y, X is the super key of the table.

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows: 1.EMP_ID C EMP_COUNTRY

2.EMP_DEPT C {DEPT_TYPE, EMP_ DEPT_NO}

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Candidate key: {EMP-ID, EMP-DEPT}

index

- ❑ Lossless join and dependency preserving decomposition
- ❑ Algorithms for checking Lossless Join (LJ)

Lossless join and dependency preservingdecomposition

- **Decomposition** of a relation is done when a relation in relational model is not in appropriate normal form.
- Relation R is decomposed into two or more relations if decomposition is lossless join as well as dependency preserving.

Lossless Join Decomposition

- If the information is **not lost from the relation that is decomposed**, then thedecomposition will be **lossless**.
- ie, the relation is said to be lossless decomposition if natural joins of all thedecomposition give the original relation.

- If we decompose a relation R into relations R1 and R2

1. Decomposition is lossy if R1 œ R2 is not R
2. Decomposition is lossless if R1 œ R2 is equal to R

EMPLOYEE_DEPARTMENT table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

œ

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

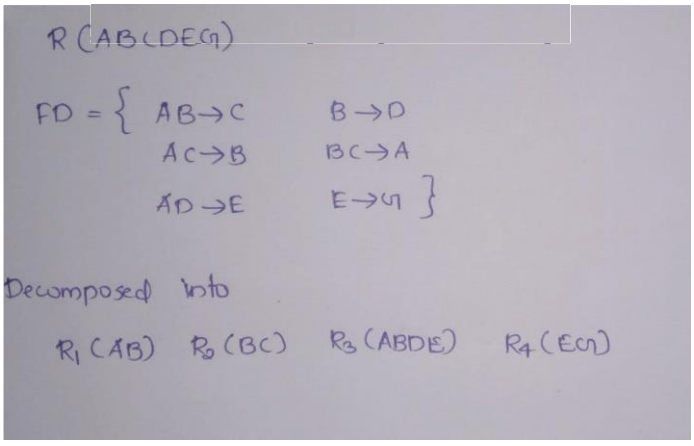
Employee

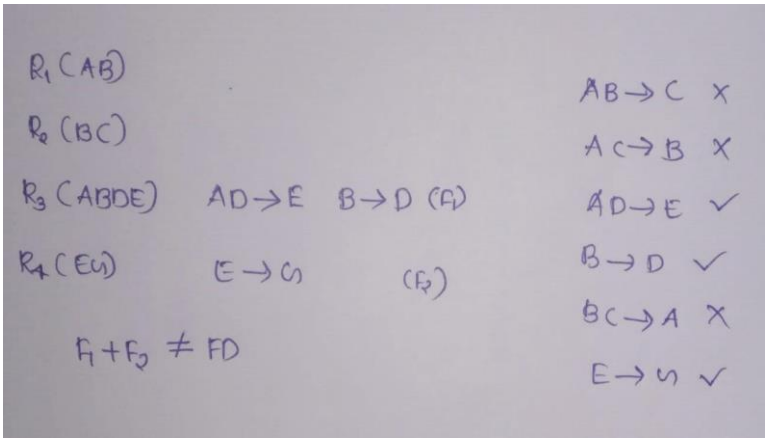
DEPT_ID	EMP_ID	DEPT_NAME
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Department

Dependency Preserving Decomposition

- **The dependency preservation property, which ensures that each functional dependency is represented in some individual relationresulting after decomposition**
- In the dependency preservation, **at least one decomposed table must**





Algorithms for checking Lossless Join (LJ)

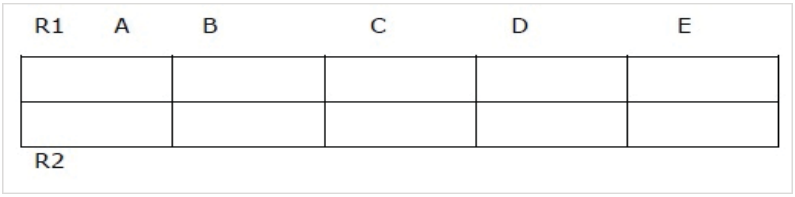
R(A,B,C,D,E)

