CSC 441: Database Systems

Vikram Pudi
vikram@iiit.ac.in

Outline

- About the course
- DBMS and OS
- Database Systems-- Background

Objectives

- Databases have become essential part of every business. To store data, a specialized software called database management system (DBMS) or database system has been developed over several decades. A database system can be used to manage large amounts of data in a persistent manner.
- The objective of this course to study the methods that have been evolved to build DBMS software in a focused manner which include storage management, query processing and transaction management.

Course topics

- Introduction (3 hours)
- Storage management and Indexing
  - Data storage (3 hours)
  - Representing data elements (3 hours)
  - Index structures (3 hours)
  - Multidimensional indexes (6 hours)
- Query processing
  - Query execution (6 hours)
  - The query compiler (6 hours)
- Concurrency control and Recovery
  - Coping with system failures (3 hours)
  - Concurrency control (6 hours)
  - More about transaction management (6 hours)

References

- Main book
  - Database System Implementation, Hector Garcia-Molina, Jeffrey D. Ullman and Jennifer Widon, Pearson Education, 2003
- Other books
  - Elmasri & Navathe, Fundamentals of Database Systems, Pearson Education, 5th Education.
  - Raghu Ramakrishnan and Johannes Gehrke, Database Management Systems, Third edition, Mc Graw Hill, 2003.
  - Abraham Silberschatz, Henry F.Korth, S.Sudarshan, Database system concepts, fifth edition, Mc Graw Hill, 2006.
- Tutorials
  - 2 hours per week

List of Some Research Papers

- A relational model of data for large shared data banks
- A History and Evaluation of System R
- Operating System Support for Database Management.
- A column-oriented DBMS
- PicoDBMS: Scaling down Database Techniques for the Smartcard
- Spanner: Google's Globally-Distributed Database
- Monitoring Streams – A New Class of Data Management Applications
- Speculative locking protocols to improve performance for distributed database management systems.

LAB WORK

- Lab assignments
  - Implementation of relational operators, indexing algorithms, optimization algorithms and concurrency control protocols.
- Reading assignments
- Quiz

GRADING

- MidSem-1: 15%;
- MidSem-II: 15 %;
- EndSem: 40 %;
- Assignment/Lab/Quiz: 30 %
- Note:
  - you should get at least 30% in each exam to pass
  - Minimum 40 % in the lab to pass.
  - Minimum 20 % in each assignment to pass.

OUTCOME

- The course will help the students in understanding the fundamental concepts of several database management systems like ORACLE, DB2, SYBASE and so on.
- Also, the students will understand the solutions/options to interesting problems which have been encountered by the designers of preceding DBMSs.
- Most important, the students are exposed to internal design of DBMSs and able to tune the DBMSs to meet the performance demands of diverse applications. .

Database Systems - Background

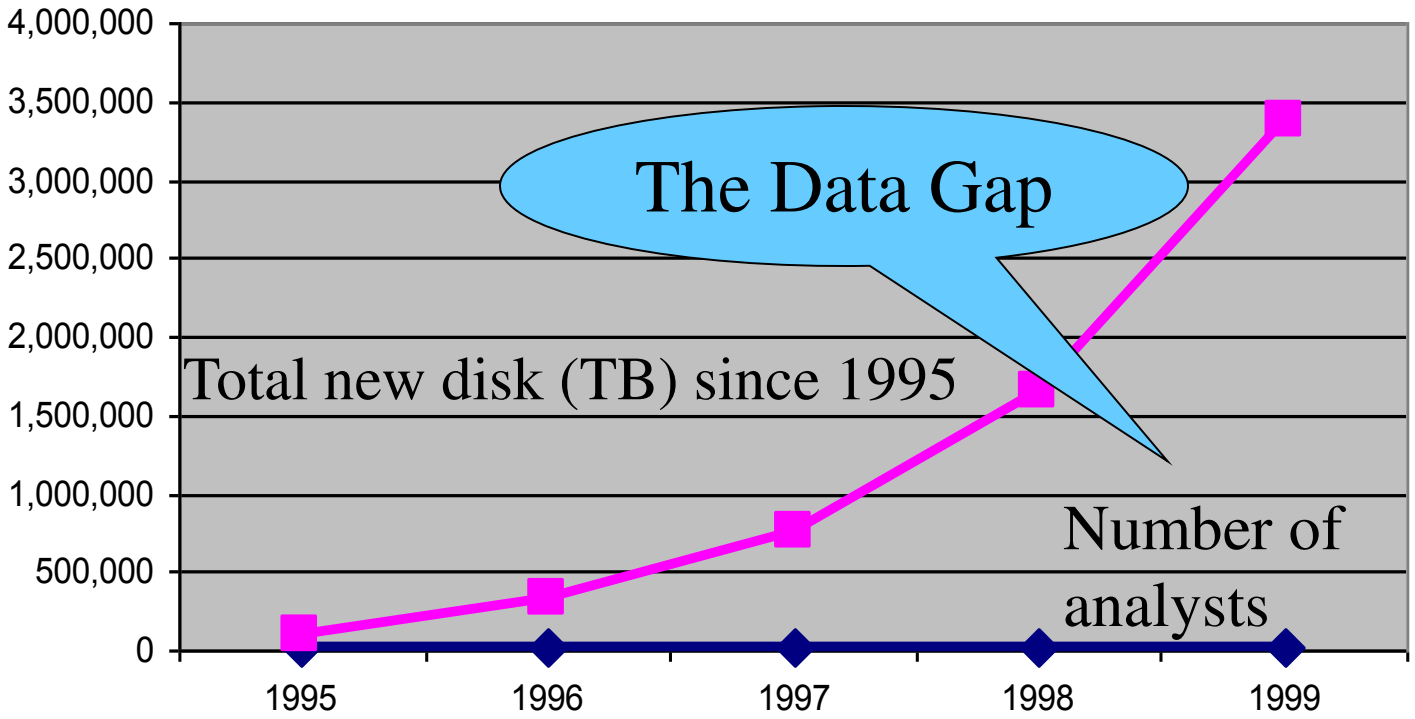
- About data science
- Database
- ER Models
- Relational Data Model
- Relational Algebra
- SQL & Tuple Relational Calculus
- Functional Dependencies & Normalization

About Data and information

- Data
  - Symbols
  - Collection of facts
  - It simply exists and has no significance beyond its existence.
  - It does not have any meaning of itself
  - Example: It is raining
- Information:
  - Data that are processed to be useful, **provides answers to who, what, and when questions.**
  - Processed data
  - Data that has been given meaning by relational connection
    - Ex: If temperature drops 15 degrees and then it started raining.

