Principles of Database Systems (CS307)

Lecturer's Cut: Theoretical Sections

Yuxin Ma

Department of Computer Science and Engineering
Southern University of Science and Technology

- Most contents are from slides made by Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.

Relational Algebra

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.

- 6 Basic Operators:
  - select: σ
  - project: $\prod$
  - union: $\cup$
  - set difference: $-$
  - Cartesian product: x
  - rename: ρ

Select Operation

- The select operation selects tuples that satisfy a given predicate
  - Notation: $\sigma_p(r)$
  - p is called the selection predicate

- Example
  - Select those tuples of the *instructor* relation where the instructor is in the "Physics" department

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

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

Select Operation

- We allow comparisons using $=, \neq, >, \geq. <. \leq$ in the selection predicate

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

$$\wedge \textbf{(and)}, \vee \textbf{(or)}, \neg \textbf{(not)}$$

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

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

- The select predicate may include comparisons between two attributes.
  - Example, find all departments whose name is the same as their building name:

$$\sigma_{dept\_name=building}(department)$$

Project Operation

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

Notation: $\prod_{A_1, A_2, A_3 \ldots A_k} (r)$

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

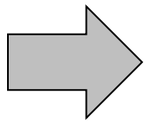
  - The result is defined as the relation of k columns obtained by erasing the columns that are not listed
  - Duplicate rows removed from result, since relations are sets

Project Operation

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

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

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



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 ×) allows us to combine information from any two relations.
  - Example: the Cartesian product of the relations instructor and teaches is written as:

$$\text{instructor} \times \text{teaches}$$

- 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)

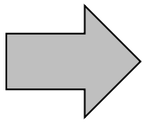
  - Since the instructor ID appears in both relations we distinguish between these attribute by attaching to the attribute the name of the relation from which the attribute originally came.
    - instructor.ID
    - teaches.ID

The "instructor × teaches" table

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

×

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4



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

- Problem: The Cartesian-Product "instructor × 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

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
...	...	...	...	...	...	...	...	...

Join Operation

- To get only those tuples of “instructor × 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 × teaches” that pertain to instructors and the courses that they taught
    - i.e., those tuples where instructor.id = teaches.id

Join Operation

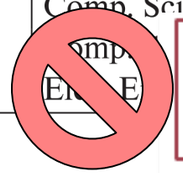
- The table corresponding to $\sigma_{instructor.id = teaches.id}(instructor \times teaches)$:

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

- The table corresponding to $\sigma_{instructor.id = teaches.id}(instructor \times teaches)$:

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.						
98345	Kim	El	E					



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
...	...	...	...	...	...	...	...	...

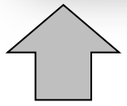
... will NOT include such tuples (rows) with different IDs

Recall: The Old Way of Writing Joins

- Use commas to separate the tables
  - Example: The solution for the same question in the previous slide

- A little bit history:
  - `join` was introduced in SQL-1999 (later than this original way)
- Problem:
  - If you forget a comma, it will still work sometimes (interpreted as “renaming”)

```sql
select m.title, c.credited_as,
       p.first_name, p.surname
from movies m,
     credits c,
     people p
where c.movieid = m.movieid
  and p.peopleid = c.peopleid
  and m.country = 'cn'
```



The SQL syntax was derived from the form of cartesian products in relational algreba

$$\sigma_{movies.id = credits.id \wedge \text{people.peopleid} = \text{credits.peopleid} \wedge \text{movies.country} = \text{“cn”}} (Movies \times Credits \times People)$$

Recall: The Old Way of Writing Joins

- Use commas to separate the tables
  - Example: The solution for the same question in the previous slide

- A little bit history:
  - `join` was introduced in SQL-1999 (later than this original way)
- Problem:
  - If you forget a comma, it will still work sometimes (interpreted as "renaming")

```
select m.title, c.credited_as,
       p.first_name, p.surname
from movies m,
     credits c,
     people p
where c.movieid = m.movieid
  and p.peopleid = c.peopleid
  and m.country = 'cn'
```

Use commas as the "multiplication signs"

The "select operation" is written as the "where" clause here

The SQL syntax was derived from the form of cartesian products in relational algreba

$$\sigma_{movies.id = credits.id \wedge \text{people.peopleid} = \text{credits.peopleid} \wedge \text{movies.country} = \text{“cn”}} (Movies \times Credits \times People)$$

Join Operation

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

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

  - Thus, $\sigma_{instructor.id = teaches.id} (instructor \times teaches)$ can equivalently be written as:

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

Union Operation

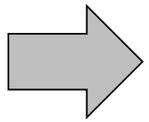
- The union operation allows us to combine two relations
  - Notation: $r \cup s$
- For $r \cup s$ to be valid:
  - r, s must have the same arity (same number of attributes)
  - The attribute domains must be compatible
    - Example: 2nd column of r deals with the same type of values as does the 2nd column of s

Union Operation

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

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

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



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

Set-Intersection Operation

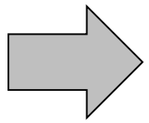
- The set-intersection operation allows us to find tuples that are in both the input relations
  - Notation: $r \cap s$
- Assume (same as Union):
  - r, s have the same arity
  - Attributes of r and s are compatible

Set-Intersection Operation

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

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

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



course_id
CS-101

Set Difference Operation

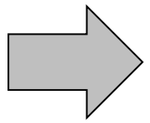
- The set-difference operation allows us to find tuples that are in one relation but are not in another
  - Notation: $r - s$
- Assume (same as Union and Set Intersection):
  - r, s have the same arity
  - Attributes of r and s are compatible

Set Difference Operation

- Example: Find all courses taught in the Fall 2017 semester, but not in the Spring 2018 semester

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

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



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 ← 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 which also renames the columns:
    - $\rho_{x(A1,A2,\ldots An)}(E)$

Equivalent Queries

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

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

    - Query 2

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

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

Equivalent Queries

- Example: Find information about courses taught by instructors in the Physics department
  - Query 1

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

    - (Join first, then select)

  - Query 2

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

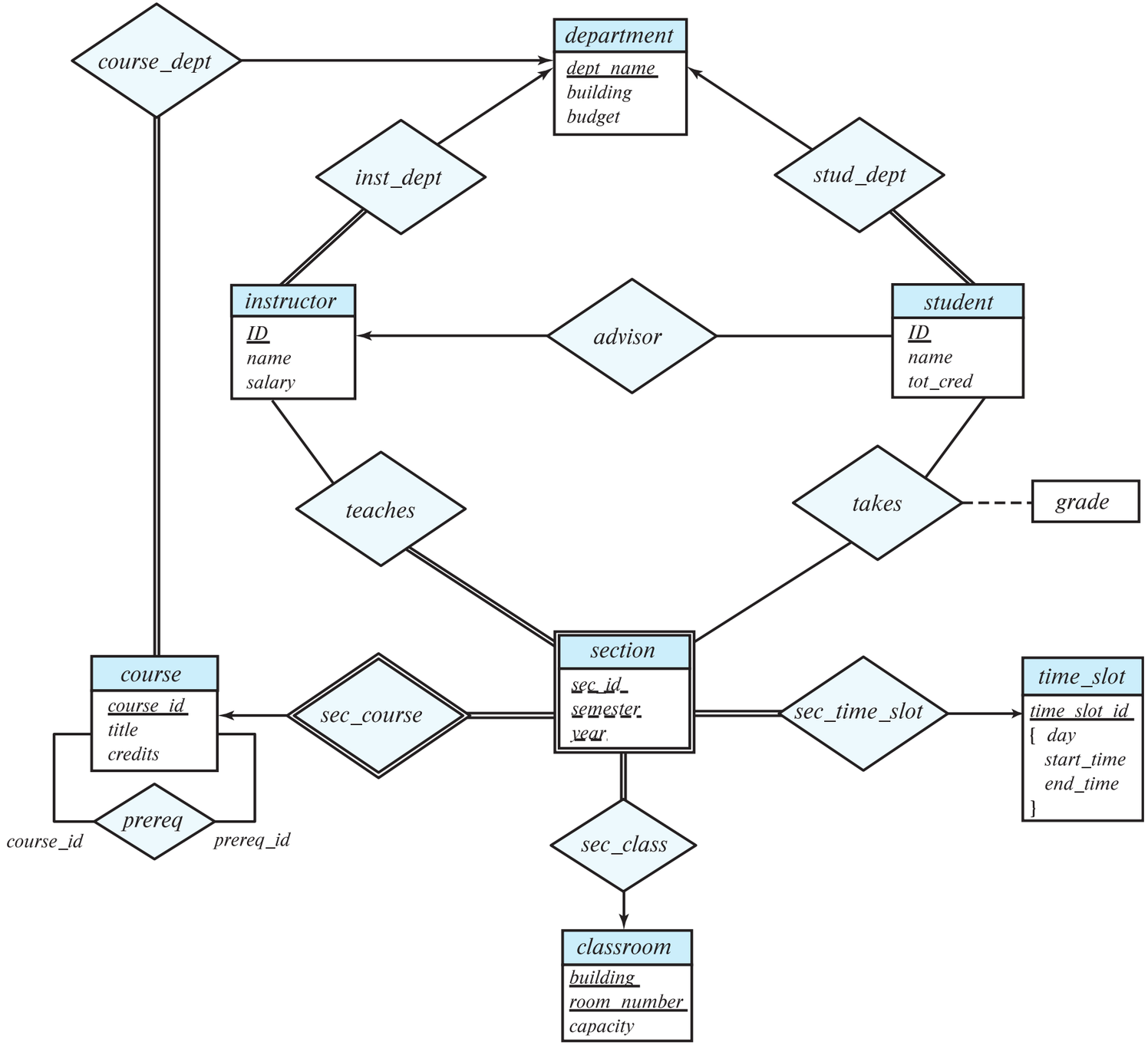
    - (Select first, then join)

Equivalent Queries

- Application of Relational Algebra: Query Optimization
  - Transform queries into equivalent ones with less computational cost

Entity-Relationship Model & Diagram

The New Running Example



Design Phases

- Initial phase: characterize fully the data needs of the prospective database users.
- Second phase: choosing a data model
  - Applying the concepts of the chosen data model
  - Translating these requirements into a conceptual schema of the database
  - A fully developed conceptual schema indicates the functional requirements of the enterprise
    - Describe the kinds of operations (or transactions) that will be performed on the data

Design Phases

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

Design Alternatives

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

Entity-Relationship Model (E-R Model)
Entity-Relationship Diagram (E-R Diagram)

Design Approaches

- Entity Relationship Model (covered in this chapter)
  - Models an enterprise as a collection of entities and relationships
    - **Entity**: a “thing” or “object” in the enterprise that is distinguishable from other objects
      - Described by a set of attributes
    - **Relationship**: an association among several entities
  - Represented diagrammatically by an entity-relationship diagram (E-R diagram)

- Normalization Theory (coming in the next few weeks)
  - Formalize what designs are bad, and test for them

Entity Sets

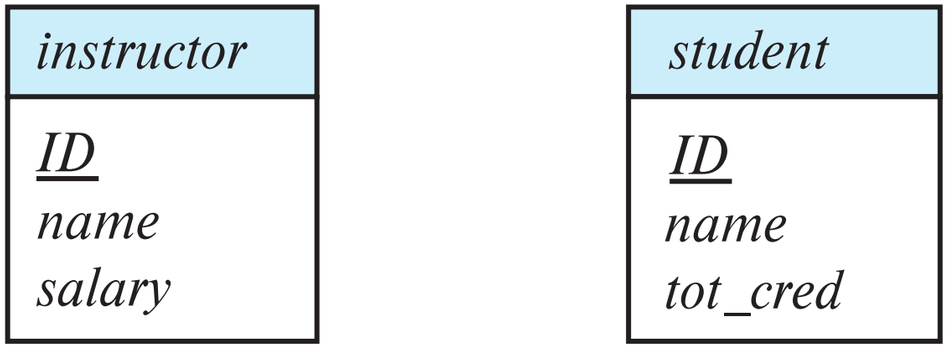
- An entity is an object that exists and is distinguishable from other objects
  - Example: specific person, company, event, plant
- An entity set is a set of entities of the same type that share the same properties
  - Example: set of all persons, companies, trees, holidays
- An entity is represented by a set of attributes; i.e., descriptive properties possessed by all members of an entity set.
  - Example:

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

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

Representing Entity sets in ER Diagram

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



Relationship Sets

- A relationship is an association among several entities

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

- A relationship set is a mathematical relation among n ≥ 2 entities, each taken from entity sets

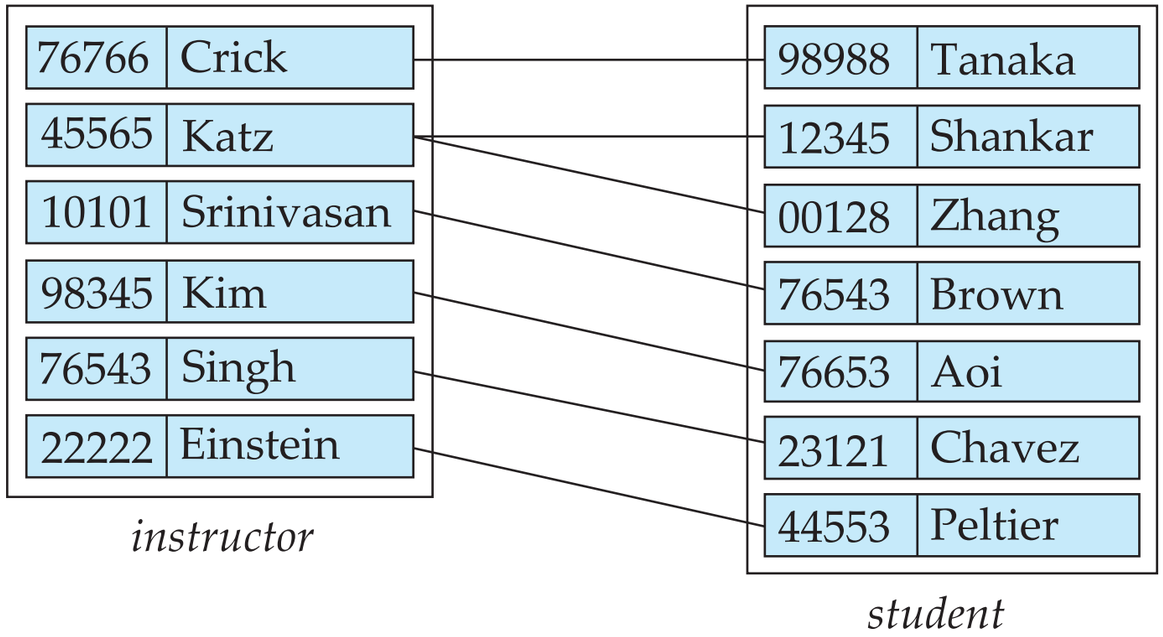
$$\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

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

  - Example: (44553,22222) ∈ advisor

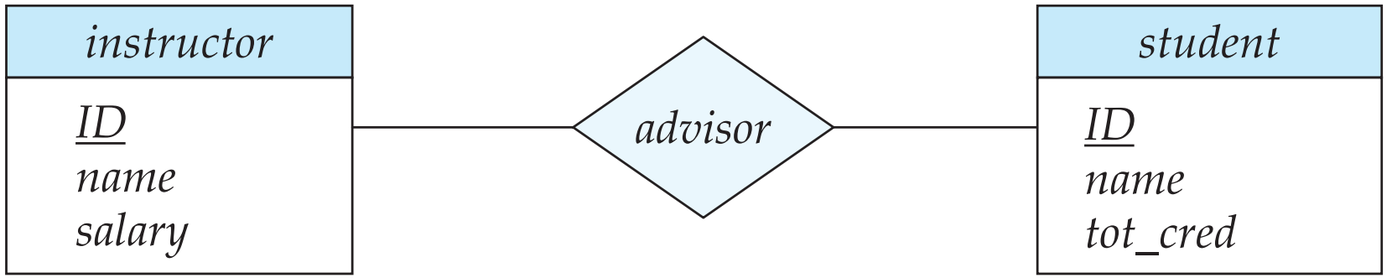
Relationship Sets

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



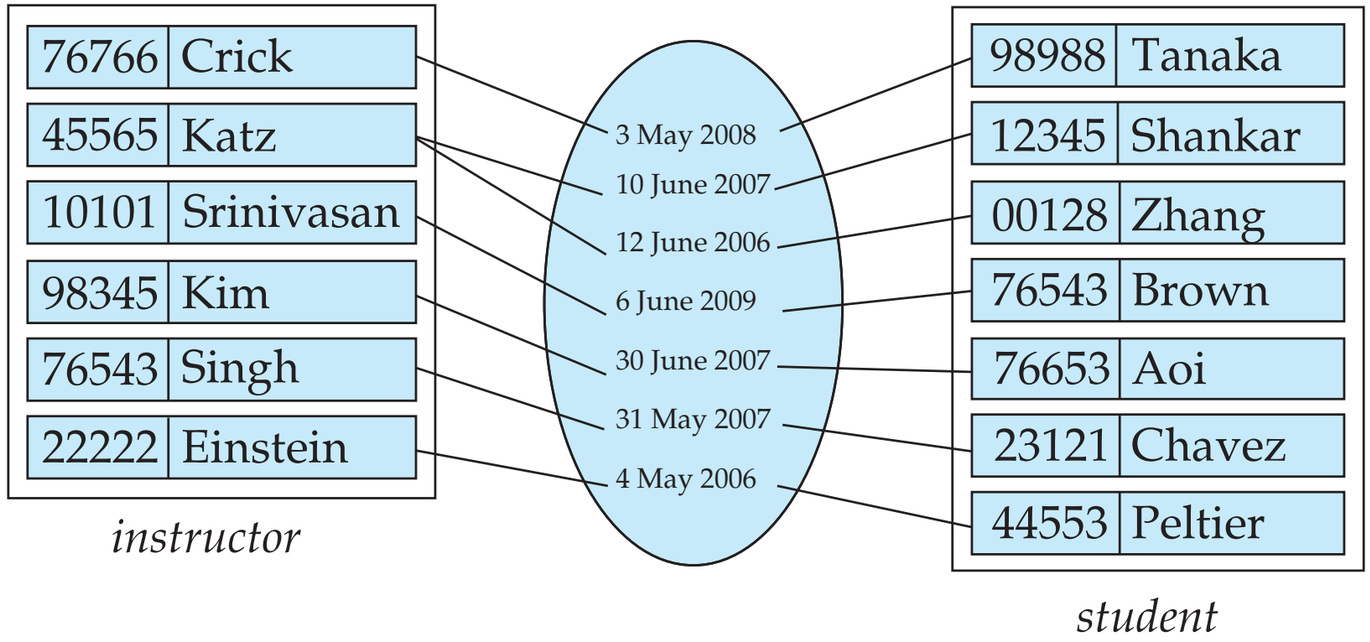
Representing Relationship Sets via E-R Diagrams

- Diamonds represent relationship sets

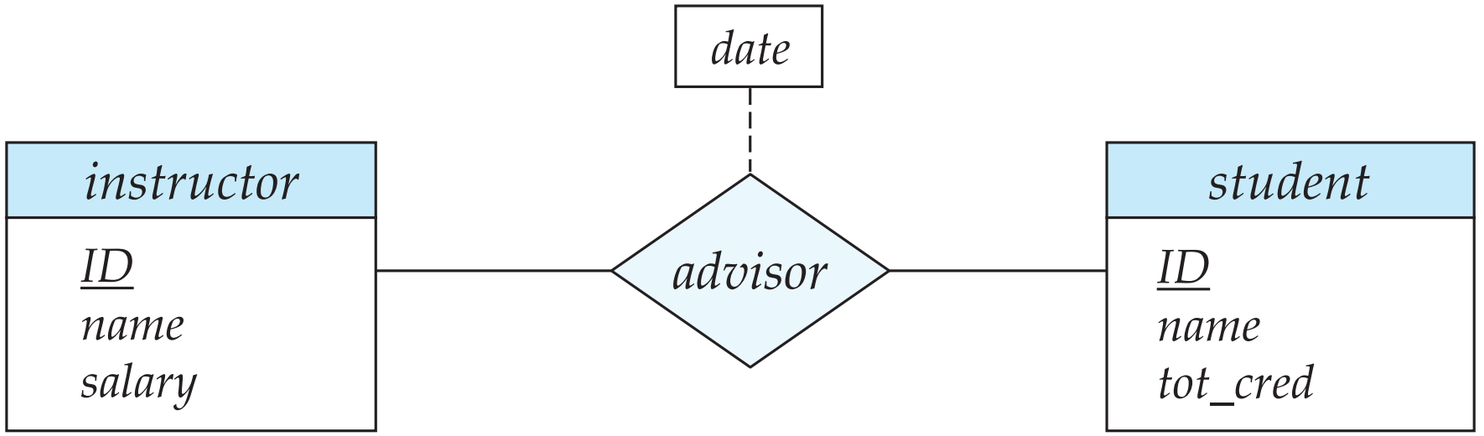


Relationship Sets (Cont.)

- An attribute can also be associated with a relationship set.
  - For instance, the advisor relationship set between entity sets instructor and student may have the attribute `date` which tracks when the student started being associated with the advisor



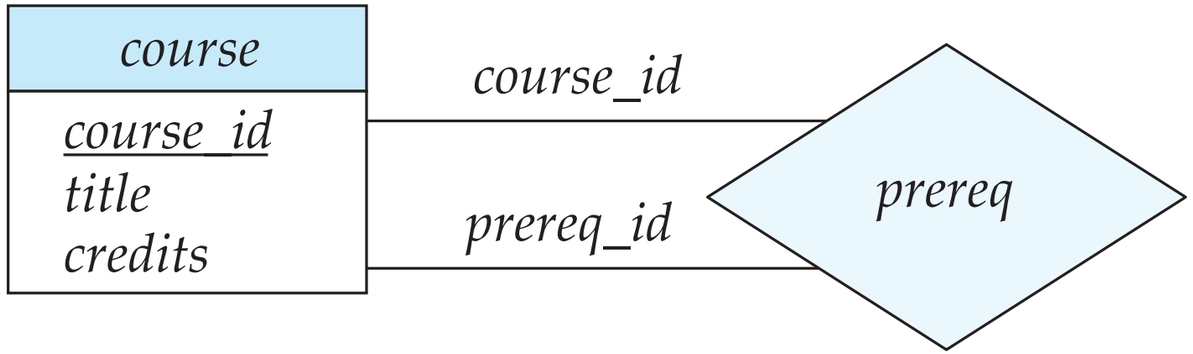
Relationship Sets with Attributes



Roles

- Entity sets of a relationship need not be distinct
  - That is to say, we can create self-pointing relationships for an entity set
  - Each occurrence of an entity set plays a "role" in the relationship

  - Example: A relationship set to represent the prerequisites of a course
    - E.g., Data Structure depends on Introduction to Programming
    - The labels "`course_id`" and "`prereq_id`" are called roles

