DSA5104

Principles of Data Management and Retrieval

Lecture 1: Introduction to Database Systems and Relational Model

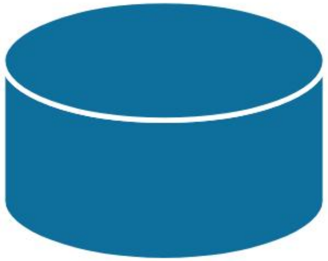
01

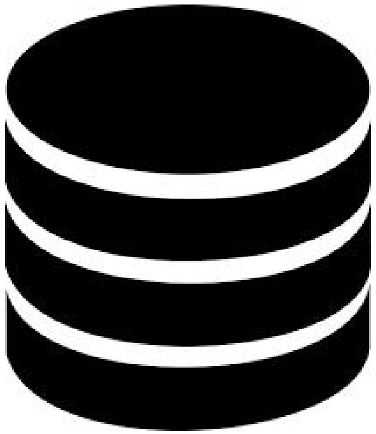
PART ONE

Databases

What is Data?

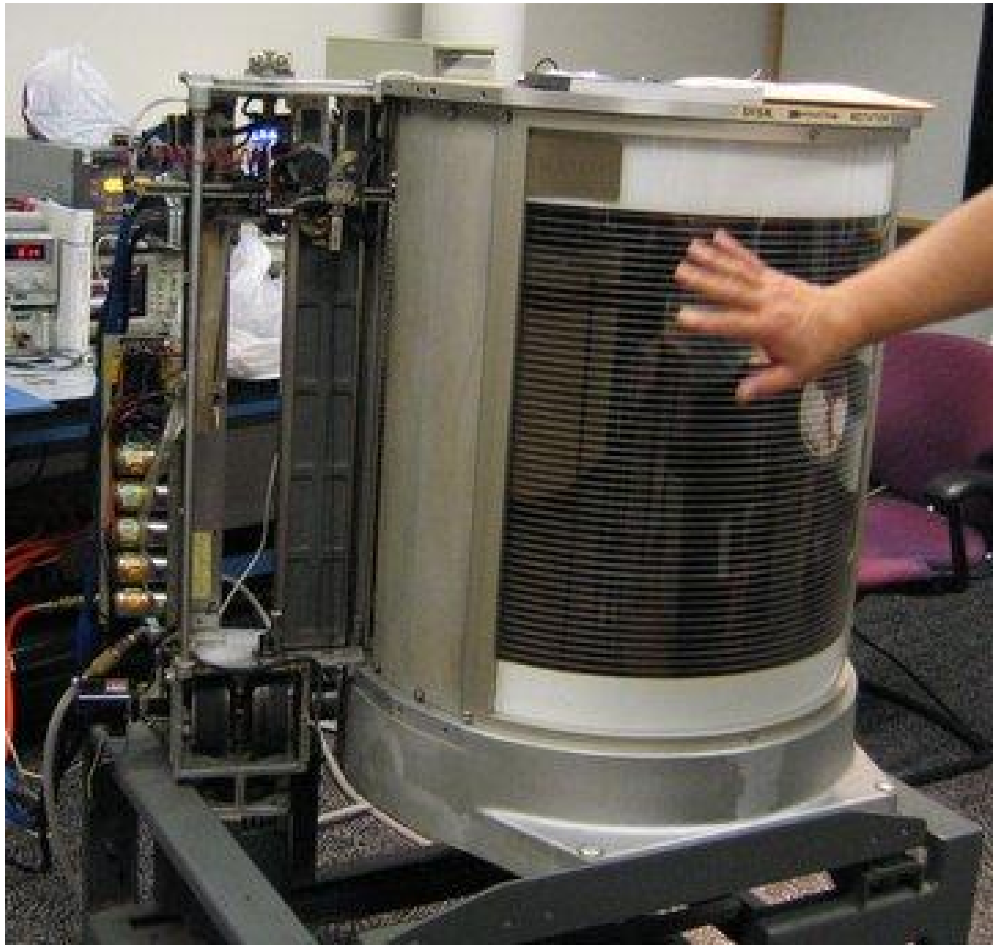
Universal Symbol for a Database?





Why the Symbol?

Looks Like?



Partially disassembled IBM 350 (RAMAC), 1954

First commercial disk drive - IBM Model 350
3.75 MB @ 1 ton

https://en.wikipedia.org/wiki/Hard_disk_drive

What is a Database?

- Data
  - *Digital data*
- Database
  - *A large, orgainzed collection of data*
- Database Management Systems (DBMS)
  - *A DBMS is a* ***software*** *that* ***stores****,* ***manages*** *and facilitates* ***access*** *to data.*

Outline

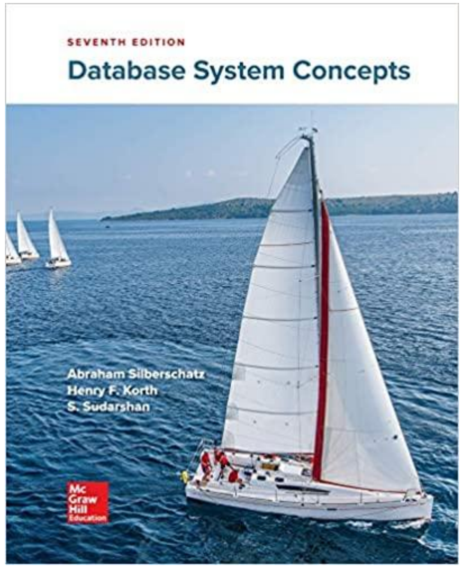
- Database-System Applications
- Purpose of Database Systems
- View of Data
- Database Languages
- Database Design
- Database Engine
- Database Architecture
- Database Users
- History of Database Systems

Database Systems

- Database Management System (DBMS) contains information about a particular enterprise
  - Collection of interrelated data
  - Set of programs to access the data
  - An environment that is both *convenient* and *efficient* to use

- Database systems are used to manage collections of data that are:
  - Highly valuable
  - Relatively large
  - Accessed by multiple users and applications, often at the same time.

- Databases touch all aspects of our lives



Database System Concepts - 7th Edition

Databases are Ubiquitous

- Data processing backs essential every app
- Database systems of one form or another back most apps