Mining Large Data Sets - Motivation

- **There is often information "hidden" in the data that is not readily evident**
- **Human analysts may take weeks to discover useful information**
- **Much of the data is never analyzed at all**



From: R. Grossman, C. Kamath, V. Kumar, "Data Mining for Scientific and Engineering Applications"

Evolution of Sciences…

- Before 1600, **empirical science**
  - Experimental, Observation, experience
- 1600-1950s, **theoretical science**
  - Application of mathematical methods
  - Each discipline has grown a *theoretical* component. Theoretical models often motivate experiments and generalize our understanding.
- 1950s-1990s, **computational science**
  - Over the last 50 years, most disciplines have grown a third, *computational* branch (e.g. empirical, theoretical, and computational ecology, or physics, or linguistics.)
  - The application of computer simulation and other forms of computation to problems in various scientific disciplines.
  - Computational Science traditionally meant simulation. It grew out of our inability to find closed-form solutions for complex mathematical models.
    - In mathematics, an expression is said to be a **closed-form expression** if, and only if, it can be expressed analytically in terms of a bounded number of certain "well-known" functions.

Evolution of Sciences

- 1990-now, **data science**
  - The flood of data from new scientific instruments and simulations
  - The ability to economically store and manage petabytes of data online
  - The Internet and computing Grid that makes all these archives universally accessible
  - Scientific info. management, acquisition, organization, query, and visualization tasks scale almost linearly with data volumes.
- Data science enables the creation of data products.
- Data mining is a major new challenge!

- Jim Gray and Alex Szalay, *The World Wide Telescope: An Archetype for Online Science*, Comm. ACM, 45(11): 50-54, Nov. 2002

About Data Science

- Commerce and research is being transformed by data-driven discovery and prediction. Skills required for data analytics at massive levels – scalable data management on and off the cloud, parallel algorithms, statistical modeling, and proficiency with a complex ecosystem of tools and platforms – span a variety of disciplines and are not easy to obtain through conventional curricula.
- Basic techniques of data science, including
  - both SQL and NoSQL solutions for massive data management (e.g., MapReduce and contemporaries), algorithms for data mining (e.g., clustering and association rule mining), and basic statistical modeling (e.g., logistic and non-linear regression).