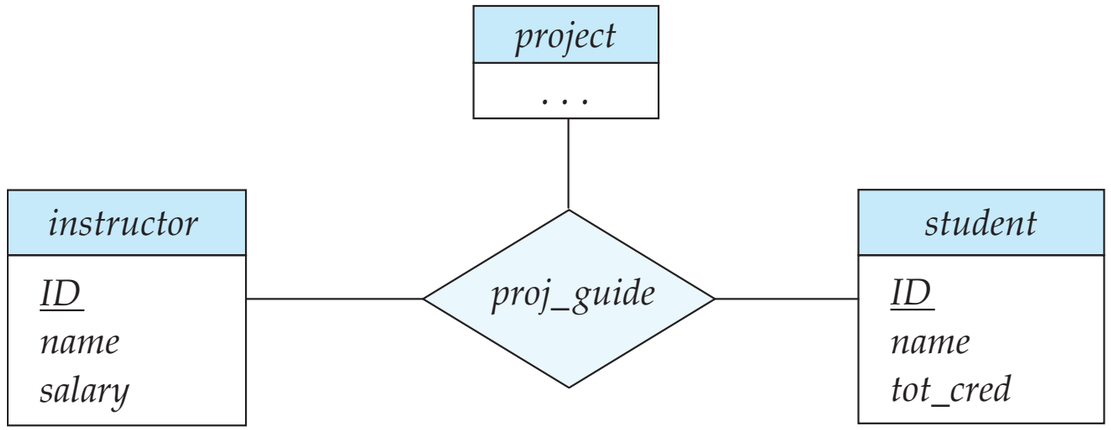


Degree of a Relationship Set

- Binary relationship
  - Involve **two** entity sets (or degree two).
  - Most relationship sets in a database system are binary
- Relationships between more than two entity sets are rare
  - Example: students work on research projects under the guidance of an instructor.
    - relationship `proj_guide` is a ternary relationship between instructor, student, and project

Non-binary Relationship Sets

- Most relationship sets are binary
  - There are occasions when it is more convenient to represent relationships as non-binary
- E-R Diagram with a Ternary Relationship



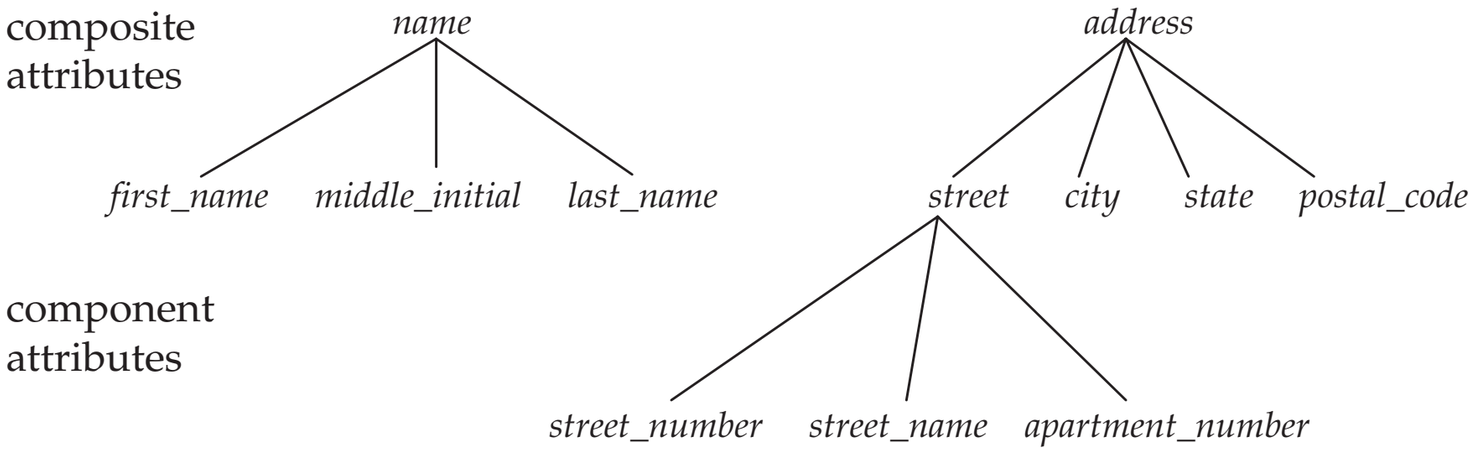
Complex Attributes

- Attribute types:
  - Simple and composite attributes.
  - Single-valued and multivalued attributes
    - Example: multivalued attribute: phone_numbers
      - A person can have 1 or more phone numbers at the same time
  - Derived attributes
    - Can be computed from other attributes
    - Example: age, given date_of_birth

- **Domain**: The set of permitted values for each attribute

Composite Attributes

- Composite attributes allow us to divided attributes into subparts (other attributes)
  - Sometimes we may only use part of the attributes, where the composite attribute is a good design choice



instructor

ID
name
- *first_name*
- *middle_initial*
- *last_name*

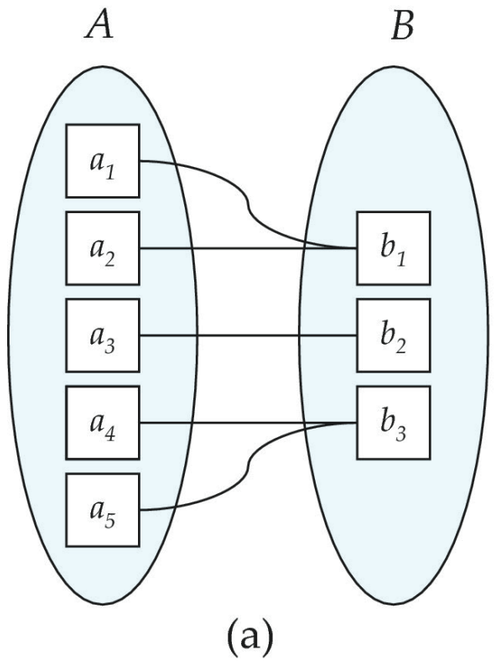
address
- *street*
  - *street_number*
  - *street_name*
  - *apt_number*
- *city*
- *state*
- *zip*

{ phone_number }
date_of_birth
age ( )

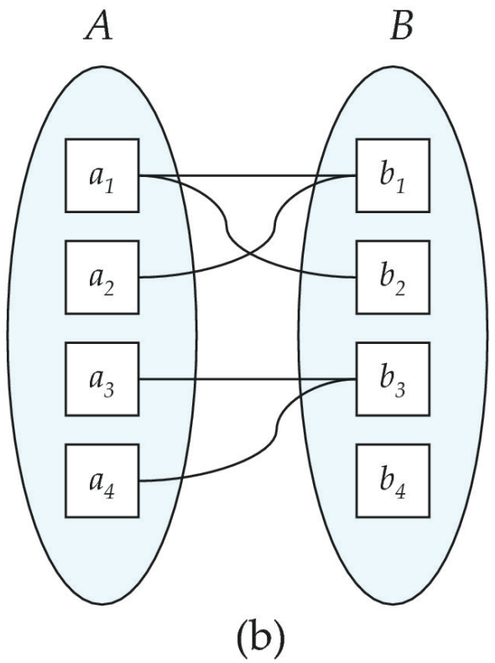
Mapping Cardinality Constraints

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

Mapping Cardinalities



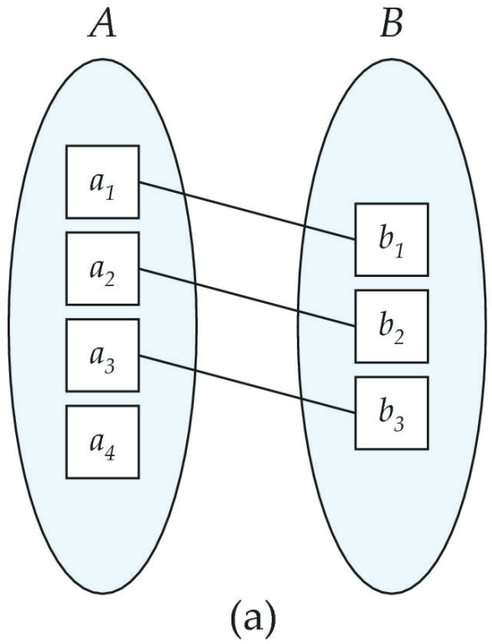
Many to one



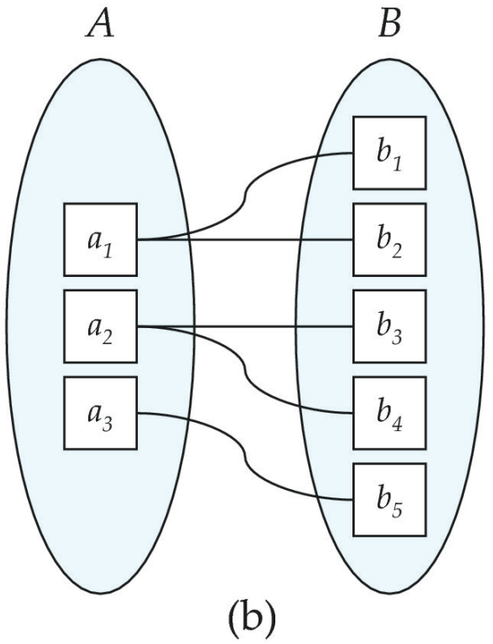
Many to many

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

Mapping Cardinalities



One to one



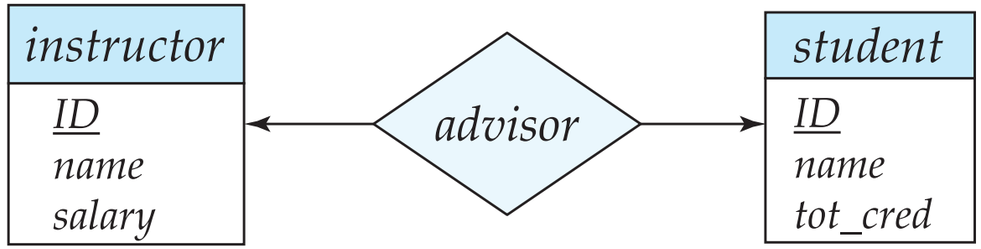
One to many

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

Representing Cardinality Constraints in ER Diagram

- We express cardinality constraints by:
  - drawing either a directed line (→), signifying “one,”
  - or an undirected line (—), signifying “many,”
- … between the relationship set and the entity set.