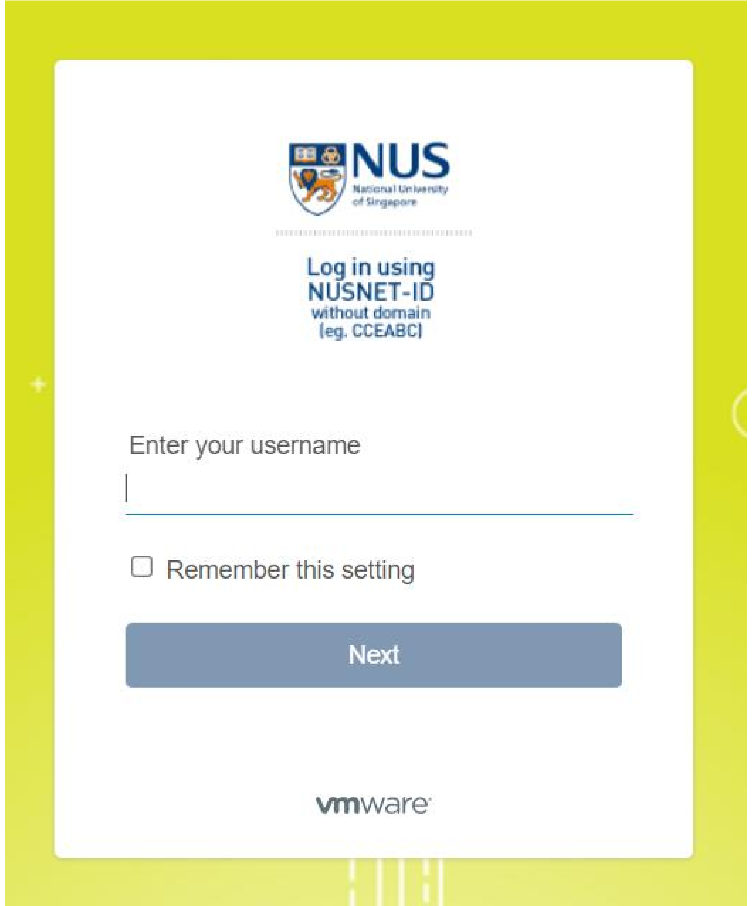
Database Applications Examples

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

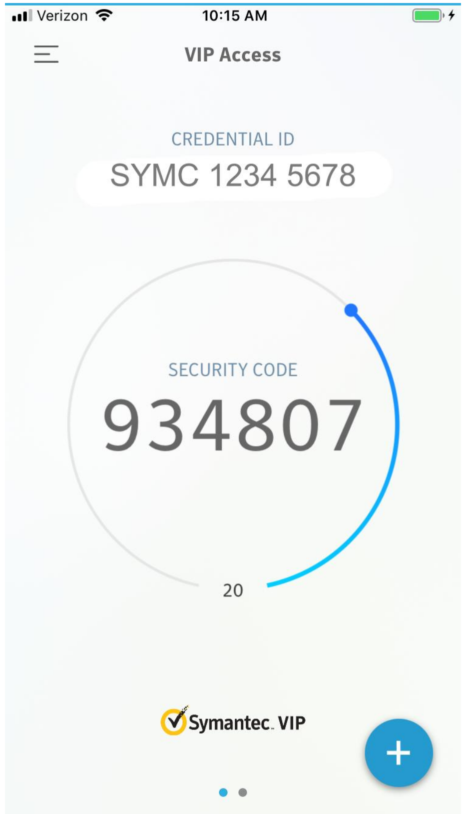
Database Applications Examples (Cont.)

- Airlines: reservations, schedules
- Telecommunication: records of calls, texts, and data usage, generating monthly bills, maintaining balances on prepaid calling cards
- Web-based services
  - Online retailers: order tracking, customized recommendations
  - Online advertisements
- Navigation systems: For maintaining the locations of varies places of interest along with the exact routes of roads, train systems, buses, etc.

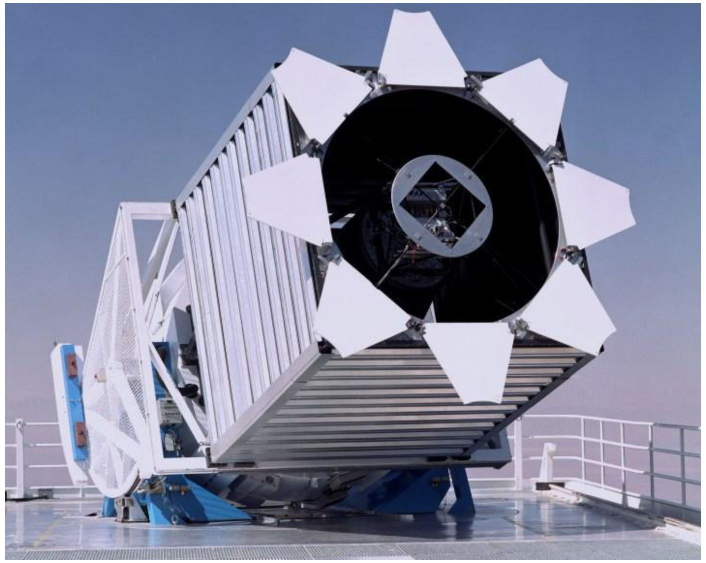
University - Lecturer Information



- Personal information
  - Name
  - Address
  - Alternate Email address
  - Mobile No.
  - …
- To create an NUS Email account
  - Account ID / Username / Password
  - Mobile No
  - Creditial ID (for 2FA)
  - Purpose / Course



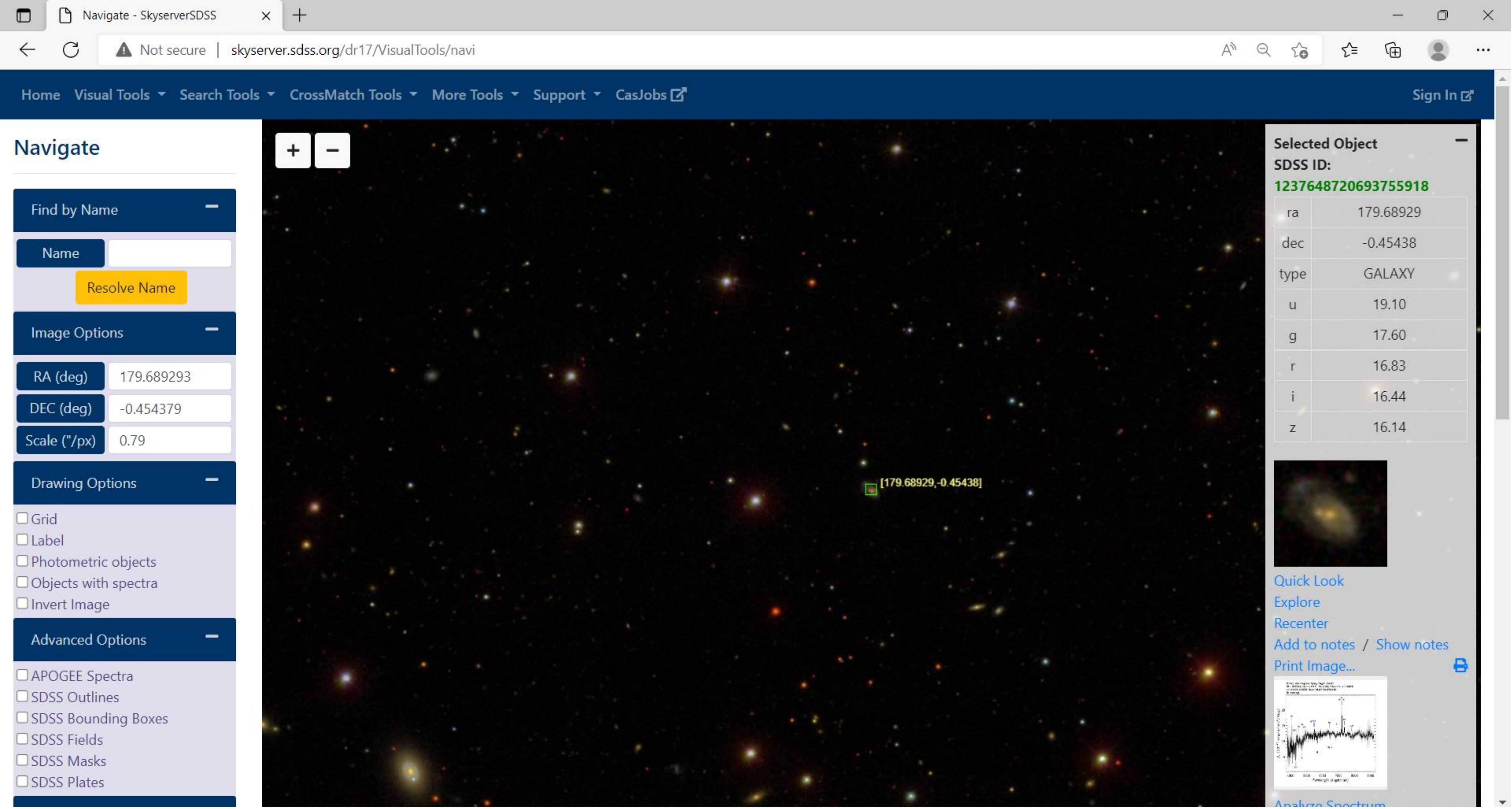
Science - SkyServer



Sloan Digital Sky Survey
(SDSS)