F:{A->B, BC->E, ED->A}

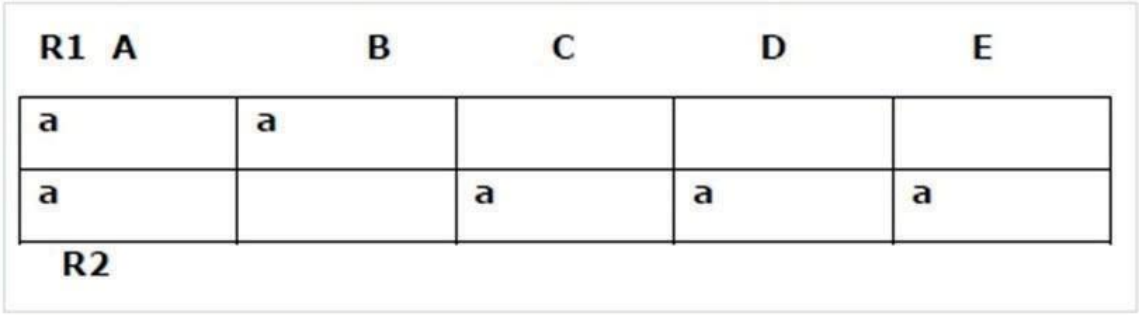
R is decomposed into R1(AB) and R2(ACDE)

Step 1 − Create a table with M rows and N columns

- **M= number of decomposed relations.**
- N= number of attributes of original relation.



Step 2 − If a decomposed relation Ri has attribute A thenInsert a symbol (say 'a') at position (Ri,A)



Step 3 – Consider each FD X->Y

If column X has two or more symbols then

Insert symbols in the same place (rows) of column Y.

Now let us insert symbol ‘a’ for A->B in second column, second row

R1 A	B	C	D	E
a	a			
a	a	a	a	a

R2

Step 4 – If any row is completely filled with symbols then

Decomposition is lossless.

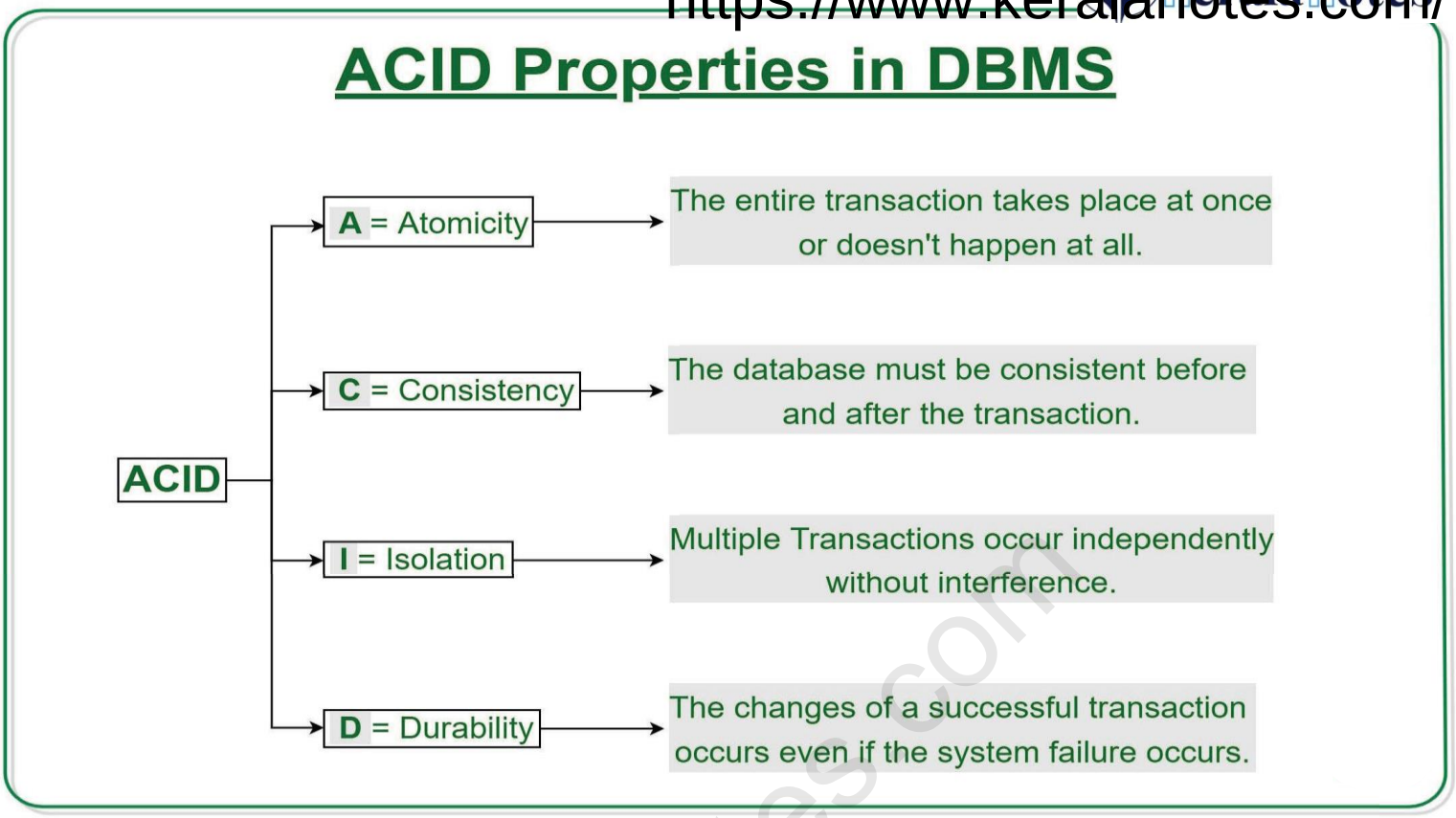
Else R2 is completely filled => decomposition is lossless.