- One-to-one relationship between an instructor and a student :
  - A student is associated with at most one instructor via the relationship advisor



Representing Cardinality Constraints in ER Diagram

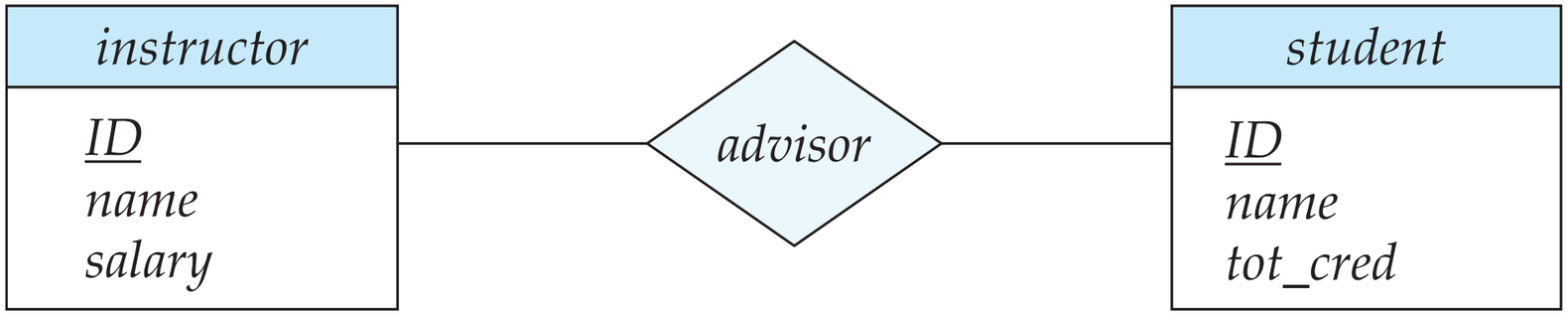
- One-to-many relationship between an instructor and a student
  - an instructor is associated with several (including 0) students via advisor
  - a student is associated with at most one instructor via advisor

Representing Cardinality Constraints in ER Diagram

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

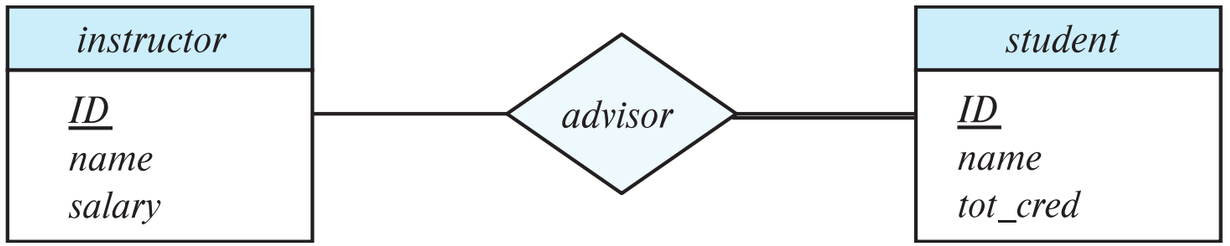
Representing Cardinality Constraints in ER Diagram

- Many-to-many relationship:
  - An instructor is associated with several (possibly 0) students via advisor
  - A student is associated with several (possibly 0) instructors via advisor



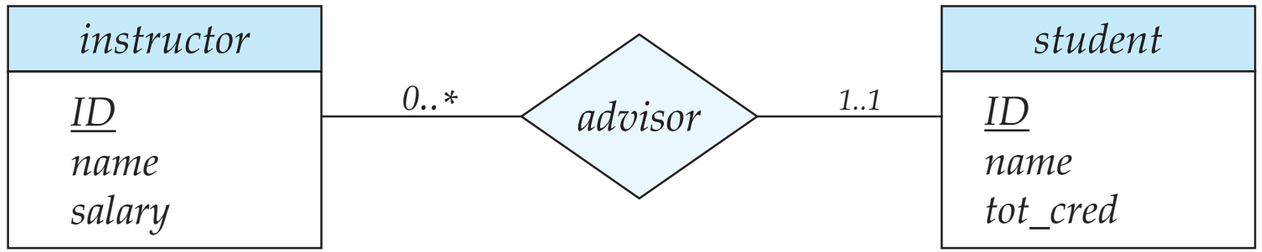
Total and Partial Participation

- Total participation (indicated by *double line*)
  - Every entity in the entity set participates in at least one relationship in the relationship set
  - Example: Participation of student in advisor relation is total
    - i.e., every student must have an associated instructor
- Partial participation
  - Some entities may not participate in any relationship in the relationship set
  - Example: participation of instructor in advisor is partial



Notation for Expressing More Complex Constraints

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



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

Primary Key

- Primary keys provide a way to specify how entities and relations are distinguished

Primary Key for Entity Sets

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

- A key for an entity is a set of attributes that suffice to distinguish entities from each other

Primary Key for Relationship Sets

- To distinguish among the various relationships of a relationship set, we use the **individual primary keys** of the entities in the relationship set.
  - Let R be a relationship set involving entity sets E1, E2, .. En
  - The primary key for R consists of the union of the primary keys of entity sets E1, E2, ..En
  - If the relationship set R has attributes $a_1, a_2, .., a_m$ associated with it, the primary key of R also includes the attributes $a_1, a_2, .., a_m$
- Example: relationship set "advisor".
  - The primary key consists of instructor.ID and student.ID

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

Choice of Primary key for Binary Relationship

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

* *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*.

Choice of Primary key for Binary Relationship

- One-to-Many relationships
  - The primary key of the “Many” side is a minimal superkey and is used as the primary key.
- Many-to-one relationships
  - The primary key of the “Many” side is a minimal superkey and is used as the primary key.

Weak Entity Sets

- Consider a section entity, which is uniquely identified by a course_id, semester, year, and sec_id.
  - Clearly, section entities are related to course entities. Suppose we create a relationship set sec_course between entity sets section and course.
  - Note that the information in sec_course is redundant, since section already has an attribute course_id, which identifies the course with which the section is related.
  - One option to deal with this redundancy is to get rid of the relationship sec_course; however, by doing so the relationship between section and course becomes implicit in an attribute, which is not desirable.

Weak Entity Sets

- An alternative way to deal with this redundancy is to not store the attribute course_id in the section entity and to only store the remaining attributes section_id, year, and semester.
  - However, the entity set section then does not have enough attributes to identify a particular section entity uniquely
- To deal with this problem, we treat the relationship sec_course as a special relationship that provides extra information, in this case, the course_id, required to identify section entities uniquely.

- A weak entity set is one whose existence is dependent on another entity, called its identifying entity
- Instead of associating a primary key with a weak entity, we use the identifying entity, along with extra attributes called discriminator to uniquely identify a weak entity.

Weak Entity Sets

- An entity set that is not a weak entity set is termed a strong entity set.

- Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be existence dependent on the identifying entity set.
  - The identifying entity set is said to own the weak entity set that it identifies.
  - The relationship associating the weak entity set with the identifying entity set is called the identifying relationship

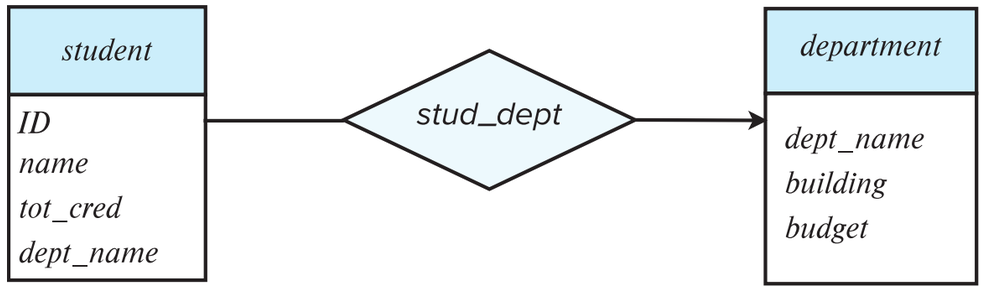
- Note that the relational schema we eventually create from the entity set section does have the attribute course_id, for reasons that will become clear later, even though we have dropped the attribute course_id from the entity set section.

Expressing Weak Entity Sets

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

Redundant Attributes

- Suppose we have entity sets:
  - student, with attributes: ID, name, tot_cred, dept_name
  - department, with attributes: dept_name, building, budget
- We model the fact that each student has an associated department using a relationship set stud_dept
- The attribute dept_name in student below replicates information present in the relationship and is therefore redundant
  - and needs to be removed.



(a) Incorrect use of attribute

- BUT: when converting back to tables, in some cases the attribute gets reintroduced.

Reduction from E-R Model to Relation Schemas

Reduction to Relation Schemas

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

Representing Entity Sets

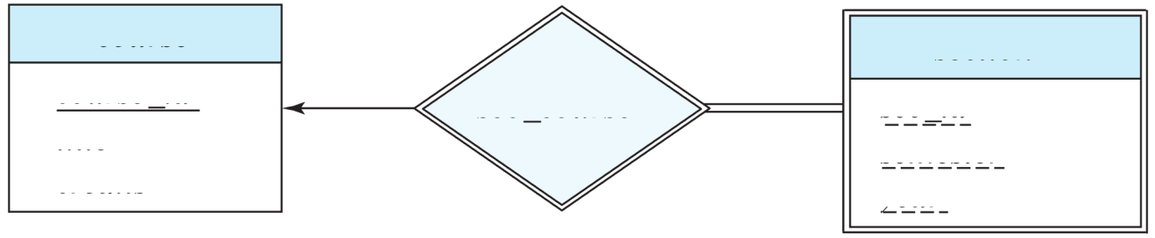
- A strong entity set reduces to a schema with the same attributes

  student(ID, name, tot_cred)

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

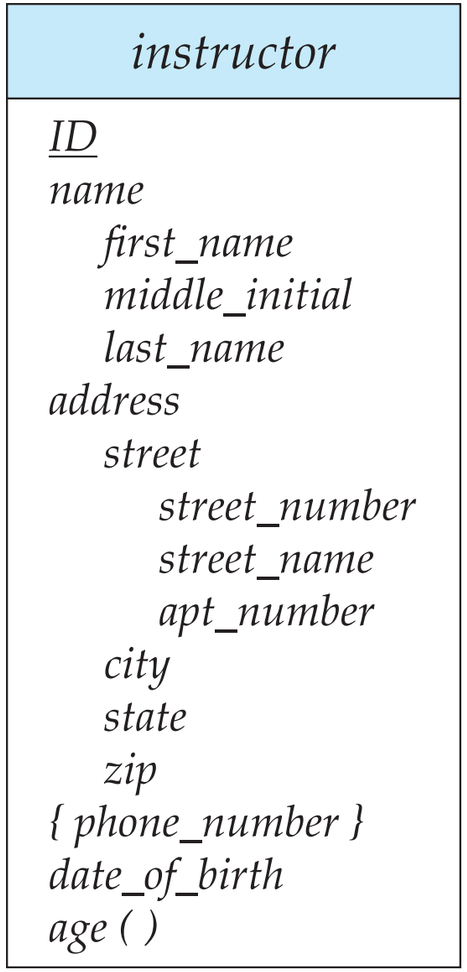
  section (course_id, sec_id, sem, year)

- Example



Representation of Entity Sets with Composite Attributes

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



Representation of Entity Sets with Multivalued Attributes

- A multivalued attribute M of an entity E is represented by a separate schema EM
  - Schema EM has attributes corresponding to the primary key of E and an attribute corresponding to multivalued attribute M
  - Example: Multivalued attribute phone_number of instructor is represented by a schema:

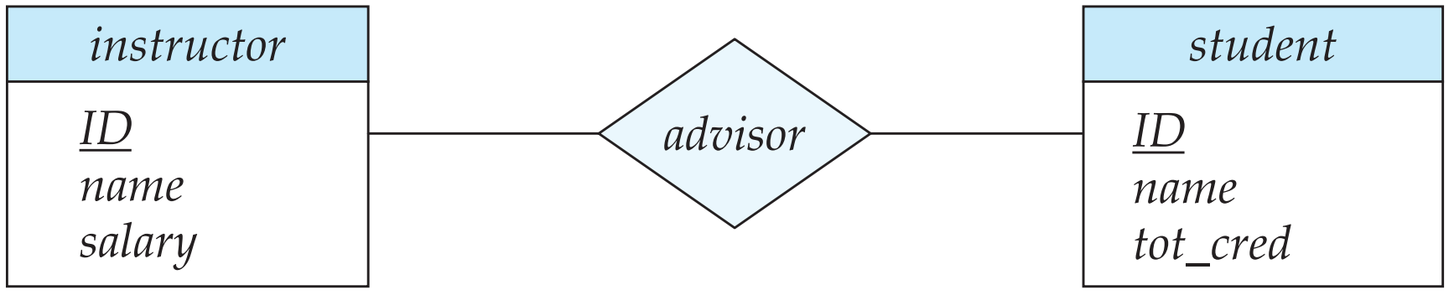
    inst_phone= (ID, phone_number)
  - Each value of the multivalued attribute maps to a separate tuple of the relation on schema EM
  - For example, an instructor entity with primary key 22222 and phone numbers 456-7890 and 123-4567 maps to two tuples:

    (22222, 456-7890) and *(22222, 123-4567)*

Representing Relationship Sets

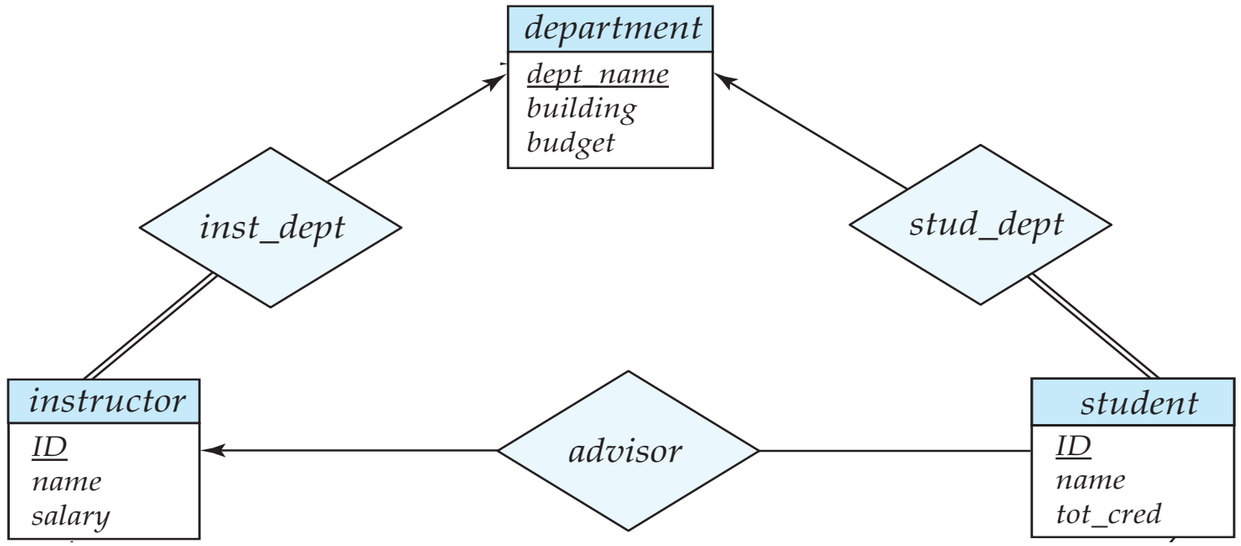
- A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.
  - Example: schema for relationship set advisor

$$advisor = (\underline{s\_id}, \underline{i\_id})$$



Redundancy of Schemas

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



Redundancy of Schemas

- For one-to-one relationship sets, either side can be chosen to act as the “many” side
  - That is, an extra attribute can be added to either of the tables corresponding to the two entity sets

- * If participation is partial on the “many” side, replacing a schema by an extra attribute in the schema corresponding to the “many” side could result in null values

Redundancy of Schemas

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

Normalization

Recall: Design Alternatives

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

Recall: Design Alternatives

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

- Do we have any guidelines on how to get a good design?
  - Normal Forms!

First Normal Form (1NF)

- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
  - Domain is atomic if its elements are considered to be indivisible units
    - Examples of non-atomic domains:
      - Set of names, composite attributes
      - Identification numbers like CS101 that can be broken up into parts
        - However, in practice, we can also consider it atomic
  - Non-atomic values complicate storage and encourage redundant (repeated) storage of data

First Normal Form (1NF)

- Example: Non-atomic attribute

station_id	name	location
1	Luohu(罗湖)	114.11833 , 22.53111
2	Guomao(国贸)	114.11889 , 22.54
3	Laojie(老街)	114.11639 , 22.54444
4	Grand Theater(大剧院)	114.10333 , 22.54472
5	Science Museum(科学馆)	114.08972 , 22.54333
6	Huaqiang Rd(华强路)	114.07889 , 22.54306
7	Gangxia(岗厦)	114.06306 , 22.53778
8	Convention and Exhibition Center Station(会展中心)	114.05472 , 22.5375
9	Shopping Park(购物公园)	114.05472 , 22.53444
10	Xiangmihu(香蜜湖)	114.034 , 22.5417

First Normal Form (1NF)

- Another example: Starring
  - Problems: 1) Redundant names; 2) difficulties in updating/deleting a specific person; 3) extra cost in splitting names; 4) difficulties in making statistics

Movie ID	Movie Title	Country	Year	Director	Starring
0	Citizen Kane	US	1941	welles, o.	Orson Welles, Joseph Cotten
1	La règle du jeu	FR	1939	Renoir, J.	Roland Toutain, Nora Grégor, Marcel Dalio, Jean Renoir
2	North By Northwest	US	1959	HITCHCOCK, A.	Cary Grant, Eva Marie Saint, James Mason
3	Singin' in the Rain	US	1952	Donen/Kelly	Gene Kelly, Debbie Reynolds, Donald O'Connor
4	Rear Window	US	1954	Alfred Hitchcock	James Stewart, Grace Kelly

First Normal Form (1NF)

- Fix it by splitting the names into two columns

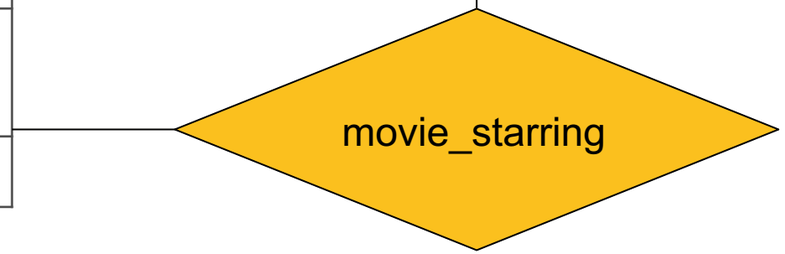
station_id	english_name	chinese_name	longitude	latitude
1	Luohu	罗湖	114.11833	22.53111
2	Guomao	国贸	114.11889	22.54
3	Laojie	老街	114.11639	22.54444
4	Grand Theater	大剧院	114.10333	22.54472
5	Science Museum	科学馆	114.08972	22.54333
6	Huaqiang Rd	华强路	114.07889	22.54306
7	Gangxia	岗厦	114.06306	22.53778
8	Convention and Exhibition Cent…	会展中心	114.05472	22.5375
9	Shopping Park	购物公园	114.05472	22.53444
10	Xiangmihu	香蜜湖	114.034	22.5417

First Normal Form (1NF)

- Fix it by treating the column as a multi-valued attribute

Movie ID	Movie Title	Country	Year	Director
0	Citizen Kane	US	1941	welles, o.
1	La règle du jeu	FR	1939	Renoir, J.
2	North By Northwest	US	1959	HITCHCOCK, A.
3	Singin' in the Rain	US	1952	Donen/Kelly
4	Rear Window	US	1954	Alfred Hitchcock

Star ID	Firstname	Lastname	Born	Died
1				
2				
3				



Second Normal Form (2NF)

- A relation satisfying 2NF must:
  - be in 1NF
  - not have any non-prime attribute that is dependent on any proper subset of any candidate key of the relation
    - A non-prime attribute of a relation is an attribute that is not a part of any candidate key of the relation.

Second Normal Form (2NF)

- Example: Consider this table with the composite primary key (*station_id, line_id*)

station_id	english_name	chinese_name	district	line_id	line_color	operator
1	Luohu	罗湖	Luohu	1	Green	Shenzhen Metro Corporation
2	Guomao	国贸	Luohu	1	Green	Shenzhen Metro Corporation
3	Laojie	老街	Luohu	1	Green	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	1	Green	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	11	Purple	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	2	Orange	Shenzhen Metro Corporation
3	Laojie	老街	Luohu	3	DeepSkyBlue	Shenzhen Metro No.3 Line

  - The columns *line_color* and *operator* are not related to station_id
    - They are only related to *line_id*, which is only part of (a subset of) the primary key
  - Similarly, *english_name*, *chinese_name*, and *district* are not related to *line_id*
    - They are only related to station_id, which is only part of (a subset of) the primary key

Second Normal Form (2NF)

- Example: Consider this table with the composite primary key (*station_id, line_id*)

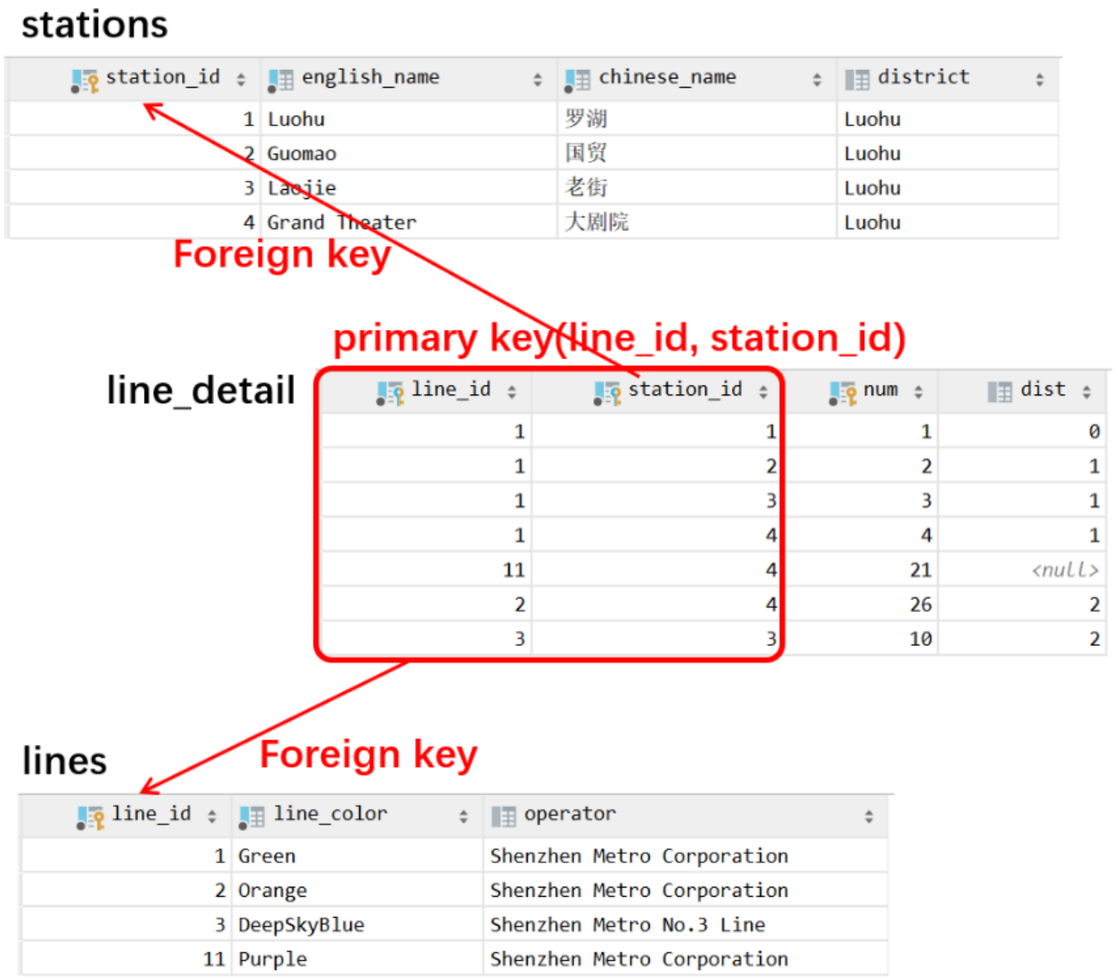
station_id	english_name	chinese_name	district	line_id	line_color	operator
1	Luohu	罗湖	Luohu	1	Green	Shenzhen Metro Corporation
2	Guomao	国贸	Luohu	1	Green	Shenzhen Metro Corporation
3	Laojie	老街	Luohu	1	Green	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	1	Green	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	11	Purple	Shenzhen Metro Corporation
4	Grand Theater	大剧院	Luohu	2	Orange	Shenzhen Metro Corporation
3	Laojie	老街	Luohu	3	DeepSkyBlue	Shenzhen Metro No.3 Line

  - Problem when not meeting 2NF: Insertion and deletion anomaly
    - We cannot insert a new station with no lines assigned yet (unless using NULLs)
    - If we delete a line, all stations associated with this line will be deleted as well

Second Normal Form (2NF)

- Fix it by
  - Splitting the two unrelated parts into two different tables of entities
  - And create a relationship set (if it is the many-to-many relationship between the two entities)

- By the way…
  - A relation with a single-attribute primary key is automatically in 2NF once it meets 1NF.