Database
Systems

SkyServer



http://skyserver.sdss.org/dr17/VisualTools/navi

Science - SciServer

SciServer

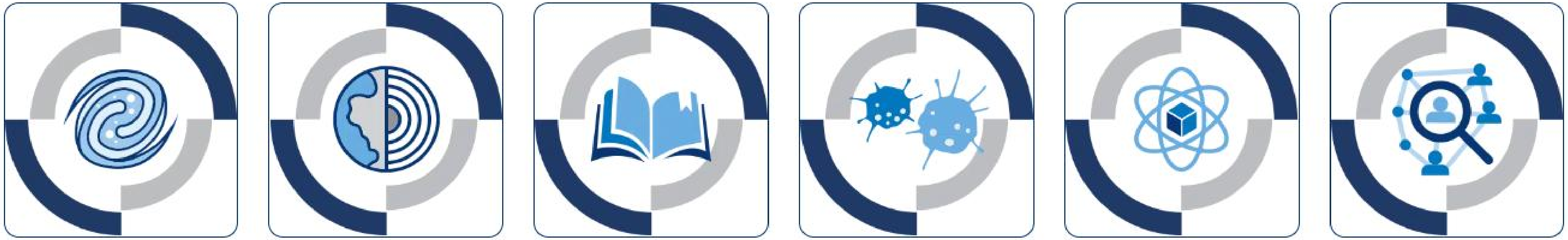
Collaborative data-driven science

SciServer consists of data hosting services coupled with integrated tools that work together to create a full-featured system.

SciServer is a fully integrated cyberinfrastructure system encompassing related tools and services to enable researchers to cope with scientific big data. SciServer enables a new approach that will allow researchers to work with Terabytes or Petabytes of scientific data, without needing to download any large datasets.

SciServer is a revolutionary new approach to achieving productive science research by **bringing the analysis to the data**.

Astronomy | Earth Sciences | Education | Life Sciences | Materials Science | Social Sciences



https://idies.jhu.edu/what-we-offer/sciserver/

Purpose of Database Systems

In the early days, database applications were built directly on top of file systems, which leads to:

- Data redundancy and inconsistency: data is stored in multiple file formats resulting in duplication of information in different files
- Difficulty in accessing data
  - Need to write a new program to carry out each new task
- Data isolation
  - Multiple files and formats

Purpose of Database Systems (Cont.)

- Integrity problems
  - Integrity constraints (e.g., account balance > 0) become "buried" in program code rather than being stated explicitly
  - Hard to add new constraints or change existing ones
- Atomicity of updates
  - Failures may leave database in an inconsistent state with partial updates carried out
  - Example: Transfer of funds from one account to another should either complete or not happen at all

Purpose of Database Systems (Cont.)

- Concurrent access by multiple users
  - Concurrent access needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies
    - Ex: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time
- Security problems
  - Hard to provide user access to some, but not all, data

Database systems offer solutions to all the above problems

Example: University Database

- Data consists of information about:
  - Students
  - Instructors
  - Courses/Classes
- Application program examples:
  - Add new students, instructors, and courses
  - Register students for courses, and generate class rosters
  - Assign grades to students, compute grade point averages (GPA) and generate transcripts

Workload

01 Homework

30%

- Done individually
- Week 3 (15%)
- Week 7 (15%)

02 Project

35%

- 3-4 persons
- Starting from week 5

03 Quiz

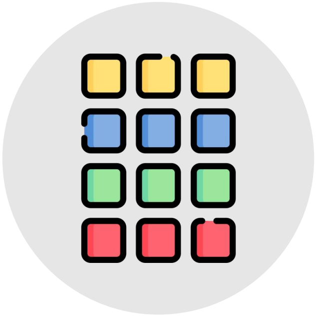
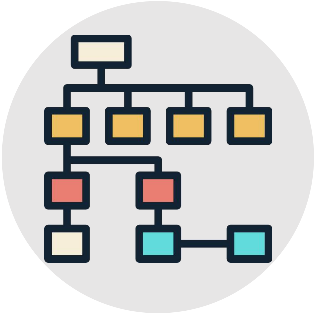
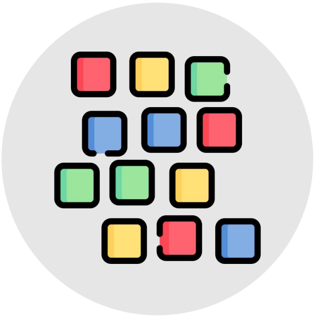
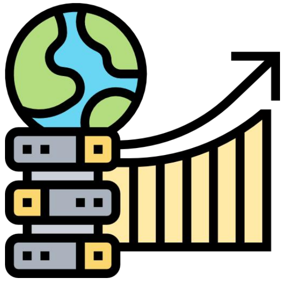
30%

- 3 Quizzes
- Considering the best 2 scores (each 15%)
- Week 5, 8, 11

04 Course Participation

5%

Recap

	Structured	**Semi-Structured**	**Unstructured**	**Big Data**
Definition	• Data with predefined schema	• Data with flexible schema (e.g., XML)	• Data without predefined schema	
Data Management	• Relational Database	• MongoDB/HBase	• No-SQL Databases	• ETL
Data Retrieval	• SQL	• XPath • XQuery	• ElasticSearch for text	• Spark SQL

Outline

- Database-System Applications
- Purpose of Database Systems
- View of Data
- Database Languages
- Database Design
- Database Engine
- Database Architecture
- Database Users
- History of Database Systems

View of Data

- A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data.
- A major purpose of a database system is to provide users with an **abstract** view of the data.
  - Data abstraction
    - Hide the complexity of data structures to represent data in the database from users through several levels of data abstraction.
  - Data models
    - A collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

Data Models

- A collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semi-structured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model

Relational Model

- All the data is stored in various tables.
- Example of tabular data in the relational model

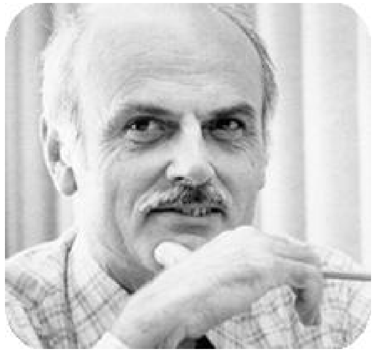
Relation - A table with rows and columns

Column

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

Row

(a) The *instructor* table



Ted Codd
Turing Award 1981

University Database: A Sample Relational DB

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

(a) The *instructor* table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

Levels of Abstraction

- **Physical level**: describes how a record (e.g., instructor) is stored.
- **Logical level**: describes what data are stored in database, and the relationships among the data.