Decomposition is lossy.

Transaction Processing Concepts - overview of concurrency control, Transaction Model, Significance of concurrency Control & Recovery, Transaction States, System Log, Desirable Properties of transactions. Serial schedules, Concurrent and Serializable Schedules, Conflict equivalence and conflict serializability, Recoverable and cascade-less schedules, Locking, Two-phase locking and its variations. Log-based recovery, Deferred database modification, check-pointing. Introduction to NoSQL Databases, Main characteristics of Key-value DB (examples from: Redis), Document DB (examples from: MongoDB) Main characteristics of Column - Family DB (examples from: Cassandra) and Graph DB (examples from : ArangoDB)

transaction

- **Transactions** group a set of tasks into a single execution unit.
- Each transaction begins with a specific task and endswhen all the tasks in the group successfully complete.
- If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: **successor failure.**
- Incomplete steps result in the failure of the transaction.
- A database transaction, by definition, **must be**



CONCURRENCY CONTROL

Concurrency Control in Database Management System is a procedure of **managing simultaneous operations without conflicting with each other**

❑ Concurrency Control Protocols

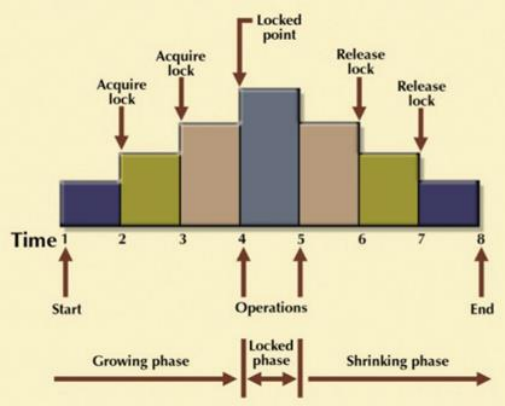
- Lock-Based Protocols
- Two Phase Locking Protocol
- Timestamp-Based Protocols
- Validation-Based Protocols

1. LOCK-BASED PROTOCOLS

- Lock Based Protocols in DBMS is a mechanism in which a **transaction cannot Read or Write the data until it acquires an appropriate lock**.
- Lock based protocols help to eliminate the concurrency problem in DBMS for simultaneous **transactions** by locking or **isolating a particulartransaction to a single user.**
- All **lock requests** are made **to** the **concurrency-control manager**. Transactions proceed only once the lock request is granted.

TWO PHASE LOCKING PROTOCOL (2 PL Protocol)

- It is a method of concurrency control in DBMS that **ensures serializability** by applying a **lock to the transaction data which blocks other transactions to access the same data simultaneously**



Growing Phase: In this phase transaction may obtain locks but may not release any locks.