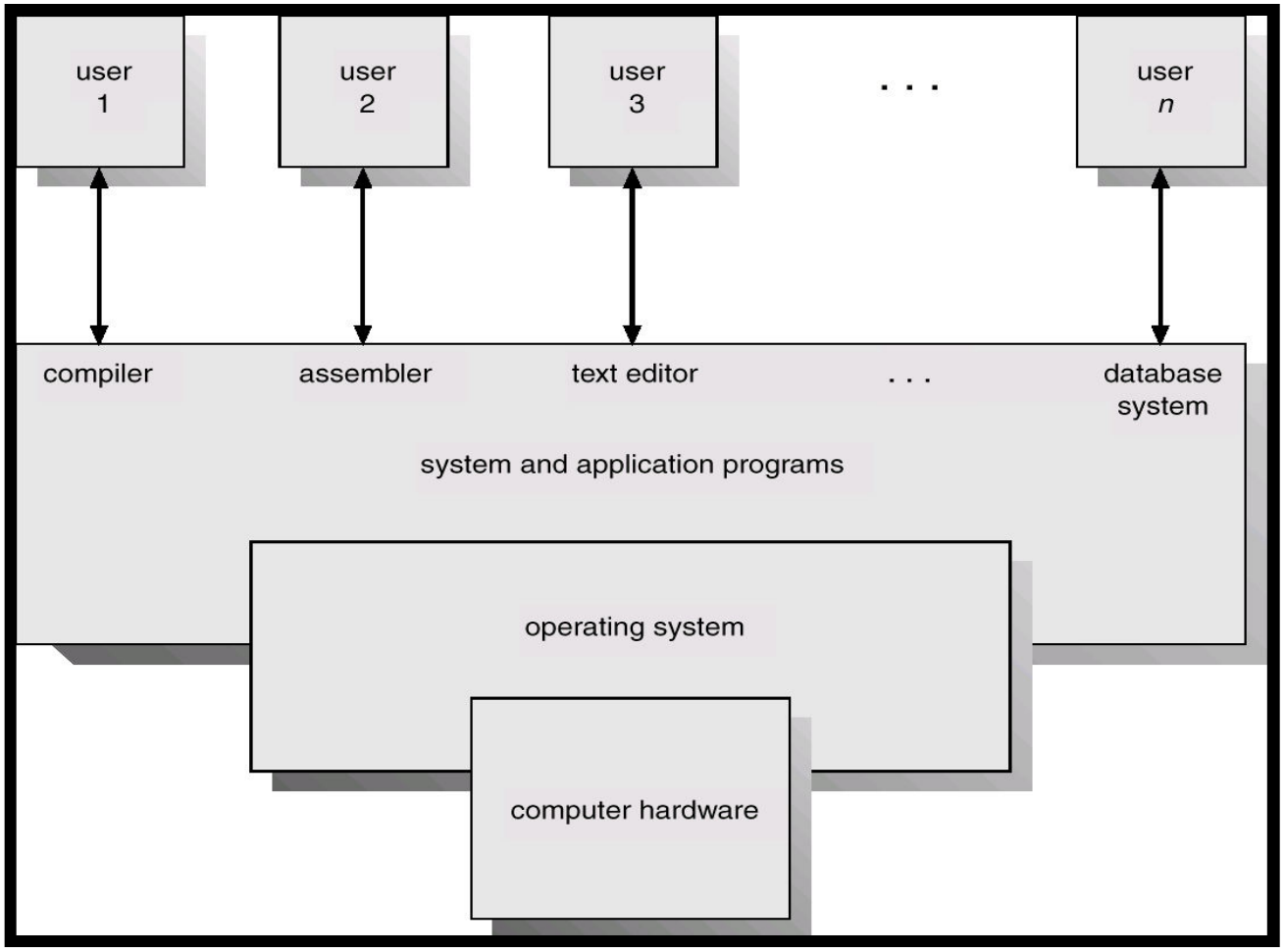
Courses You Have Completed

- Digital logic design
- Programming language
- Computer System Organization
- Scripting/IT workshop
- Operating Systems
- Introduction to databases/SQL/database design

Computer System Components

1. **Hardware** – provides basic computing resources (CPU, memory, I/O devices).
2. **Operating system** – controls and coordinates the use of the hardware among the various application programs for the various users.
3. **Applications programs** – define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).
4. **Users** (people, machines, other computers).

Abstract View of Computer System



About Storing and accessing data with programming language

- Suppose there is no DBMS.
  - Create a record.
  - Create a file of records
  - Write different programs for different requirements.
  - For small applications it is OK.
- What about big database ?
  - Amazon
    - Data of 20 million books occupies 2 terabytes (one tera byte is 10^{12} bytes) maintained on 200 different servers.
    - About 15 million visitors visit the site every day.
    - Database is continually updated
    - Stock quantities are updated whenever a purchase is made.
    - About 100 people keep the database up-to-date.
- Database can be managed manually
  - A library card catalog.

Database Systems - Background

- Database
- ER Models
- Relational Data Model
- Relational Algebra
- SQL & Tuple Relational Calculus
- Functional Dependencies & Normalization

Basic Definitions

- **Database**: A collection of related data.
- **Data**: Known facts that can be recorded and have an implicit meaning.
- **Mini-world**: Some part of the real world about which data is stored in a database. For example, student grades and transcripts at a university.
  - Database should be the reflection of realworld.
- **Database Management System (DBMS)**: A software package/ system to facilitate the creation and maintenance of a computerized database.
- **Database System**: The DBMS software together with the data itself. Sometimes, the applications are also included.

Typical DBMS Functionality

- **Define a database** : Defining data types, structures and constraints
  - **Meta data**: the database definition or descriptive information is stored as a database as a database catalog or data dictionary.
- **Constructing the database**: Process of storing the database on some storage medium controlled by DBMS.
- **Manipulating the database** : querying, generating reports, insertions, deletions and modifications to its content
- **Concurrent Processing and Sharing** by a set of users and programs – yet, keeping all data valid and consistent