  type *instructor* = record

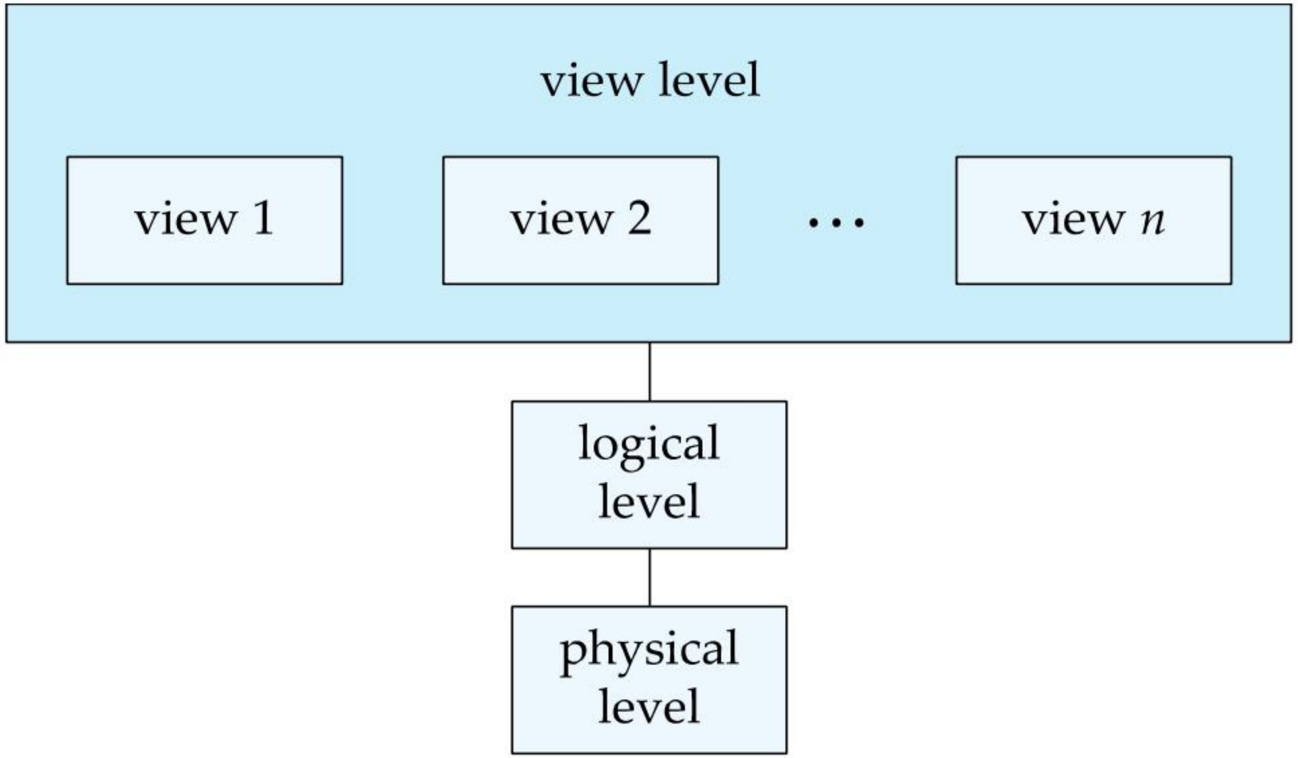
  ID : string;
  name : string;
  dept_name : string;
  salary : integer;

  end;

- **View level**: describes only part of the entire database. Application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.

View of Data

- An architecture for a database system



Application: Course Registration

Type/Table definition: instructor

Block of consecutive bytes

Instances and Schemas

- Similar to types and variables in programming languages
- Logical Schema – the overall logical structure of the database
  - Example: The database consists of information about a set of customers and accounts in a bank and the relationship between them
    - Analogous to type information of a variable in a program
- Physical schema – the overall physical structure of the database
- Instance – the actual content of the database at a particular point in time
  - Analogous to the value of a variable

Physical Data Independence

- Physical Data Independence – the ability to modify the physical schema without changing the logical schema
  - Applications depend on the logical schema
  - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

Database Languages - Data Definition Language (DDL)

- Specification notation for defining the database schema

  Example:

```
create table instructor (
    ID          char(5),
    name        varchar(20),
    dept_name   varchar(20),
    salary      numeric(8,2))
```

- DDL compiler generates a set of table templates stored in a *data dictionary*
- Data dictionary contains metadata (i.e., data about data)
  - Database schema
  - Integrity constraints
    - Primary key (ID uniquely identifies instructors)
  - Authorization
    - Who can access what

Database Languages - Data Manipulation Language (DML)

- Language for accessing and updating the data organized by the appropriate data model

- There are basically two types of data-manipulation language
  - Procedural DML - require a user to specify what data are needed and how to get those data.
  - Declarative DML - require a user to specify what data are needed without specifying how to get those data.

Query Language

- Query - a statement requesting the retrieval of information.

- Query Language - the portion of a DML that involves information retrieval
  - DML also known as query language

SQL Query Language

- SQL query language is nonprocedural.
- Example to find all instructors in Comp. Sci. dept

    select *name*
    from *instructor*
    where *dept_name* = 'Comp. Sci.'

- SQL is NOT a Turing machine equivalent language
    - To be able to compute complex functions SQL is usually embedded in some higher-level language
- Application programs generally **access** databases through
    - Language extensions to allow embedded SQL
    - Application program interface (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database

Database Design

The process of designing the general structure of the database:

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

Database Engine

- A database system is partitioned into modules that deal with each of the responsibilities of the overall system.

- The functional components of a database system can be divided into
  - The storage manager,
  - The query processor component,
  - The transaction management component.

Storage Manager

- A program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible to the following tasks:
  - Interaction with the OS file manager
  - Efficient storing, retrieving and updating of data
- The storage manager components include:
  - Authorization and integrity manager
  - Transaction manager
  - File manager
  - Buffer manager

Storage Manager (Cont.)

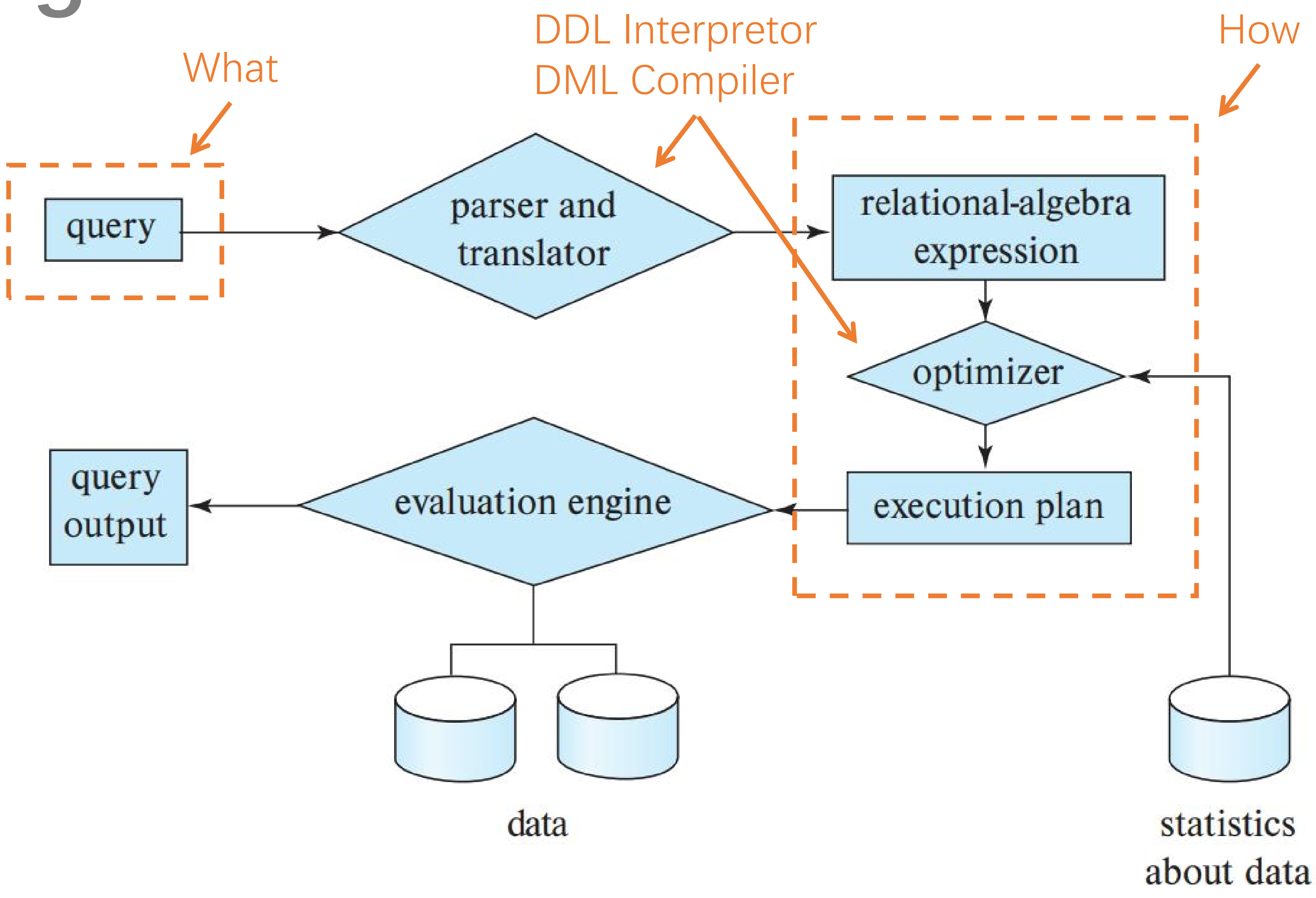
- The storage manager implements several data structures as part of the physical system implementation:
  - Data files - store the database itself
  - Data dictionary - stores metadata about the structure of the database, in particular the schema of the database.
  - Indices - can provide fast access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