Shrinking Phase: In this phase, a transaction may release locks but not obtain any new lock

TIMESTAMP-BASED PROTOCOLS

- It is an algorithm which uses the **System Time or Logical Counter as atimestamp to serialize the execution** of concurrent transactions.
- It ensures that every conflicting read and write operations areexecuted in a timestamp order.
- The **older transaction** is always **given priority** in this method.
- It **uses system time to determine the time stamp** of the transaction.
- This is the most **commonly used** concurrency protocol.

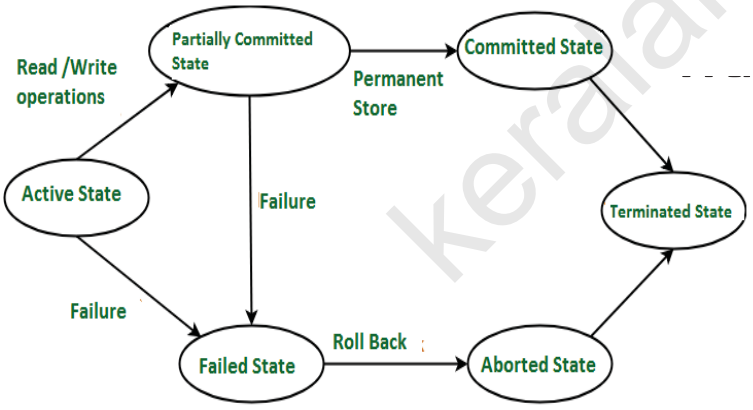
VALIDATION BASED PROTOCOL

- It is also called **Optimistic Concurrency Control Technique**.
- It is called optimistic **because of the assumption it makes**, i.e. very less interference occurs, therefore, there is no need for checking while the transaction is executed.
- Until the transaction end is reached updates in the transaction are not applied directly to the database. All updates are applied to local copies of data items kept for the transaction. **At the end of transaction** execution, while execution of the transaction, a validation phase checks whether any of transaction updates violate serializability. If there is **no violation** of serializability the **transaction is committed** and the **database is updated**

Transaction Model

Transactions access data using two operations:

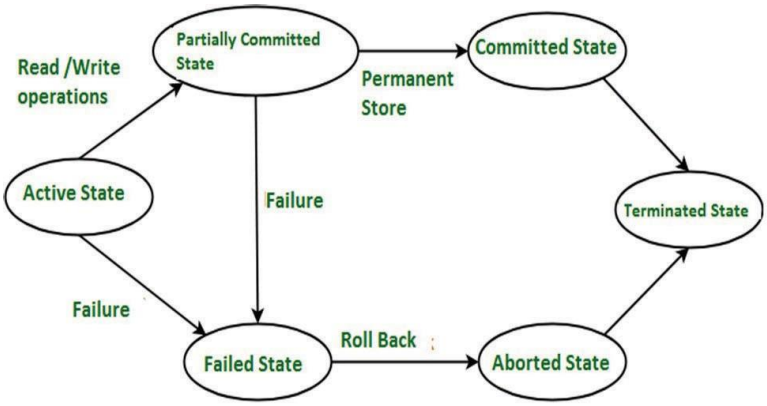
- **read(X)**, which transfers the data item X from the database to a variable, also called X, in a buffer in main memory belonging to the transaction that executed the read operation.
- **write(X)**, which transfers the value in the variable X in the main-memory buffer of the transaction that executed the write to the data item X in the database.



Transaction States in DBMS

nsaction States

- **Active State:When the instructions of the transaction are running** then the transaction is in active state. If all the '**read and write**' operations are performed without any error then it goes to the "**partially committed state**"; if any instruction **fails**, it goes to the "**failed state**".
- **Partially Committed:After completion of all the read and write** operation the changes are made in main memory or local buffer. If the the **changes are made permanent** on the Data Base then the state will change to "**committed state**" and in case of **failure** it will go to the "**failed state**".
- **Failed State**:When **any instruction of the transaction fails**, it goes to the "failed state"



Transaction States in DBMS

- **Aborted State** :**After having any type of failure** the transaction goes from "**failed state" to "aborted state"**

- **Committed State:It is the state when the changes are made permanent on the Data Base** and the transaction is complete and therefore terminated in the "**terminated state**".
- **Terminated State** :the **transaction comes** from the "**committed state**" goes to this state, then the **system is consistent and ready for new transaction** and the old transaction is terminated.