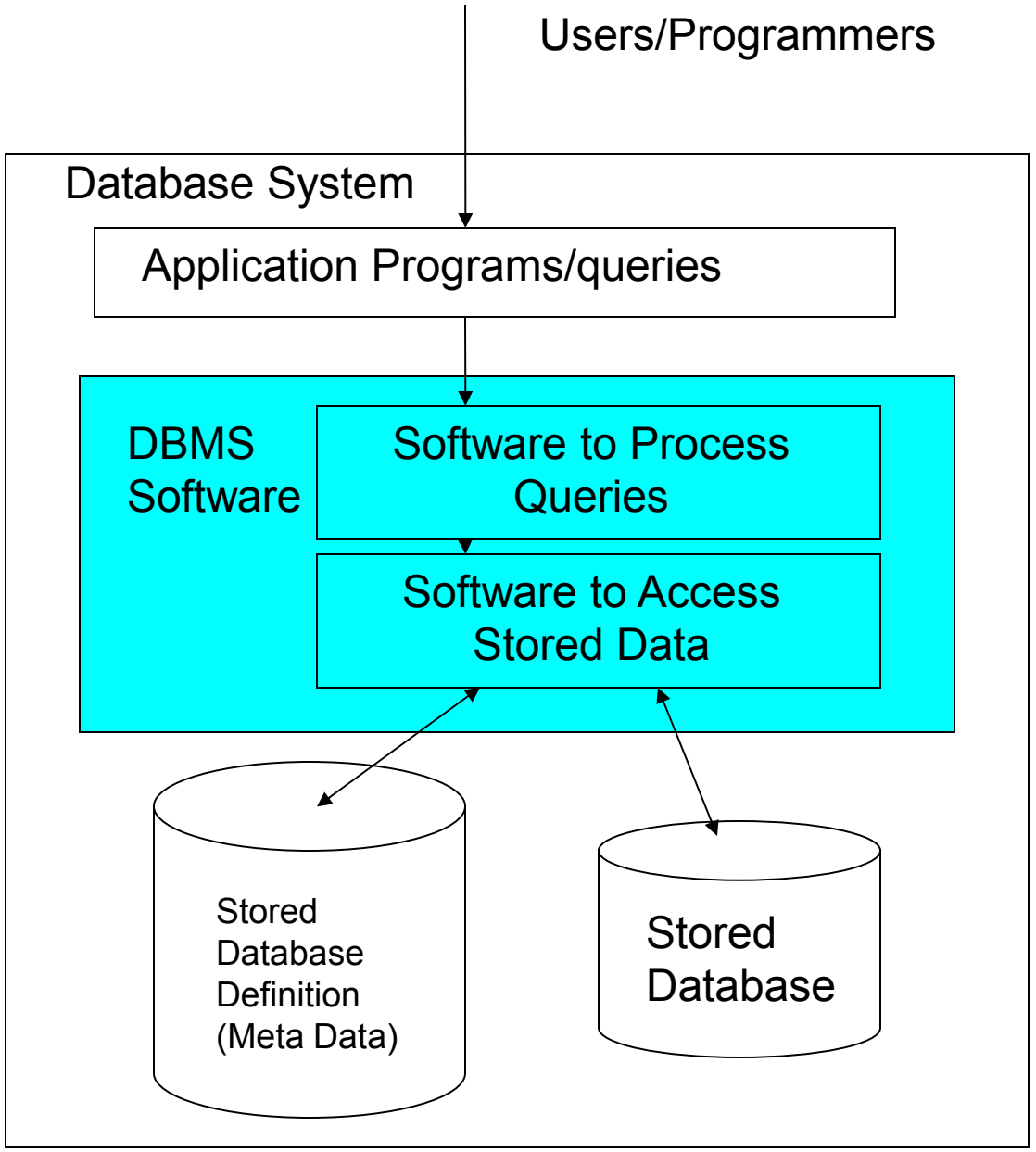
Typical DBMS Functionality

Other features:

- **Protection**
  - Against hardware or software malfunction
- **Security**
  - prevent unauthorized access
- "**Active**" processing to take internal actions on data
- **Presentation** and **Visualization** of data

- **Application Program**: accesses the database by sending queries or requests for data to the DBMS.
  - **Query**: retrieves some data
  - **Transaction**: may retrieve and write some data

A Simplified Database System Environment



Example of a Database

Consider a part of a University environment

We need data about:

- **STUDENTs**
- **COURSEs**
- **SECTIONs (of COURSEs)**
- **(academic) DEPARTMENTs**
- **INSTRUCTORs**

The above data is related as follows:

- **SECTIONs are of specific COURSEs**
- **STUDENTs take SECTIONs**
- **COURSEs have prerequisite COURSEs**
- **INSTRUCTORs teach SECTIONs**
- **COURSEs are offered by DEPARTMENTs**
- **STUDENTs major in DEPARTMENTs**

Example Database

STUDENT			
Name	Student Number	Class	Major
Smith	17	1	COSC
Brown	8	2	COSC

GRADE REPORT		
Student Number	Section-Identifier	Grade
17	85	A
18	102	B+

PREREQUISITE	
Course Number	Prerequisite Number
COSC3380	COSC3320
COSC3320	COSC1310

SECTION				
Section-Identifier	Course Number	Semester	Year	Instructor
85	MATH2410	Fall	91	King
92	COSC1310	Fall	91	Anderson
102	COSC3320	Spring	92	Knuth
135	COSC3380	Fall	92	Stone

COURSE			
Course Name	Course Number	Credit Hours	Department
Intro to CS	COSC1310	4	COSC
Data Structures	COSC3320	4	COSC
Discrete Mathematics	MATH2410	3	MATH
Data Base	COSC3380	3	COSC