Third Normal Form (3NF)

- A relation satisfying 3NF must:
  - be in 2NF
  - all the attributes in a table are determined only by the candidate keys of that relation and not by any non-prime attributes

Third Normal Form (3NF)

- Example: Consider this table which describes the bus lines and their stops
  - Primary key (*bus_line*)

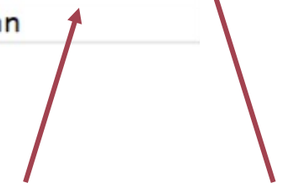
bus_line	station_id	chinese_name	english_name	district
B796	21	鲤鱼门	Liyumen	Nanshan
M343	21	鲤鱼门	Liyumen	Nanshan
M349	21	鲤鱼门	Liyumen	Nanshan
M250	26	坪洲	Pingzhou	Bao'an
374	61	安托山	Antuo Hill	Futian
B733	61	安托山	Antuo Hill	Futian
B828	120	临海	Linhai	Nanshan

  - The column *station_id* depends on the primary key (bus_line)
  - However, the columns *chinese_name*, *english_name*, and *district* depend on *station_id*, which is not the primary key.
    - They only have “indirect/transitive” dependence on the primary key
  - Problem: Data redundancy

Third Normal Form (3NF)

- Example: Consider this table which describes the bus lines and their stops
  - Primary key (*bus_line*)

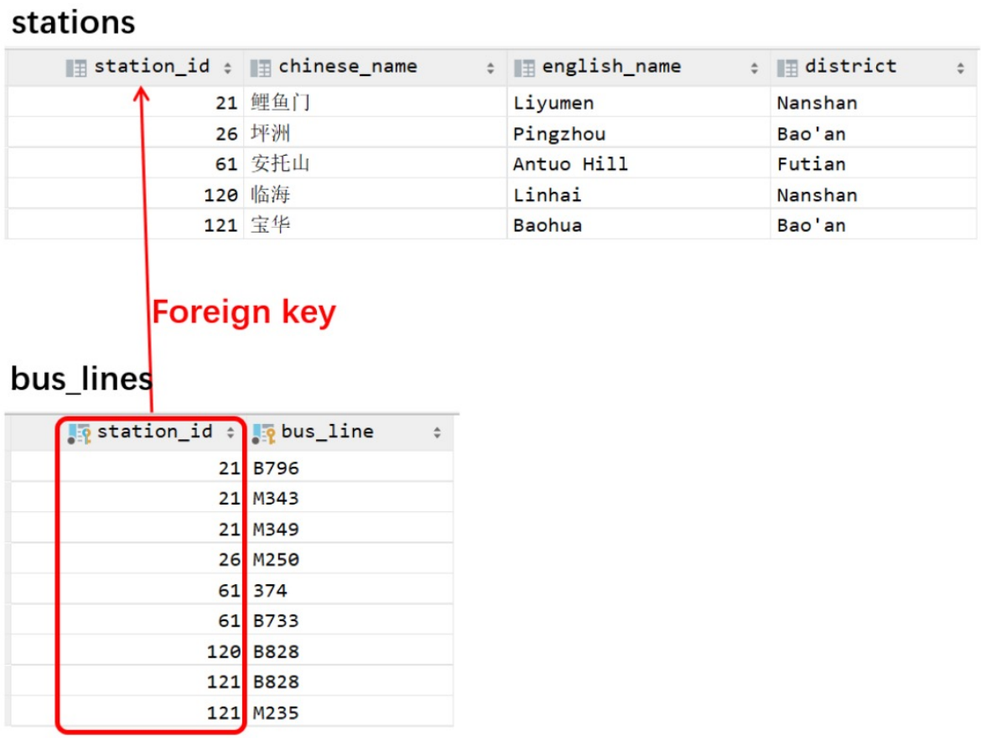
bus_line	station_id	chinese_name	english_name	district
B796	21	鲤鱼门	Liyumen	Nanshan
M343	21	鲤鱼门	Liyumen	Nanshan
M349	21	鲤鱼门	Liyumen	Nanshan
M250	26	坪洲	Pingzhou	Bao'an
374	61	安托山	Antuo Hill	Futian
B733	61	安托山	Antuo Hill	Futian
B828	120	临海	Linhai	Nanshan



- Problem when not meeting 3NF:
  - Data redundancy: as you can see in the table, the attributes for a station have been stored multiple times
  - Insertion and deletion anomaly: inserting a new bus line with no station becomes impossible without NULLs; deleting a station/bus line may also delete corresponding bus lines/stations.

Third Normal Form (3NF)

- Fix it by:
  - Create a new table with *station_id* as the primary key
    - i.e., the column which *chinese_name*, *english_name*, and *district* depend on
  - Move all columns which depend on the new primary key into the new table
    - And, only leave the primary key of the new table (station_id) in the original table
  - Add a foreign-key constraint



Normalization

- In practice, we usually just satisfy 1NF, 2NF and 3NF

	UNF (1970)	1NF (1970)	2NF (1971)	3NF (1971)	EKNF (1982)	BCNF (1974)	4NF (1977)	ETNF (2012)	5NF (1979)	DKNF (1981)	6NF (2003)
Primary key (no duplicate tuples)[4]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Atomic columns (cells cannot have tables as values)[5]	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either does not begin with a proper subset of a candidate key or ends with a prime attribute (no partial functional dependencies of non-prime attributes on candidate keys)[5]	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either begins with a superkey or ends with a prime attribute (no transitive functional dependencies of non-prime attributes on candidate keys)[5]	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either begins with a superkey or ends with an elementary prime attribute	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	N/A
Every non-trivial functional dependency begins with a superkey	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	N/A
Every non-trivial multivalued dependency begins with a superkey	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	N/A
Every join dependency has a superkey component[8]	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	N/A
Every join dependency has only superkey components	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	N/A
Every constraint is a consequence of domain constraints and key constraints	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
Every join dependency is trivial	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

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

Normalization

Every non key attribute must provide a fact about the key, the whole key, and nothing but the key.

William Kent (1936 – 2005)

William Kent. "A Simple Guide to Five Normal Forms in Relational Database Theory", Communications of the ACM 26 (2), Feb. 1983, pp. 120–125.

More to Read

- A formal definition of NFs
  - Chapter 7, Relational Database Design, “Database System Concepts”

- The original research papers on normal forms
  - E. F. Codd. 1970. A relational model of data for large shared data banks. Commun. ACM 13, 6 (June 1970), 377–387. DOI:https://doi.org/10.1145/362384.362685
  - Codd, E. F.. “Further Normalization of the Data Base Relational Model.” Research Report / RJ / IBM / San Jose, California RJ909 (1971).
  - Armstrong, William Ward. “Dependency Structures of Data Base Relationships.” IFIP Congress (1974).

Transaction

Transaction in Real Life

- “An exchange of goods for money”
  - A series of steps
  - All or nothing

Flickr:BracketingLife (Clarence)

Transaction in Computer

- A transaction is a unit of program execution that accesses and possibly updates various data items
  - A classical example in database: money transfer

E.g., transaction to transfer $50 from account A to account B:

```
1. read(A)
2. A := A – 50
3. write(A)
4. read(B)
5. B := B + 50
6. write(B)
```

An Example of Transactions in PostgreSQL

- BEGIN, COMMIT, ROLLBACK

```
begin;  -- Start a transaction

update people_1 set num_movies = 50000 where peopleid = 1;

select * from people_1 where peopleid = 1;

delete from people_1 where peopleid > 100 and peopleid < 200;

commit;  -- start executing all the queries above
-- or "rollback;", which means to revoke the operationso of all the queries
```

Transaction in Computer

- A transaction is a unit of program execution that accesses and possibly updates various data items
  - A classical example in database: money transfer

E.g., transaction to transfer $50 from account A to account B:

```
1. read(A)
2. A := A – 50
3. write(A)
4. read(B)
5. B := B + 50
6. write(B)
```

- Two main issues to deal with:
  - Failures of various kinds, such as hardware failures and system crashes
  - Concurrent execution of multiple transactions

Requirements in Transactions

- Atomicity Requirement
  - If the transaction ***fails*** after step 3 and before step 6, money will be "lost" leading to an inconsistent database state
    - Failure could be due to software or hardware
  - The system should ensure that updates of a partially executed transaction are not reflected in the database

E.g., transaction to transfer $50 from account A to account B:

```
1. read(A)
2. A := A - 50
3. write(A)
4. read(B)
5. B := B + 50
6. write(B)
```

Requirements in Transactions

- Durability Requirement
  - Once the user has been notified that the transaction has completed (i.e., the transfer of the $50 has taken place), the updates to the database by the transaction **must persist** even if there are software or hardware failures.

Requirements in Transactions

- Consistency Requirement
  - Explicitly specified integrity constraints such as primary keys and foreign keys
  - Implicit integrity constraints
    - e.g., sum of balances of all accounts
      - In the example: The sum of A and B is unchanged by the execution of the transaction

E.g., transaction to transfer $50 from account A to account B:

```
1. read(A)
2. A := A - 50
3. write(A)
4. read(B)
5. B := B + 50
6. write(B)
```

Requirements in Transactions

- Isolation Requirement
  - If between steps 3 and 6, another transaction **T2** is allowed to access the partially updated database, it will see an inconsistent database
    - The sum A + B will be less than it should be

```
T1                              T2
1.  read(A)
2.  A := A – 50
3.  write(A)
                                    read(A), read(B), print(A+B)

4.  read(B)
5.  B := B + 50
6.  write(B)
```

  - Isolation can be ensured trivially by running transactions serially, that is, one after the other
    - However, executing multiple transactions concurrently has significant benefits

ACID Properties

- A transaction is a unit of program execution that accesses and possibly updates various data items
  - To preserve the integrity of data the database system must ensure:

Atomicity: Either all operations of the transaction are properly reflected in the database, or none are.

Consistency: Execution of a transaction in isolation preserves the consistency of the database.

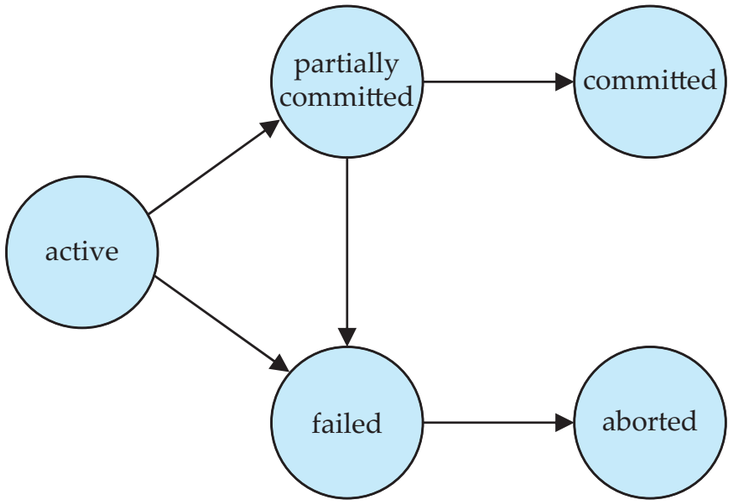
Isolation: Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.