System Log

- **Log is a sequence of records**, which maintains the records of actions performed bya transaction.
- It is **important that the logs are written prior to the actual modification andstored on a stable storage media**, which is failsafe.

Log-based recovery works as follows –

- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it.
- **<Tn, Start>**

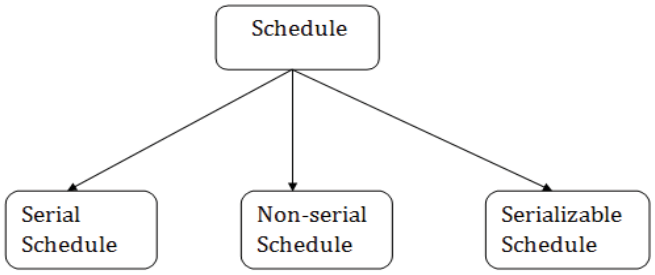
- When the transaction modifies an item X, it write logs as **<Tn, X, V1, V2>.**It readsTn has changed the value of X, from V1 to V2.
- When the transaction finishes, it logs **<Tn, commit>**

index

- Schedules
- Conflict equivalence
- Recoverable and cascade-less schedules
- Deferred database modification
- check-pointing

schedule

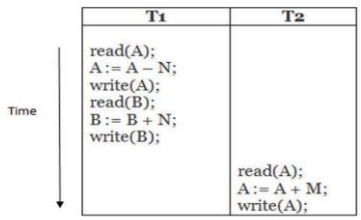
- A **series of operation from one transaction to another** transaction isknown as schedule.
- It is used to preserve the order of the operation in each of theindividual transaction.



SERIAL SCHEDULE

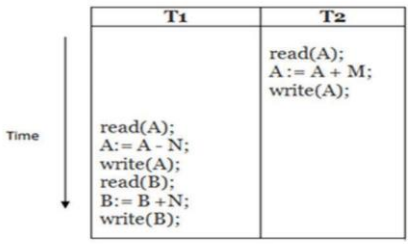
- The serial schedule is a type of schedule where one transaction is executedcompletely before starting another transaction.
- In the serial schedule, when the first transaction completes its cycle, thenthe next transaction is executed.

(a)



Schedule A

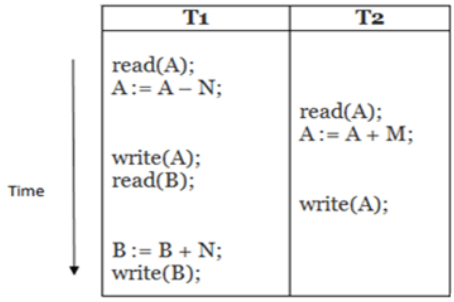
(b)



Schedule B

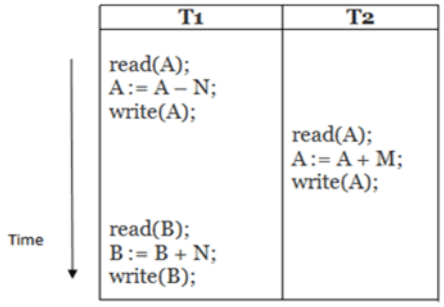
- If interleaving of operations is allowed, then there will be non-serialschedule.
- It contains many possible orders in which the system can execute theindividual operations of the transactions.

(c)



Schedule C

(d)



Schedule D

Serializable schedule

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

SERIALIZABILITY is a concept that helps us to **check which schedules are serializable.**

Conflict equivalence

Two schedules are said to be conflict equivalent if and only if:

1. They **contain the same set of the transaction.**
2. If each pair of **conflict operations are ordered in the same way.**

Non-serial schedule

T1	T2
Read(A) Write(A)	
	Read(A) Write(A)
Read(B) Write(B)	
	Read(B) Write(B)

Schedule S1

Serial Schedule

T1	T2
Read(A) Write(A) Read(B) Write(B)	
	Read(A) Write(A) Read(B) Write(B)

Schedule S2

Recoverable Schedule

A recoverable schedule is one where, for each pair of Transaction T_i and T_j such that **T_j reads data item previously written by T_i the commit operation of T_i appears before the commit operation T_j.**

T8	T9		
read(A)			T9 is dependent on T8
write(A)			Non recoverable schedule if T9 commits before T8
	read(A)		
read(B)			

T10	T11	T12
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	

ess schedules

Schedule based on Recoverability

Recoverable | Cascadeless | Strict

Transac action
T11 wri int T10
fails. T1 e rolled
back, T

This ph of
transaction rollbacks is called Cascading rollback.