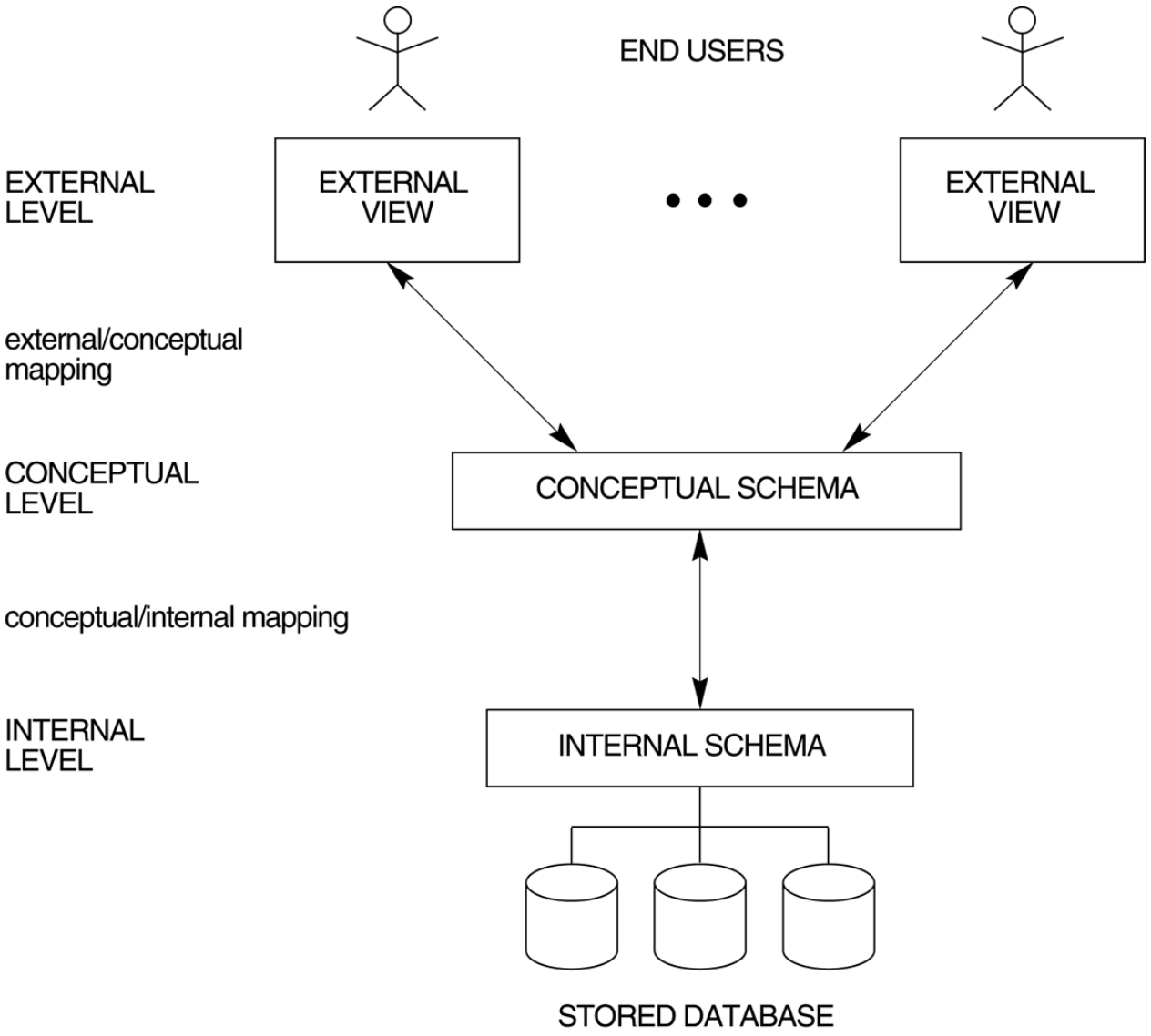
Three-Schema Architecture

- Proposed to support DBMS characteristics of:
  - **Program-data independence**.
  - Support of **multiple views** of the data.

Three-Schema Architecture

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

The three-schema architecture.



Three-Schema Architecture

Mappings among schema levels are needed to transform requests and data. Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.

Data Independence

- **Logical Data Independence**: The capacity to change the conceptual schema without having to change the external schemas and their application programs.
- **Physical Data Independence**: The capacity to change the internal schema without having to change the conceptual schema.

Data Independence

When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence. The higher-level schemas themselves are *unchanged*. Hence, the application programs need not be changed since they refer to the external schemas.

DBMS Languages

- **Data Definition Language** (**DDL**): Used by the DBA and database designers to specify the *conceptual schema* of a database. In many DBMSs, the DDL is also used to define internal and external schemas (views). In some DBMSs, separate **storage definition language** (**SDL**) and **view definition language** (**VDL**) are used to define internal and external schemas.

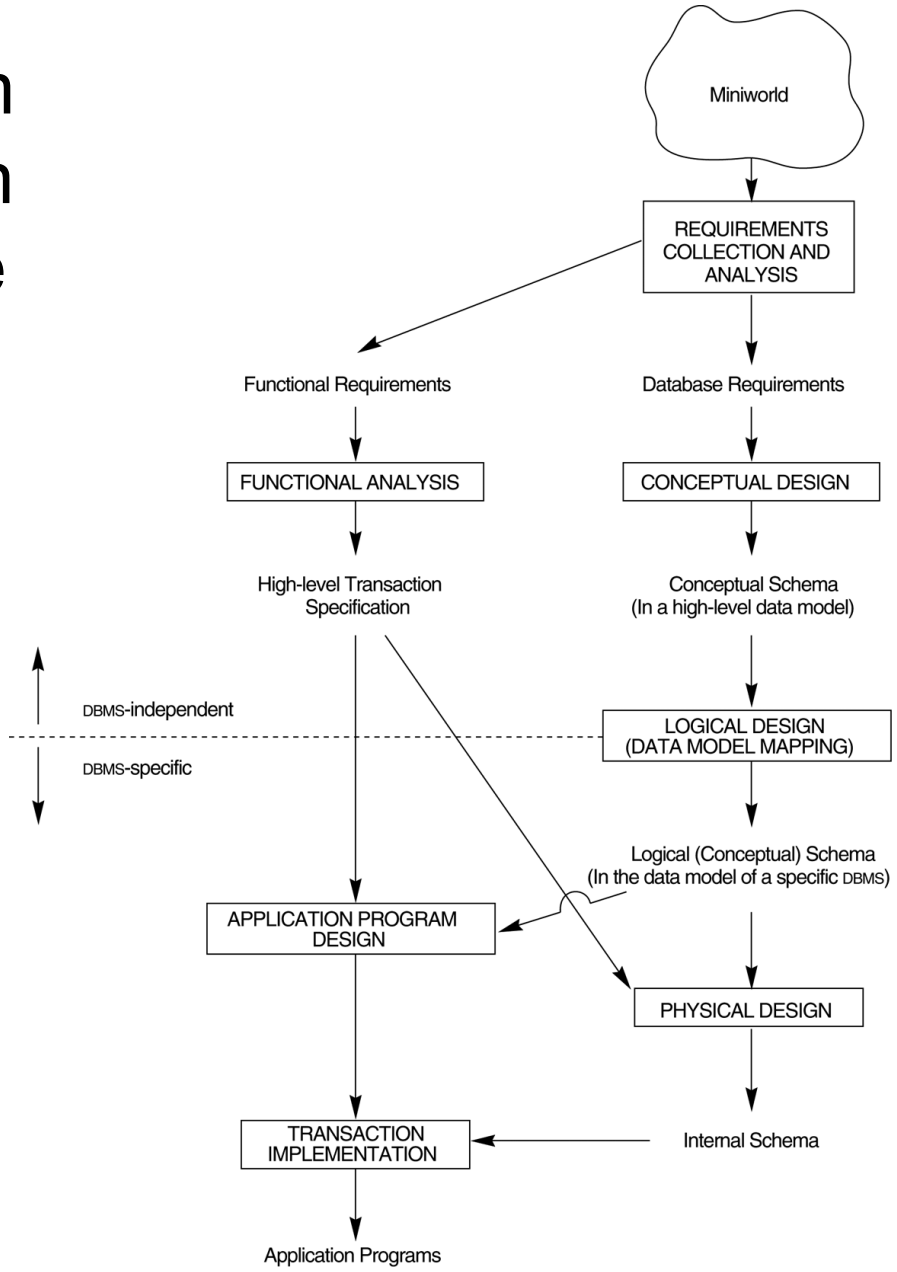
DBMS Languages

- **Data Manipulation Language** (**DML**): Used to specify database retrievals and updates.
  - DML commands (**data sublanguage**) can be *embedded* in a general-purpose programming language (**host language**), such as COBOL, C or an Assembly Language.
  - Alternatively, *stand-alone* DML commands can be applied directly (**query language**).