- That is, for every pair of transactions T_i and T_j, it appears to T_i that either T_j, finished execution before T_i started, or T_j started execution after T_i finished.

Durability: After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

Transaction State

- Active
  - The initial state; the transaction stays in this state while it is executing
- Partially committed
  - After the final statement has been executed.
- Failed
  - After the discovery that normal execution can no longer proceed.
- Aborted – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
  - Restart the transaction
    - Can be done only if no internal logical error
  - Kill the transaction
- Committed
  - After successful completion.



Concurrent Executions

- Multiple transactions are allowed to run concurrently in the system. Advantages are:
  - Increased processor and disk utilization, leading to better transaction throughput
    - E.g., one transaction can be using the CPU while another is reading from or writing to the disk
  - Reduced average response time for transactions: short transactions need not wait behind long ones.
- Concurrency control schemes – mechanisms to achieve isolation
  - That is, to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

Schedules

- **Schedule** – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed
  - A schedule for a set of transactions must consist of all instructions of those transactions
  - Must preserve the order in which the instructions appear in each individual transaction
- A transaction that successfully completes its execution will have a commit instructions as the last statement
  - By default, transaction assumed to execute commit instruction as its last step
- A transaction that fails to successfully complete its execution will have an abort instruction as the last statement

Schedule 1

- Let T_1 transfer \$50 from A to B, and T_2 transfer 10% of the balance from A to B.
  - A *serial schedule* in which T_1 is followed by T_2:

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

Schedule 2

- A serial schedule where T_2 is followed by T_1

T_1	T_2
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	

Schedule 3

- Let T_1 and T_2 be the transactions defined previously
  - The following schedule is not a serial schedule, but it is *equivalent* to Schedule 1
    - In Schedules 1, 2 and 3, the sum A + B is preserved.

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit

Schedule 4

- The following concurrent schedule does not preserve the value of (A + B)

T_1	T_2
read (A)	
$A := A - 50$	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit

Serializability

- Basic Assumption:
  - Each transaction preserves database consistency
  - Thus, serial execution of a set of transactions preserves database consistency

- A (possibly concurrent) schedule is **serializable** if it is equivalent to a serial schedule.
  - Different forms of schedule equivalence give rise to the notions of:
    - 1. Conflict serializability
    - 2. View serializability

Simplified View of Transactions

- We ignore operations other than **read** and **write** instructions
- We assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes.
- Our simplified schedules consist of only **read** and **write** instructions.

Conflicting Instructions

- Instructions I_i and I_j of transactions T_i and T_j respectively, **conflict** if and only if there exists some item Q accessed by both I_i and I_j, and at least one of these instructions wrote Q.
  - 1. I_i = **read**(Q), I_j = **read**(Q). I_i and I_j don't conflict
  - 2. I_i = **read**(Q), I_j = **write**(Q). They conflict.
  - 3. I_i = **write**(Q), I_j = **read**(Q). They conflict
  - 4. I_i = **write**(Q), I_j = **write**(Q). They conflict

- Intuitively, a conflict between I_i and I_j forces a (logical) temporal order between them.
  - If I_i and I_j are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.

Conflict Serializability

- If a schedule *S* can be transformed into a schedule *S'* by a series of swaps of non-conflicting instructions, we say that *S* and *S'* are **conflict equivalent**.

- We say that a schedule *S* is **conflict serializable** if it is conflict equivalent to a serial schedule

Conflict Serializability

- Schedule 3 can be transformed into Schedule 6, a serial schedule where T_2 follows T_1
  - … by series of swaps of non-conflicting instructions
  - Therefore, Schedule 3 is *conflict serializable*.

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Schedule 3

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

Schedule 6

Conflict Serializability

- Example of a schedule that is not conflict serializable:

T_3	T_4
read (Q)	
	write (Q)
write (Q)	

  - We are unable to swap instructions in the above schedule to obtain either the serial schedule < T_3, T_4 >, or the serial schedule < T_4, T_3 >.

View Serializability *

- Let *S* and *S'* be two schedules with the same set of transactions. *S* and *S'* are **view equivalent** if the following three conditions are met, for each data item *Q*,
  - If in schedule S, transaction T_i reads the initial value of *Q*, then in schedule *S'* also transaction T_i must read the initial value of *Q*.
  - If in schedule S transaction T_i executes **read**(*Q*), and that value was produced by transaction T_j (if any), then in schedule *S'* also transaction T_i must read the value of *Q* that was produced by the same **write**(Q) operation of transaction T_j .
  - The transaction (if any) that performs the final **write**(*Q*) operation in schedule *S* must also perform the final **write**(*Q*) operation in schedule *S'*.
- As can be seen, view equivalence is also based purely on **reads** and **writes** alone.

View Serializability *

- A schedule *S* is **view serializable** if it is view equivalent to a serial schedule.

- Every conflict serializable schedule is also view serializable.
- Below is a schedule which is view-serializable but *not* conflict serializable.

T_{27}	T_{28}	T_{29}
read (Q)		
	write (Q)	
write (Q)		
		write (Q)

- What serial schedule is above equivalent to?
- Every view serializable schedule that is not conflict serializable has **blind writes.**

Testing for Serializability

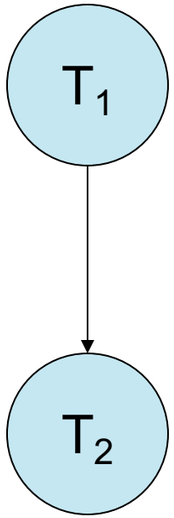
- Consider some schedule of a set of transactions T_1, T_2, ..., T_n
- Precedence graph
  - A direct graph where the vertices are the transactions (names).
  - We draw an arc from T_i to T_j if the two transactions **conflict**
    - which means, in the schedule S, T_i must appear earlier than T_j
  - We may label the arc by the item that was accessed.

Conflict – At least one of the following situations exists for a data item Q:

- **T_i: write(Q) -> T_j: read(Q)**
- **T_i: read(Q) -> T_j: write(Q)**
- **T_i: write(Q) -> T_j: write(Q)**

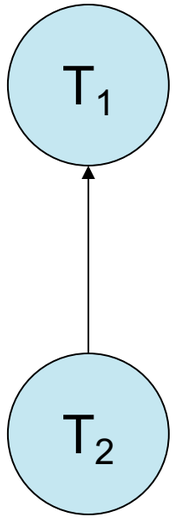
Testing for Serializability

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit



Schedule 1

T_1	T_2
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	

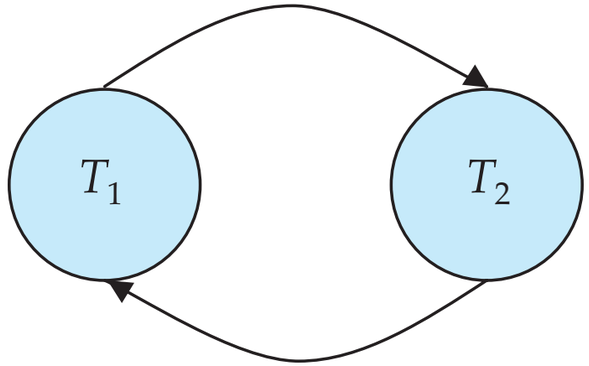


Schedule 2

Testing for Serializability

T_1	T_2
read (A)	
$A := A - 50$	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit

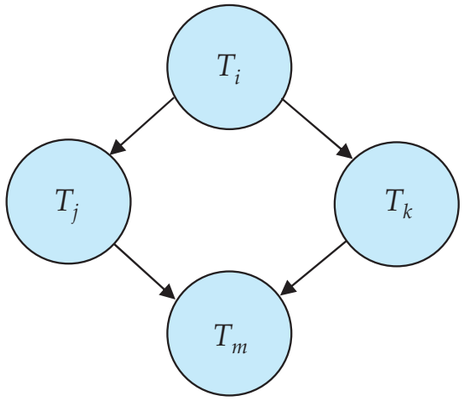
Schedule 4



Testing for Serializability

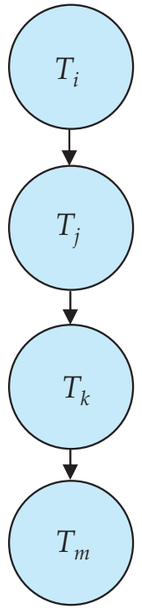
- A schedule is conflict serializable if and only if its precedence graph is acyclic.

Cycle-detection: Cycle-detection algorithms exist which take order n^2 time, where n is the number of vertices in the graph.

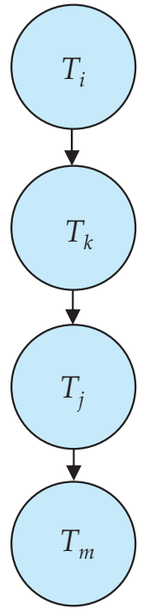


(a)

- If precedence graph is acyclic, the serializability order can be obtained by a *topological sorting* of the graph
  - E.g., The topological order of (a) can be (b) and (c)



(b)



(c)

Recoverable Schedules

- Need to address the effect of transaction failures on concurrently running transactions.

- **Recoverable schedule** — if a transaction T_j reads a data item previously written by a transaction T_i, then the commit operation of T_i appears before the commit operation of T_j.
- The following schedule is not recoverable

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	

  - If T_8 should abort, T_9 would have read (and possibly shown to the user) an inconsistent database state. Hence, database must ensure that schedules are recoverable.

Weak Levels of Consistency

- Some applications are willing to live with weak levels of consistency, allowing schedules that are not serializable
  - E.g., a read-only transaction that wants to get an approximate total balance of all accounts
    - Such transactions need not be serializable with respect to other transactions
  - Purpose: Tradeoff accuracy for performance

Levels of Consistency (in SQL-92)

- Serializable (Strongest)
  - Default
- Repeatable read — only committed records to be read.
  - Repeated reads of same record must return same value.
  - However, a transaction may not be serializable – it may find some records inserted by a transaction but not find others.
- Read committed — only committed records can be read.
  - Successive reads of record may return different (but committed) values.
- Read uncommitted (Weakest) — even uncommitted records may be read.