- **Restrict the schedules to those where cascading rollbacks cannot occur**, Such schedules are called **Cascadeless Schedules**.
- A cascadeless schedule is one where for each pair of transaction T_i and T_j such that **T_j reads data item, previously written by T_i the commit operation of T_i appears before the read operation of T_j**
- Every Cascadeless schedule is also recoverable schedule.

T10	T11
read(A)	
write(A)	
	read(B)
commit	
	read(A)

Deferred database modification

- The deferred modification technique occurs if the transaction does not modify the database until it has committed.
- Ie, **the changes are not applied immediately to the database.**
- In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

If database modifications occur while the transaction is still active, the transaction is said to use the immediate-modification technique.

check-pointing

- The methodology utilized for removing all previous transaction logs and storing them in permanent storage is called a **Checkpoint**.
- A checkpoint is used for recovery if there is an unexpected shutdown inthe database.
- Checkpoints work on some intervals and write all dirty pages (modifiedpages) from logs relay to data file from i.e from a buffer to physical disk. It is also known as the **hardening of dirty pages**
- It speeds up data recovery process.

Create table employee (id,name , place,designation,salary)

```
postgres=# create table employee(id int,name varchar(20),place varchar(20),designation varchar(20),salary int);
CREATE TABLE
postgres=# insert into employee values(101,'anu','manathana','clerk',20000);
INSERT 0 1
postgres=# insert into employee values(102,'vinu','kannur','staff',15000);
INSERT 0 1
postgres=# insert into employee values(103,'ammu','iritty','cashier',22000);
INSERT 0 1
postgres=# insert into employee values(104,'dein','mattannur','manager',30000);
INSERT 0 1
postgres=# select * from employee;
 id  | name |   place   | designation | salary
-----+------+-----------+-------------+--------
 101 | anu  | manathana | clerk       |  20000
 102 | vinu | kannur    | staff       |  15000
 103 | ammu | iritty    | cashier     |  22000
 104 | dein | mattannur | manager     |  30000
(4 rows)
```

Begin transaction by deleting the details of employ with id 102 and laterrollback the changes

```
postgres=# begin;
BEGIN
postgres=# delete from employee where id=102;
DELETE 1
postgres=# select * from employee;
 id  | name |   place   | designation | salary
-----+------+-----------+-------------+--------
 101 | anu  | manathana | clerk       |  20000
 103 | ammu | iritty    | cashier     |  22000
 104 | dein | mattannur | manager     |  30000
(3 rows)


postgres=# rollback;
ROLLBACK
postgres=# select * from employee;
 id  | name |   place   | designation | salary
-----+------+-----------+-------------+--------
 101 | anu  | manathana | clerk       |  20000
 102 | vinu | kannur    | staff       |  15000
 103 | ammu | iritty    | cashier     |  22000
 104 | dein | mattannur | manager     |  30000
(4 rows)
```

Begin transaction by updating salary(increment by 10%) and commitchanges being made

```
postgres=# begin;
BEGIN
postgres=# update employee set salary=salary+(salary*.1);
UPDATE 4
postgres=# select * from employee;
 id  | name |   place   | designation | salary
-----+------+-----------+-------------+--------
 101 | anu  | manathana | clerk       |  22000
 102 | vinu | kannur    | staff       |  16500
 103 | ammu | iritty    | cashier     |  24200
 104 | dein | mattannur | manager     |  33000
(4 rows)


postgres=# commit;
COMMIT
```