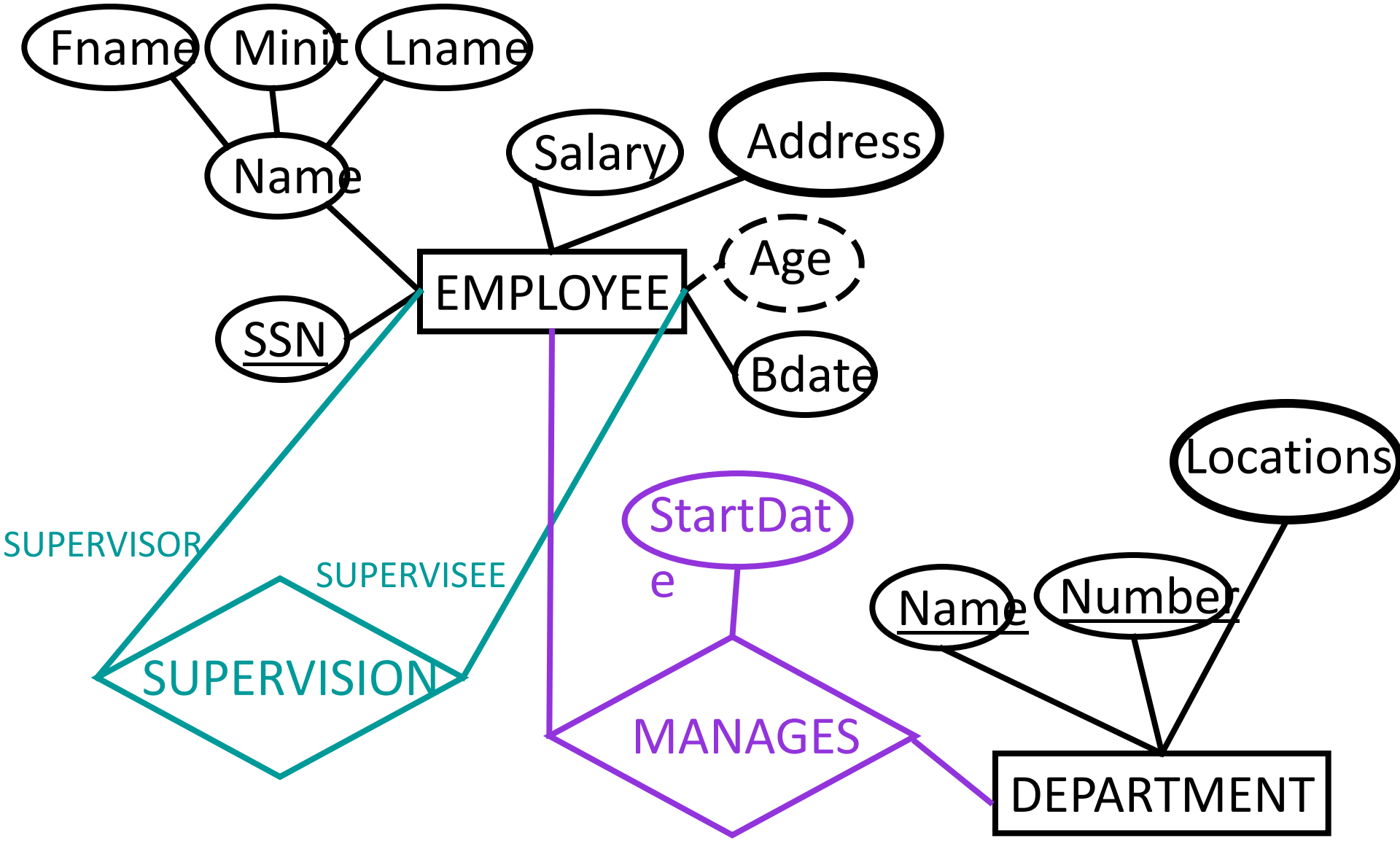
DBMS Languages

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

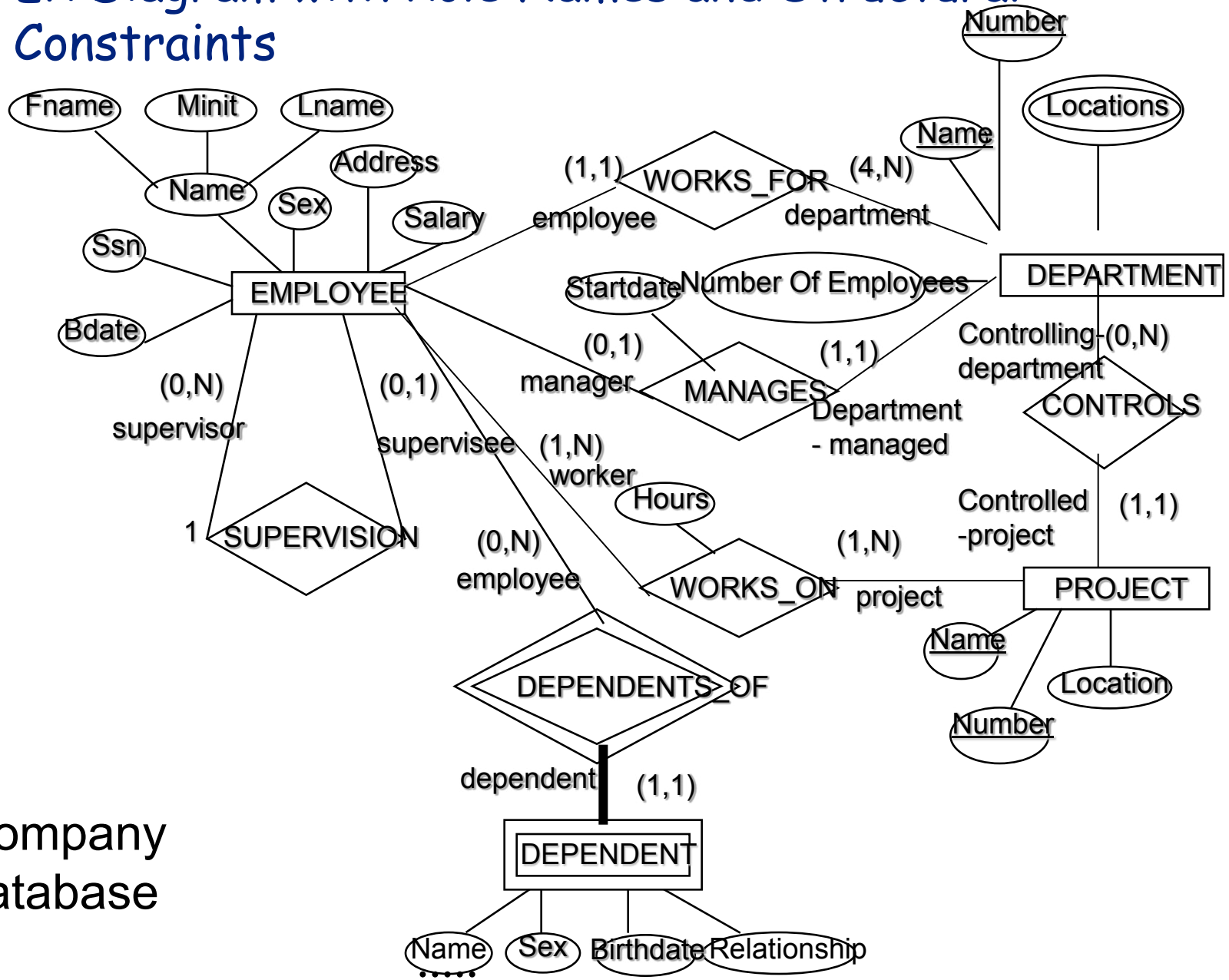
A simplified diagram to illustrate the main phases of database design.