Query Processor

- The query processor components include:
  - **DDL interpreter** - interprets DDL statements and records the definitions in the data dictionary.
  - **DML compiler** - translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
    - The DML compiler performs **query optimization**; that is, it picks the lowest cost evaluation plan from among the various alternatives.
  - **Query evaluation engine** - executes low-level instructions generated by the DML compiler.

Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Transaction Management

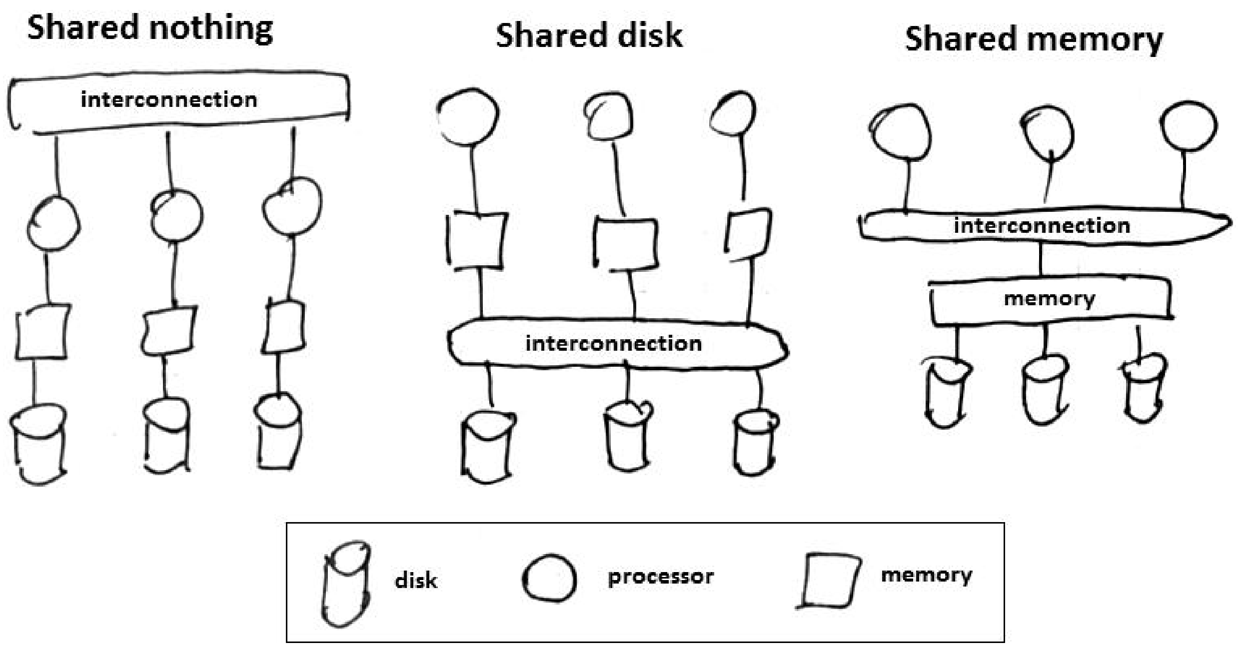
- A transaction is a collection of operations that performs a single logical function in a database application
  - ACID - Atomicity, Consistency, Isolation, Durability

- Transaction-management component ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.

- Concurrency-control manager controls the interaction among the concurrent transactions, to ensure the consistency of the database.

Database Architecture

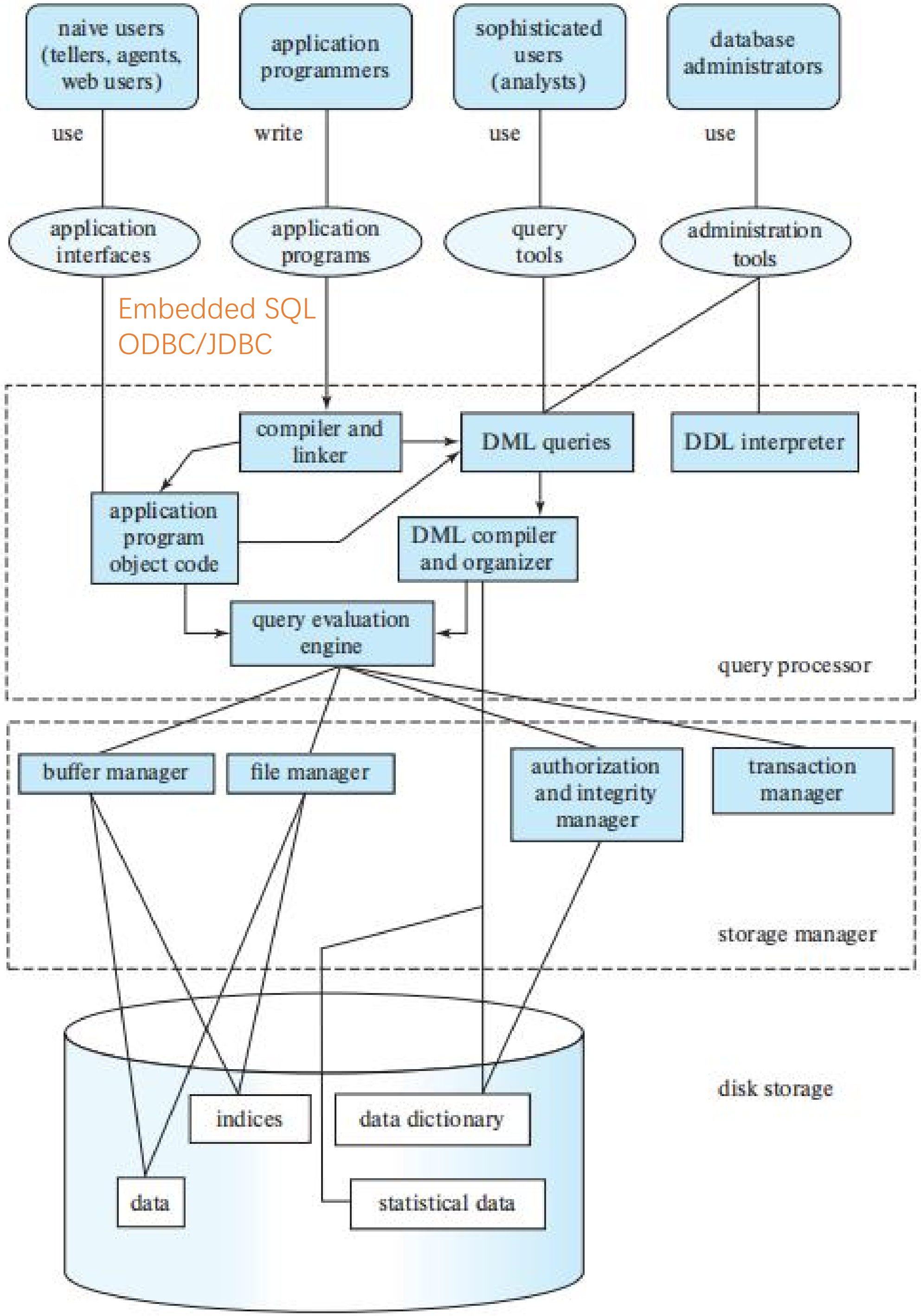
- Centralized databases
  - One to a few cores, shared memory
- Parallel databases
  - Shared memory (Shared everything)
  - Shared disk
  - Shared nothing
- Distributed databases
  - Geographical distribution
  - Schema/data heterogeneity



https://raw.github.com/alexeygrigorev/ulb-adb-project-couchbd/master/report/images/parallel-arhitectures.png