use save point in the transaction

```
postgres=# begin;
BEGIN
postgres=# savepoint q1;
SAVEPOINT
postgres=# update employee set salary=25000 where id=101;
UPDATE 1
postgres=# select * from employee;
 id  | name |   place   | designation | salary
-----+------+-----------+-------------+--------
 102 | vinu | kannur    | staff       |  16500
 103 | ammu | iritty    | cashier     |  24200
 104 | dein | mattannur | manager     |  33000
 101 | anu  | manathana | clerk       |  25000
(4 rows)

postgres=# savepoint q2;
SAVEPOINT
postgres=# delete from employee where id=102;
DELETE 1
postgres=# select * from employee;
 id  | name |   place   | designation | salary
-----+------+-----------+-------------+--------
 103 | ammu | iritty    | cashier     |  24200
 104 | dein | mattannur | manager     |  33000
 101 | anu  | manathana | clerk       |  25000
(3 rows)

postgres=# savepoint q3;
SAVEPOINT
postgres=# insert into employee values(105,'keethu','chala','lower division clerk',10000);
INSERT 0 1
postgres=# rollback to q2;
ROLLBACK
postgres=# select * from employee;
 id  | name |   place   | designation | salary
-----+------+-----------+-------------+--------
 102 | vinu | kannur    | staff       |  16500
 103 | ammu | iritty    | cashier     |  24200
 104 | dein | mattannur | manager     |  33000
 101 | anu  | manathana | clerk       |  25000
(4 rows)
```

index

- NoSQL Databases
  - ✓Key-value DB (examples from: Redis)
  - ✓Document DB (examples from: MongoDB)
  - ✓Column - Family DB (examples from:Cassandra)
  - ✓Graph DB (examples from : ArangoDB)

NoSQL

- **NoSQL** ("**not only SQL**") databases are non-tabular databases and store data differently than relationaltables.
- NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph.
- They provide flexible schemas and scale easily withlarge amounts of data and high user loads.

The data model we design for a NoSQL database will depend on the type of NoSQL database we choose.

Users

ID	first_name	last_name	cell	city
1	Leslie	Yepp	8125552344	Pawnee

Hobbies

ID	user_id	hobby
10	1	scrapbooking
11	1	eating waffles
12	1	working

```
{
   "_id": 1,
   "first_name": "Leslie",
   "last_name": "Yepp",
   "cell": "8125552344",
   "city": "Pawnee",
   "hobbies": ["scrapbooking", "eating waffles", "working"]
}
```

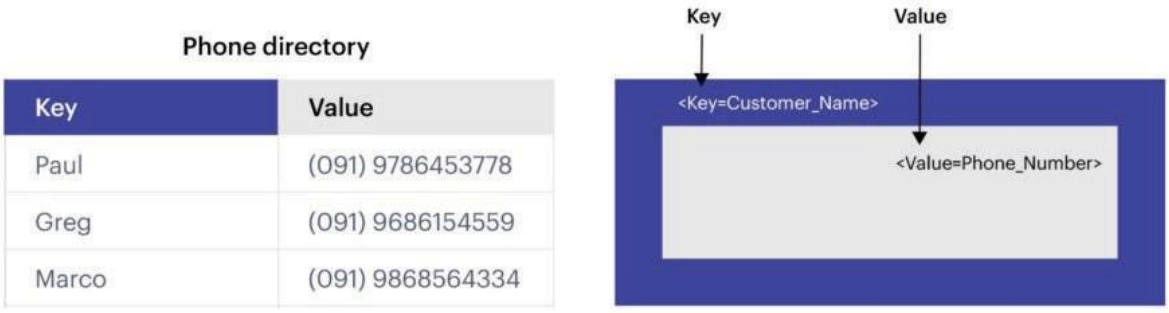
Key-value db

- A key-value database (or **key-value store**) uses a simple key-value method to store data.
- These **databases contain** a simple string (the key) that is always unique and an arbitrary large data field (the value).
- They are **easy to design and implement**.

An Example of Key-value database

Phone directory

Key	Value
Paul	(091) 9786453778
Greg	(091) 9686154559
Marco	(091) 9868564334



A simple example of key-value data store.