ER Model


Fname
Minit
Lname
Name
Salary
Address
EMPLOYEE
Age
SSN
Bdate
SUPERVISOR
SUPERVISEE
SUPERVISION
StartDat
e
MANAGES
Locations
Name
Number
DEPARTMENT

ER Diagram with Role Names and Structural Constraints



Company database

Database models

- Hierarchical model
- Network model
- **Relational model**
- Object-relational model
- XML model

Relational Model Concepts

Relation (informally): A table of values. Each column in the table has a column header called an attribute. Each row is called a tuple.

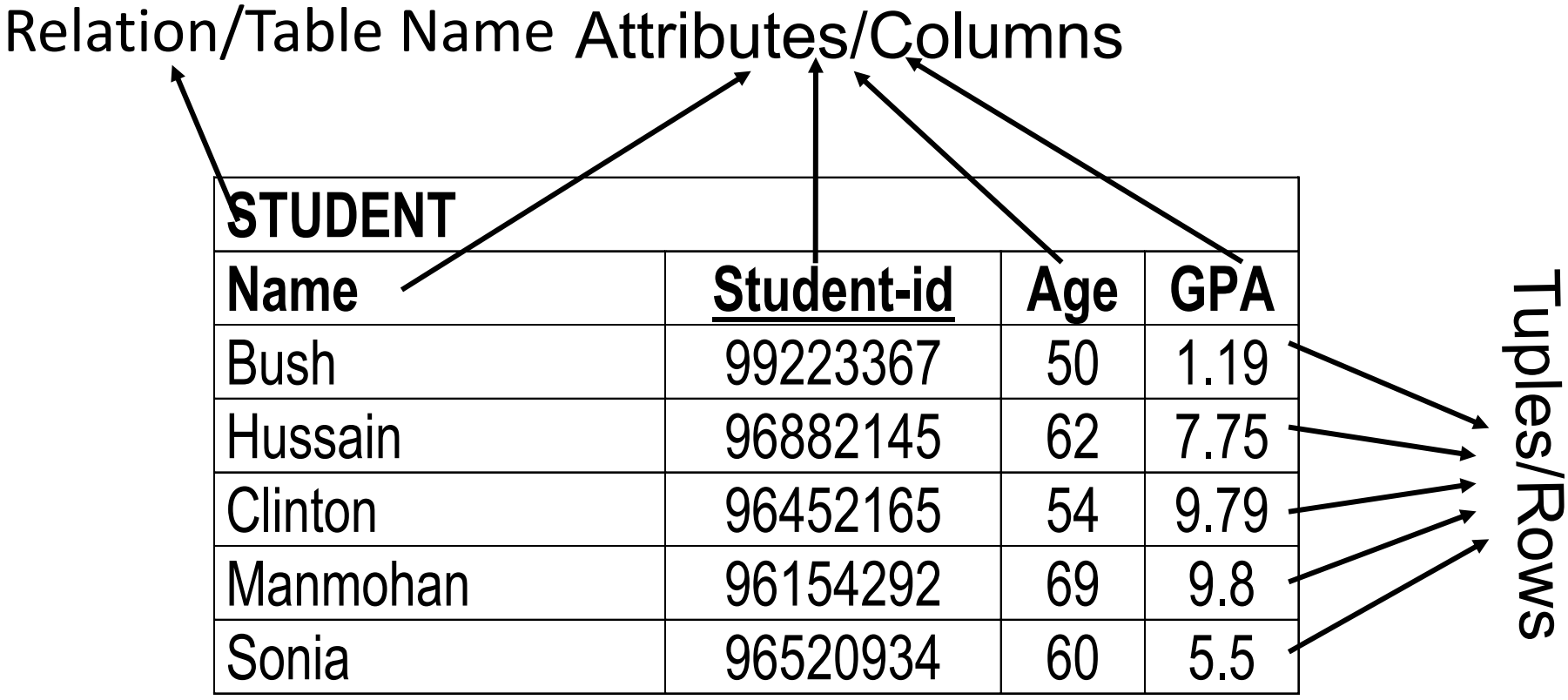
Formal Relational Concepts

- Domain: A set of atomic (indivisible) values.
- Attribute: A name to suggest the meaning that a domain plays in a particular relation. Each attribute A_i has a domain $dom(A_i)$.
- Relational Schema: A relation name R and a set of attributes A_i that define the relation.