Database Architecture

Centralized



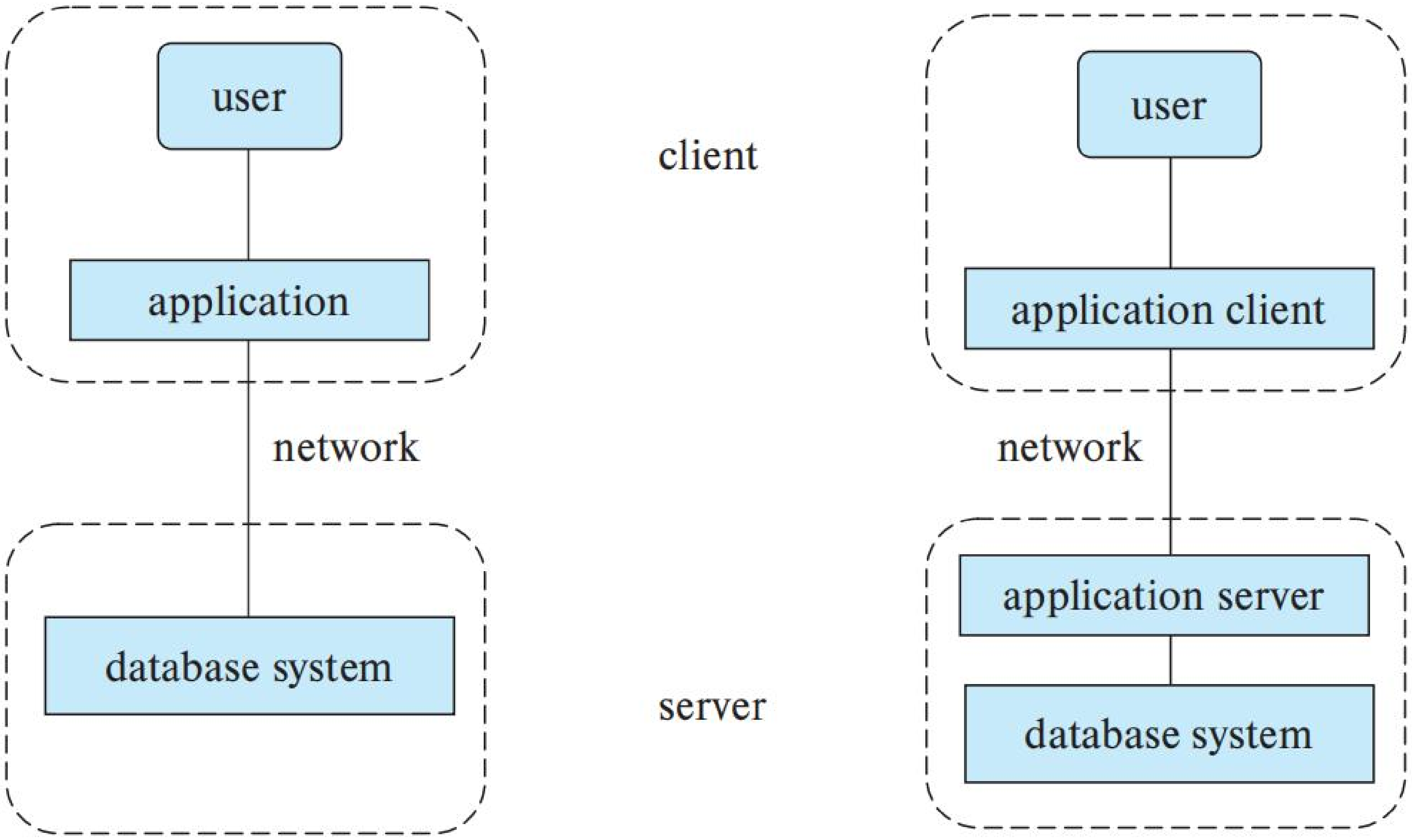
Database Application Architecture

Database applications are usually partitioned into two or three parts:

- Two-tier architecture - the application resides at the client machine, where it invokes database system functionality at the server machine

- Three-tier architecture - the client machine acts as a front end and does not contain any direct database calls.
  - The client end communicates with an application server, usually through a forms interface.
  - The application server in turn communicates with a database system to access data.
  - Better performance and security

Two-tier and Three-tier Architectures



(a) Two-tier architecture (b) Three-tier architecture

Database Users

Three different types of database-system users:

- Naive users -- unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.

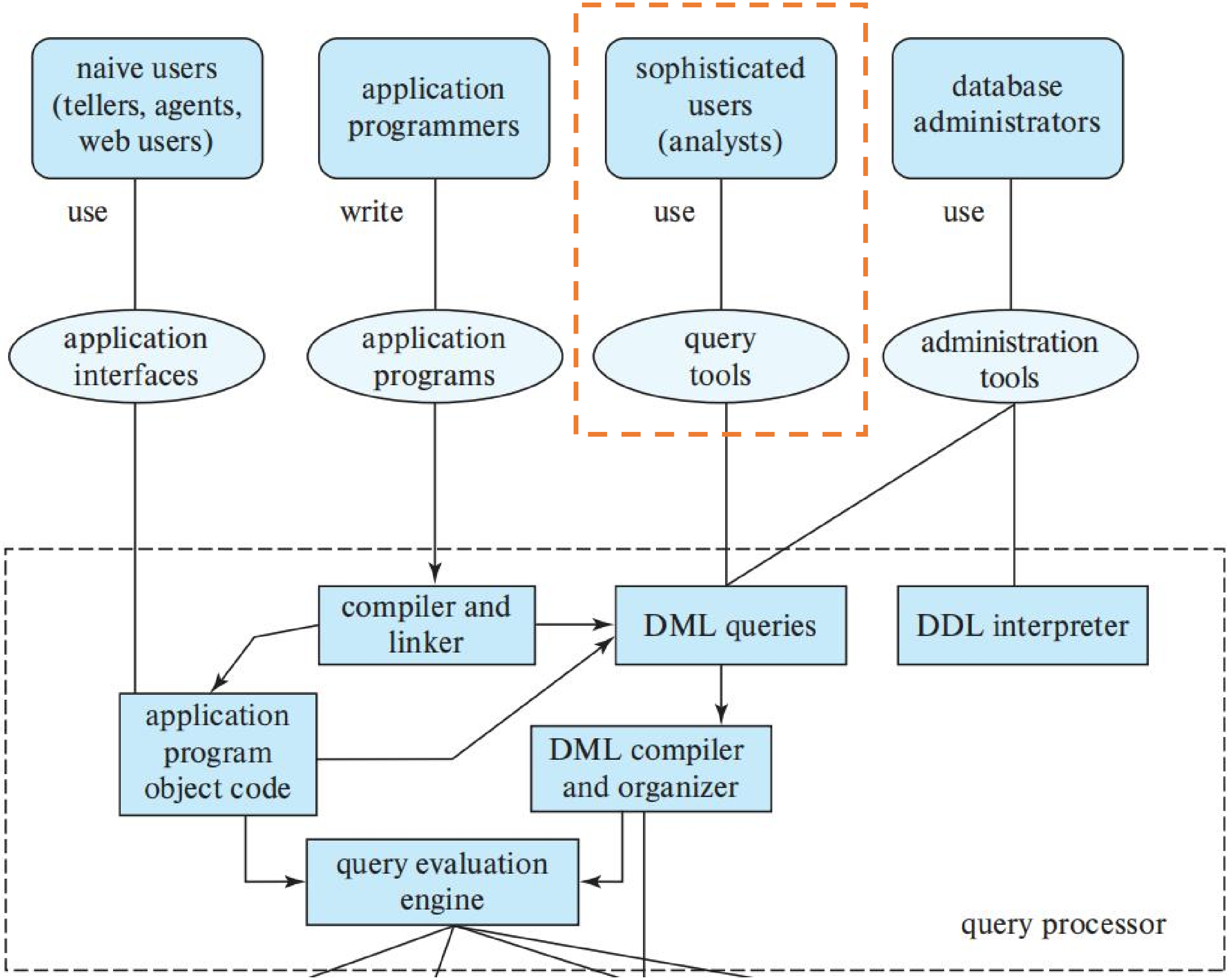
  View Level

- Application programmers - are computer professionals who write application programs.