REDIS EXAMPLE

- For the **vast majority of data storage with Redis**, data will be stored in a simple **key/value pair**. This is best shown through the **redis-cli** (command line interface)using GET and SET commands.
- EG: we may want to store some information about books, such as the title and author of a few of our favorites.

```
> SET title "The Hobbit"
OK

>SET author "J.R.R. Tolkien"
OK
```

```
> GET title
"The Hobbit"
> GET author
"J.R.R. Tolkien"
```

DOCUMENT DB

- **Built around JSON-like documents**, document databases are **both natural and flexible** for developers to work with.
- They promise **higher developer productivity**, and **faster evolution** with application needs.
- As a class of non-relational, sometimes called NoSQL database, the document data model has become the **most popular alternative to tabular, relational databases.**

MONGODB EXAMPLE

Relational

ID	first_name	last_name	cell	city	year_of_birth	location_x	location_y
1	'Mary'	'Jones'	'516-555-2048'	'Long Island'	1986	'-73.9876'	'40.7574'

ID	user_id	profession
10	1	'Developer'
11	1	'Engineer'

ID	user_id	name	version
21	1	'DocFinder'	2.5.7

ID	user_id	make	year
30	1	'Bentley'	1973

MongoDB

```
first_name: "Mary",
last_name: "Jones",
cell: "516-555-2048",
city: "Long Island",
year_of_birth: 1986,
location: {
        type: "Point",
        coordinates: [-73.9876, 40.7574]
},
profession: ["Developer", "Engineer"],
apps: [
  { name: "MyApp",
    version: 1.0.4 },
  { name: "DocFinder",
    version: 2.5.7 }
],
cars: [
  { make: "Bentley",
    year: 1973 },
  { make: "Rolls Royce",
    year: 1965 }
]
```

Column - Family DB

Column-family databases **store data in column families as rows that have many columns associated with a row key**

Example:

- **RDBMS:** Table having the columns ID , Name , Age, Gender, City.

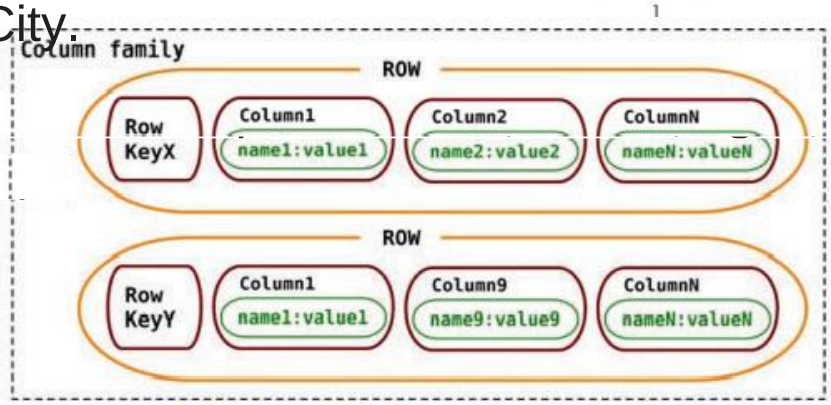
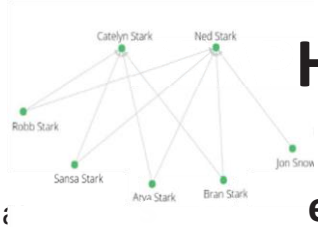


Figure 10.1. Cassandra's data model with column families

Cassandra EXAMPLE

```
//column family
{
//row
  "pramod-sadalage" : {
     firstName: "Pramod",
     lastName: "Sadalage",
     lastVisit: "2012/12/12"
  }
//row
  "martin-fowler" : {
     firstName: "Martin",
     lastName: "Fowler",
     location: "Boston"
  }
}
```



H DB

- Graph dat **ema-free objects** (vertices or nodes) where arbitrary data can be stored (properties) and **relations between the objects** (edges). **store sch**
- **Edges** typically have a direction going from **one object to another or multiple objects**.
- Vertices and edges form a network of data points which is called a "**graph**".

ArangoDB EXAMPLE

```
LET data = [
   {
      "parent": { "name": "Ned", "surname": "Stark" },
      "child": { "name": "Robb", "surname": "Stark" }
   }, {
      "parent": { "name": "Ned", "surname": "Stark" },
      "child": { "name": "Sansa", "surname": "Stark" }
   }, {
      "parent": { "name": "Ned", "surname": "Stark" },
      "child": { "name": "Arya", "surname": "Stark" }
   }, {
      "parent": { "name": "Ned", "surname": "Stark" },
      "child": { "name": "Bran", "surname": "Stark" }
}……..
]
```