Denoted by: $R(A_1, A_2, ..., A_n)$

Example: STUDENT(Name, Student-id, Age,GPA)

Example: Student Relation



STUDENT			
Name	**Student-id**	**Age**	**GPA**
Bush	99223367	50	1.19
Hussain	96882145	62	7.75
Clinton	96452165	54	9.79
Manmohan	96154292	69	9.8
Sonia	96520934	60	5.5

Integrity Constraints

Constraints are conditions that must hold on all valid relation instances. These constraints are Domain constraints, Key constraints, Entity integrity constraints, and Referential integrity constraints

Domain Constraints

Each attribute A must be an atomic value from the domain dom(A) for that attribute. The standard types of domain include integers, real numbers, characters, fixed length strings.

Relational Algebra

For those students who received Grade ‘A’ in a course, get the student name and course name

Answer:

Student name is in relation STUDENT, Course name is in relation COURSE, while grade is in relation GRADE-REPORT. For a student to retrieve the Course name, one has to join STUDENT with GRADE-REPORT to get Section number, and then with SECTION to get the course number, and then from COURSE the Course name. Since we want only those students who got grade ‘A’, a selection operation on relation GRADE-REPORT is needed. Since we want only student Name and Course Name (and not all attributes), a project operation is required.

$$\Pi_{\text{Name, Course Name}}(\sigma_{\text{Grade = `A'}}(\text{STUDENT}*\text{GRADE-REPORT}$$

$$*\text{SECTION}* \text{ COURSE}))$$

Summary of SQL Queries

A query in SQL can consist of up to six clauses, but only the first two are mandatory

```
SELECT      <ATTRIBUTE LIST>
FROM        <TABLE LIST>
[WHERE      <CONDITION>]
[GROUP BY   <GROUPING ATTRIBUTES>]
[HAVING     <GROUP CONDITION>]
[ORDER BY   <ATTRIBUTE LIST>]
```

A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause

UPDATE

Give all employees in the 'Research' department a 10% raise in salary.

```
UPDATE EMPLOYEE
SET SALARY = SALARY *1.1
WHERE DNO IN (SELECT DNUMBER
                     FROM    DEPARTMENT
                     WHERE DNAME='Research');
```

In this request, the modified SALARY value depends on the original SALARY value in each tuple. The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification. The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification.

Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a *key.*

Functional Dependencies

- Let *R* be a relation schema

  $\alpha \subseteq R$ *and* $\beta \subseteq R$

- The functional dependency

  $\alpha \rightarrow \beta$

  holds on *R* if and only if for any legal relations *r*(R), whenever any two tuples t_1 and t_2 of *r* agree on the attributes α, they also agree on the attributes β. That is,

  $t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$

- Example: Consider *r*(A,*B*) with the following instance of *r.*

1	4
1	5
3	7

- On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold.

Inference Rules for FDs

- F denotes the set of FDs specified for R
- Other FDs may also hold on all legal instances that satisfy F
- The set of all such FDs is called closure of F denoted by F^+

- A FD $X \rightarrow Y$ is inferred from F if $X \rightarrow Y$ holds in every legal instance r of R
  - ☞ need a set of inference rules that can systematically infer new FDs from a given set of FDs

☞ Reflexitivity, Augmentation, Transitivity

- Notation:

  $F \models X \rightarrow Y$ (F infers $X \rightarrow Y$); $\{X,Y\} \rightarrow Z$ written as $XY \rightarrow Z$;

  and $\{X,Y,X\} \rightarrow \{UV\}$ written as $XYZ \rightarrow UV$

Normalization

- A systematic process of decomposing unsatisfactory relation schemas into smaller relations schemas that possess desirable properties.

- Provides a series of tests for relation schemas
- expressed in terms of normal forms
  - ☞ first four: only use FDs
  - ☞ additional: use other types of dependencies

- Not necessary to normalize to highest form!

- Normal forms do not guarantee good design
  - ☞ also need to consider
    - ◆ loss less join property
    - ◆ dependency preserving property

Normalization

1 NF

2NF

3NF

BCNF

Summary

Normal Form	Test	Normalization (Remedy)
First (1NF)	Relation should have no non-atomic attributes or nested relations.	Form new relations for each non-atomic attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no non-key attributes should be functionally dependent on a part of primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a non-key attribute functionally determined by another non-key attribute (or by a set of non-key attributes.) That is, there should be no transitive dependency of a non-key attribute on the primary key.	Decompose and set up a relation that includes the non key attribute(s) that functionally determine(s) other non-key attribute(s).

Algorithm for computing X^+

Algorithm: X^+ (closure of X under F)

$X^+ := X;$

while (changes to X^+) do

for each FD $Y \rightarrow Z$

begin

if $Y \subseteq X^+$; then $X^+ = X^+ \cup Z$

end

Lossless Join and Dependency Preserving 3NF algorithm

1. Find a minimal set (cover) of FDs G equivalent to F
2. For each X of an FD $X \rightarrow A$ in G

   Create a relation schema R_i in D with the attributes $\{X \cup A_1 \cup A_2 \cup \ldots \cup A_k\}$ where the A_j's are all the attributes appearing in an FD in G with X as left hand side
3. If any attributes in R are not placed in any R_i, create another relation in D for these attributes
4 If none of them contain a candidate key K or R, create one more relation schema that contains attributes in K.

   ☞ Step 2 assures dependency preserving property

   ☞ Step 4 assures lossless join property

Lossless Join Decomposition into BCNF

1. set D:={R}
2. While there is a relation schema Q in D that is not in BCNF do
    - choose a Q in D that is not in BCNF
    - find a FD $X \rightarrow Y$ that violates BCNF
    - replace Q in D by two relation schemas (Q - Y) and ($X \cup Y$)