  Logical Level

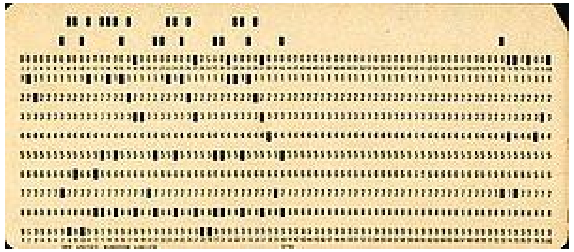
- Sophisticated users - interact with the system without writing programs
  - Using a database query language
  - Using tools such as data analysis software.

Database Users (Cont.)

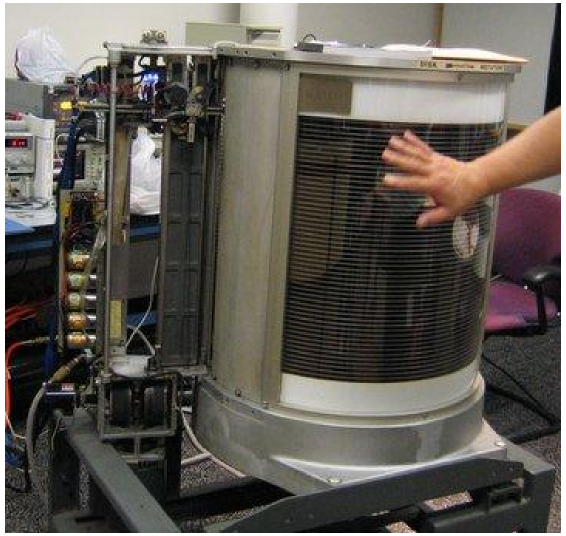


History of Database Systems

- 1950s and early 1960s:
  - Data processing using magnetic tapes for storage
    - Tapes provided only sequential access
  - Punched cards for input



- Late 1960s and 1970s:
  - Hard disks allowed direct access to data
  - Network and hierarchical data models in widespread use
  - Ted Codd defines the relational data model
    - Would win the ACM Turing Award for this work (1981)
    - IBM Research begins System R prototype
    - UC Berkeley (Michael Stonebraker) begins Ingres prototype
    - Oracle releases first commercial relational database



First commercial disk drive
IBM Model 350

History of Database Systems (Cont.)

- 1980s:
  - Research relational prototypes evolve into commercial systems
    - SQL becomes industrial standard
  - Parallel and distributed database systems
    - Wisconsin, IBM, Teradata
  - Object-oriented database systems

- 1990s:
  - Large decision support and data-mining applications
  - Large multi-terabyte data warehouses
  - Emergence of Web commerce

History of Database Systems (Cont.)

- 2000s
  - Big data storage systems
    - Google BigTable, Yahoo PNuts, Amazon,
    - “NoSQL” systems.
  - Big data analysis: beyond SQL
    - Map reduce and friends
- 2010s
  - SQL reloaded
    - SQL front end to Map Reduce systems (e.g., Apache Hive, Apache Phoenix)
    - Massively parallel database systems
    - Multi-core main-memory databases

Review Terms

- Database-management system (DBMS)
- Data abstraction
  - Physical level
  - Logical level
  - View level
- Instance
- Schema
- Physical data independence
- Data models
- Database languages
  - Data-definition language (DDL)
  - Data-manipulation language (DML)
  - Query language
- Database Engine
  - Storage manager
  - Query processor
- Transactions
  - ACID
- Database/Application Architecture

PART TWO

02

Relational Model

Outline

- Structure of Relational Databases
- Database Schema
- Keys
- Schema Diagrams
- Relational Query Languages
- The Relational Algebra

Example of a Instructor Relation

Attributes (or columns)

- ID
- name
- dept_name
- salary

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

(a) The *instructor* table

Tuples
(or rows)

Instance

Relation Schema and Instance

- $A1, A2, \cdots, An$ are attributes

- $R = (A1, A2, \cdots, An)$ is a relation schema

  Example:

  instructor = (ID, name, dept_name, salary)

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

Terms in the Relational Model

- **Relation** vs. **Table**
  - The current values of a **relation** are specified by a **table**

- **Tuple** vs. **Row**
  - Tuple - an element *t* of relation *r*
  - Tuple is represented by a **row** in a table

- **Attribute** vs. **Column**
  - Attribute refers to a column of a table

Attributes

- Domain - the set of allowed values for each attribute is called the *domain* of the attribute
  - Attribute values (domain) are (normally) required to be **atomic**; that is, indivisible
  - E.g., attribute *phone_number* storing a set of phone numbers corresponding to the instructor is not atomic

- The special value *null* is a member of every domain.
  - Indicating that the value is "unknown" or does not exist
  - The null value causes complications in the definition of many operations

Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: instructor relation with unordered tuples

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

Database Schema & Instance

- Database schema - is the logical structure of the database.
- Database instance - is a snapshot of the data in the database at a given instant in time.
- Example:
  - schema:

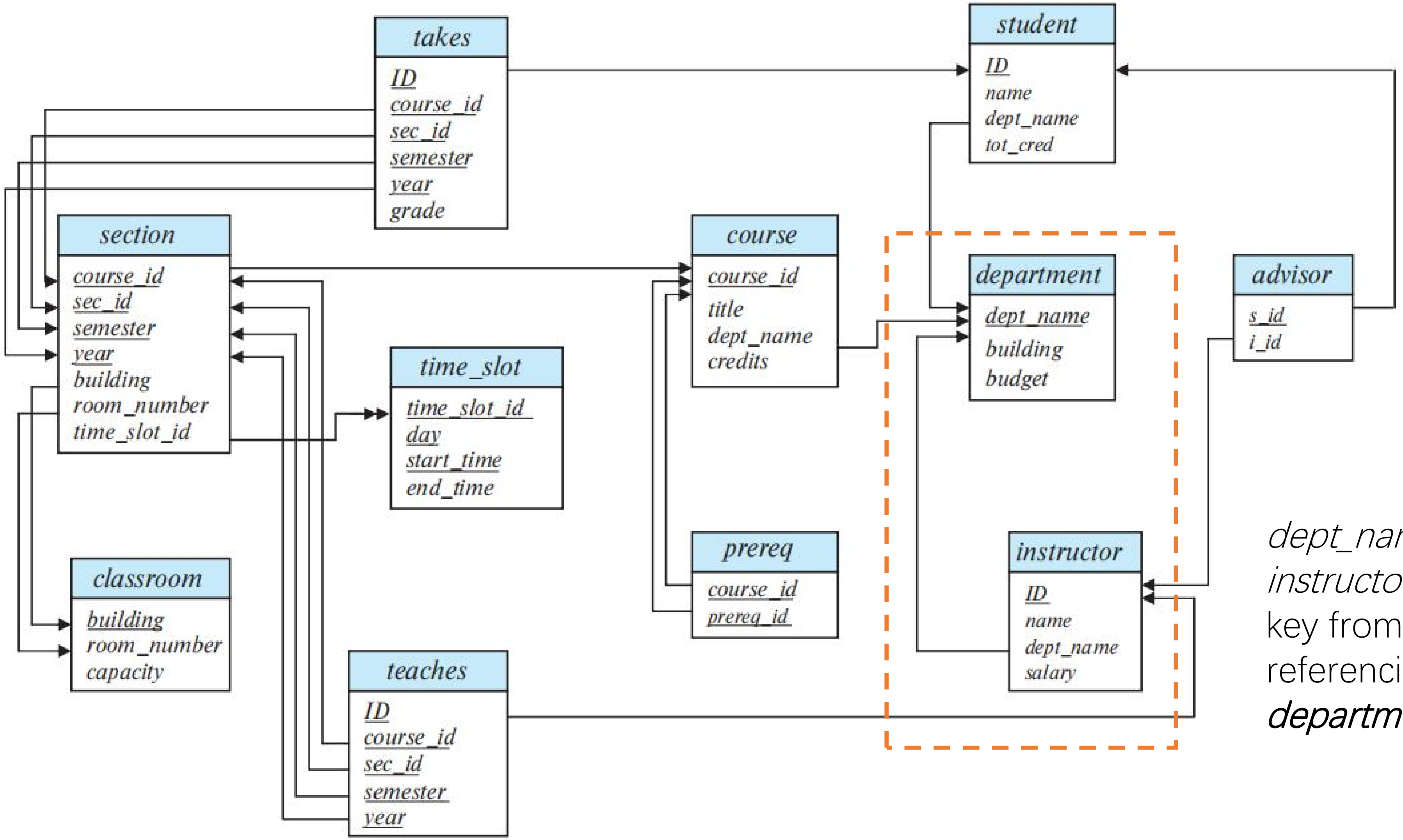
  instructor (*ID, name, dept_name, salary*)
  - Instance:

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

Keys

- Let $K \subseteq R$
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor.*
- Superkey *K* is a **candidate key** if *K* is minimal
  - Example: {*ID*} is a candidate key for *Instructor* while {ID, name} is not
- One of the candidate keys is selected to be the primary key.
  - Which one?
- Foreign key constraint: Value in one relation must appear in another
  - Referencing relation (foreign key)
  - Referenced relation (primary key)
  - Example: *dept_name* in i*nstructor* is a foreign key from *instructor* referencing *department*

Schema Diagram for University Database



dept_name in *instructor* is a foreign key from ***instructor*** referencing ***department***

Relational Query Languages

- Procedural versus non-procedural, or declarative

- "Pure" query languages
  - Relational algebra - procedural & functional
  - Tuple relational calculus - declarative
  - Domain relational calculus - declarative
- The above 3 pure languages are equivalent in computing power

- Relational algebra
  - Not Turing-machine equivalent
  - Consists of 6 basic operations

Relational Algebra

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

- Six basic operators
  - select: σ
  - project: Π
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ

Relational Algebra

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

- Six basic operators
  - select: σ
  - project: Π
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ

Unary Operation

Binary Operation

Select Operation

- The **select** (sigma) operation selects tuples that satisfy a given predicate.
  - Notation: $\sigma_{p}(r)$
  - p is called the selection predicate

- Example:

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

  - Query

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

  - Result

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

Select Operation (Cont.)

- We allow comparisons using

  $=, \neq, >, \geq. <. \leq$

  in the selection predicate.
- We can combine several predicates into a larger predicate by using the connectives:

  $\wedge$ (and), $\vee$ (or), $\neg$ (not)
- Example: Find the instructors in Physics with a salary greater $90,000, we write:

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

- The select predicate may include comparisons between two attributes.
  - Example, find all departments whose name is the same as their building name:
  - $\sigma_{\text{dept\_name=building}}$ (*department*)

Project Operation

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

$$\Pi_{A_1, A_2, A_3 \ldots, A_k}(r)$$

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

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets

Project Operation Example

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

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

- Result:

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

Composition of Relational Operations

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

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

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

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

Cartesian-Product Operation

- The Cartesian-product operation (denoted by X) allows us to combine information from any two relations.
- Example: the Cartesian product of the relations *instructor* and t*eaches* is written as:

  instructor X *teaches*

- We construct a tuple of the result out of each possible pair of tuples: one from the *instructor* relation and one from the *teaches* relation (see next slide)
- Since the instructor *ID* appears in both relations we distinguish between these attribute by attaching to the attribute the name of the relation from which the attribute originally came.
  - *instructor.ID*
  - *teaches.ID*

The instructor X teaches Table

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

Join Operation

- The Cartesian-Product

  instructor X *teaches*

  associates every tuple of instructor with every tuple of teaches.
  - Most of the resulting rows have information about instructors who did NOT teach a particular course.
- To get only those tuples of "*instructor* X *teaches*" that pertain to instructors and the courses that they taught, we write:

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

  - We get only those tuples of "*instructor* X *teaches*" that pertain to instructors and the courses that they taught.
- The result of this expression, shown in the next slide

Join Operation (Cont.)

- The table corresponding to:

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

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

Join Operation (Cont.)

- The **join** operation allows us to combine a select operation and a Cartesian-Product operation into a single operation.
- Consider relations *r* (*R*) and *s* (*S*)
- Let "theta" be a predicate on attributes in the schema R "union" S. The join operation is defined as follows:

$$r \bowtie_{\theta} s = \sigma_{\theta}\,(r \times s)$$

- Thus

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

- Can equivalently be written as

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

Set Operation - Union

- The union operation allows us to combine two relations
- Notation: $r \cup s$
- For $r \cup s$ to be valid.

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

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

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

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

Set Operation - Union (Cont.)

- Result of:

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

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

Set Operation - Intersection

- The set-intersection operation allows us to find tuples that are in both the input relations.
- Notation: $r \cap s$
- Assume:
  - *r*, *s* have the *same arity*
  - attributes of *r* and *s* are compatible
- Example: Find the set of all courses taught in both the Fall 2017 and the Spring 2018 semesters.

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

  - Result

course_id
CS-101

Set Difference Operation

- The set-difference operation allows us to find tuples that are in one relation but are not in another.
- Notation $r - s$
- Set differences must be taken between **compatible** relations.
  - r and s must have the same arity
  - attribute domains of r and s must be compatible
- Example: to find all courses taught in the Fall 2017 semester, but not in the Spring 2018 semester

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

course_id
CS-347
PHY-101

The Assignment Operation

- It is convenient at times to write a relational-algebra expression by assigning parts of it to temporary relation variables.
- The assignment operation is denoted by $\leftarrow$ and works like assignment in a programming language.
- Example: Find all instructor in the “Physics” and Music department.

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

$$Music \leftarrow \sigma_{dept\_name=\text{“Music”}}(instructor)$$

$$Physics \cup Music$$

- With the assignment operation, a query can be written as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as the result of the query.

The Rename Operation

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

$$\rho_x (E)$$

returns the result of expression *E* under the name *x*

- Another form of the rename operation:

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

Equivalent Queries

- There is more than one way to write a query in relational algebra.
- Example: Find information about courses taught by instructors in the Physics department with salary greater than 90,000
- Query 1

$$\sigma_{dept\_name=\text{"Physics"} \land salary > 90{,}000}(instructor)$$

- Query 2

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

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

Equivalent Queries (Cont.)

- There is more than one way to write a query in relational algebra.
- Example: Find information about courses taught by instructors in the Physics department
- Query 1

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

- Query 2

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

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

Review Terms

- Table
- Relation
- Tuple
- Attribute
- Relation instance
- Domain
- Atomic domain
- Null value
- Database schema
- Database instance
- Relation schema
- Keys
- Primary key constraint
- Foreign key constraint
- Schema diagram
- Relational algebra
- Relational-algebra expression
- Relational-algebra operations