Levels of Consistency

- Lower degrees of consistency useful for gathering approximate information about the database
- Warning: some database systems do not ensure serializable schedules by default
  - E.g., Oracle (and PostgreSQL prior to version 9) by default support a level of consistency called snapshot isolation (not part of the SQL standard)

- Warning 2: All SQL-92 consistency levels infer that dirty writes are prohibited
  - Dirty write - when one transaction overwrites a value that has previously been written by another still in-flight transaction

Storage 1

Physical Storage System

Physical Storage System

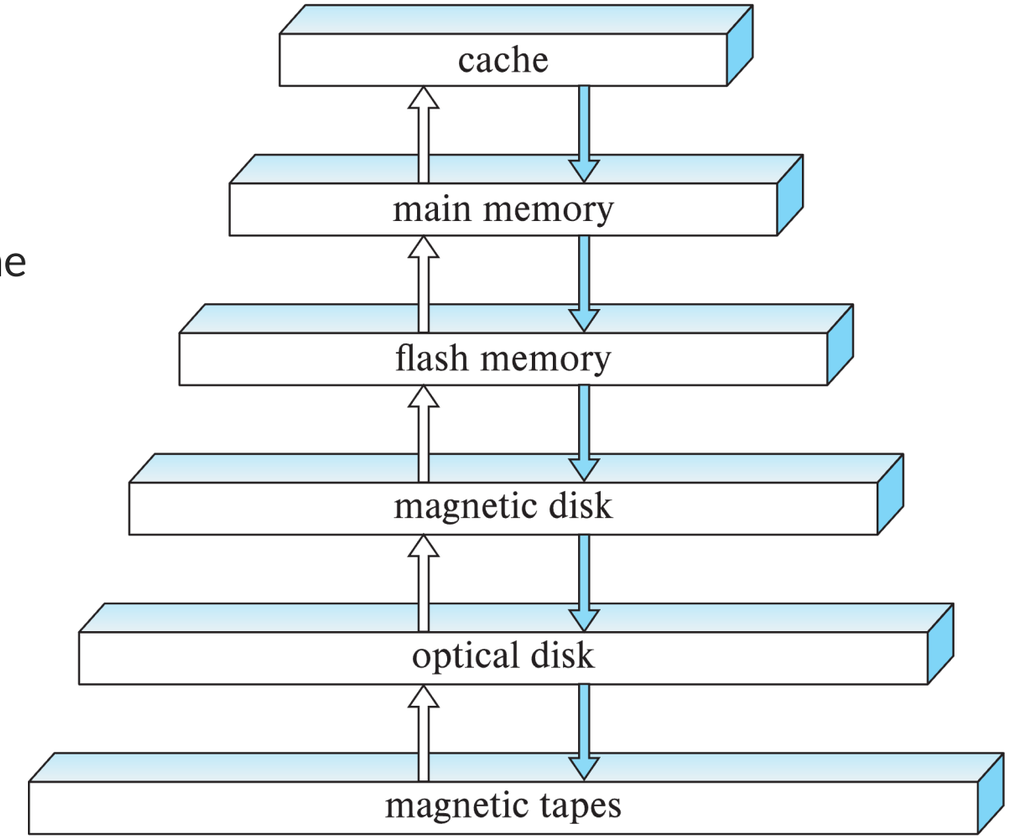
- The hardware where data is recorded

Classification of Physical Storage Media

- Can differentiate storage into:
  - Volatile storage: loses contents when power is switched off
  - Non-volatile storage:
    - Contents persist even when power is switched off.
    - Includes secondary and tertiary storage, as well as batter-backed up main-memory.
- Factors affecting choice of storage media include
  - Speed with which data can be accessed
  - Cost per unit of data
  - Reliability

Storage Hierarchy

- Primary storage
  - Fastest media but volatile (cache, main memory).
- Secondary storage
  - Next level in hierarchy, non-volatile, moderately fast access time
    - … also called on-line storage
  - E.g., flash memory, magnetic disks
- Tertiary storage
  - Lowest level in hierarchy, non-volatile, slow access time
    - … also called off-line storage and used for archival storage
  - e.g., magnetic tape, optical storage
    - Magnetic tape
      - Sequential access, 1 to 12 TB capacity
      - A few drives with many tapes
      - Juke boxes with petabytes (1000's of TB) of storage

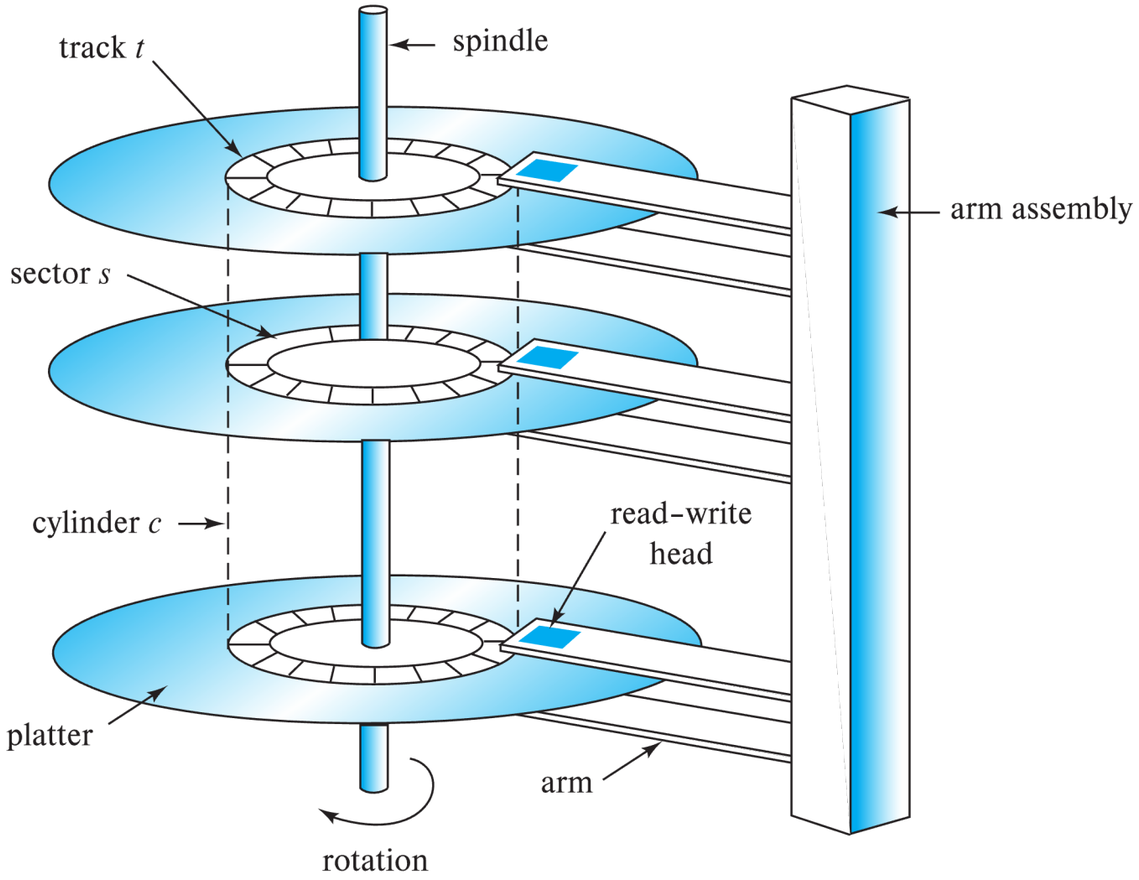


Storage Interfaces

- Disk interface standards families
  - SATA (Serial ATA)
    - SATA 3 supports data transfer speeds of up to 6 gigabits/sec
  - SAS (Serial Attached SCSI)
    - SAS Version 3 supports 12 gigabits/sec
  - NVMe (Non-Volatile Memory Express) interface
    - Works with PCIe connectors to support lower latency and higher transfer rates
    - Supports data transfer rates of up to 24 gigabits/sec

- Disks usually connected directly to computer system, however…
  - In Storage Area Networks (SAN), a large number of disks are connected by a high-speed network to a number of servers
  - In Network Attached Storage (NAS) networked storage provides a file system interface using networked file system protocol, instead of providing a disk system interface

Magnetic Hard Disk Mechanism



Schematic diagram of magnetic disk drive

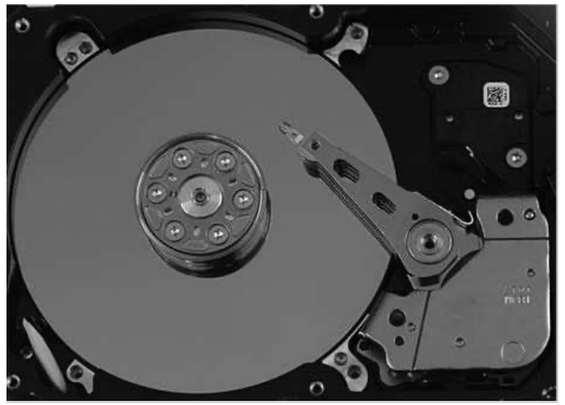


Photo of a magnetic disk drive

Magnetic Hard Disk Mechanism

- Read-write head
- Surface of platter divided into circular tracks
  - Over 50K-100K tracks per platter on typical hard disks
- Each track is divided into sectors.
  - A sector is the smallest unit of data that can be read or written.
  - Sector size typically 512 bytes (modern OS requires 4KB)
  - Typical sectors per track: 500 to 1000 (on inner tracks) to 1000 to 2000 (on outer tracks)
- To read/write a sector
  - Disk arm swings to position head on right track
  - Platter spins continually; data is read/written as sector passes under head
- Head-disk assemblies
  - Multiple disk platters on a single spindle (1 to 5 usually)
  - One head per platter, mounted on a common arm.
- Cylinder i consists of ith track of all the platters

Magnetic Hard Disk Mechanism

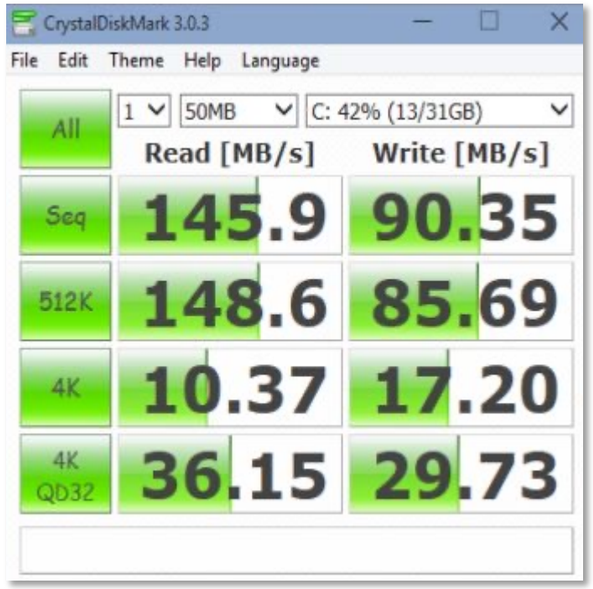
- Disk controller
  - Interfaces between the computer system and the disk drive hardware.
    - Accepts high-level commands to read or write a sector
    - Initiates actions such as moving the disk arm to the right track and actually reading or writing the data
    - Computes and attaches checksums to each sector to verify that data is read back correctly
      - If data is corrupted, with very high probability stored checksum won't match recomputed checksum
    - Ensures successful writing by reading back sector after writing it
    - Performs remapping of bad sectors

Performance Measures of Disks

- Access time: The time it takes from when a read or write request is issued to when data transfer begins. Consists of:
  - Seek time – time it takes to reposition the arm over the correct track.
    - Average seek time is 1/2 the worst case seek time.
      - Would be 1/3 if all tracks had the same number of sectors, and we ignore the time to start and stop arm movement
    - 4 to 10 milliseconds on typical disks
  - Rotational latency – time it takes for the sector to be accessed to appear under the head.
    - 4 to 11 milliseconds on typical disks (5400 to 15000 r.p.m.)
    - Average latency is 1/2 of the above latency.
  - Overall latency is 5 to 20 msec depending on disk model
- Data-transfer rate: The rate at which data can be retrieved from or stored to the disk.
  - 25 to 200 MB per second max rate, lower for inner tracks

Performance Measures of Disks

- Disk block is a logical unit for storage allocation and retrieval
  - 4 to 16 kilobytes typically
    - Smaller blocks: more transfers from disk
    - Larger blocks: more space wasted due to partially filled blocks
- Sequential access pattern
  - Successive requests are for successive disk blocks
  - Disk seek required only for first block
- Random access pattern
  - Successive requests are for blocks that can be anywhere on disk
  - Each access requires a seek
  - Transfer rates are low since a lot of time is wasted in seeks
- I/O operations per second (IOPS)
  - Number of random block reads that a disk can support per second
  - 50 to 200 IOPS on current generation magnetic disks



Performance Measures of Disks

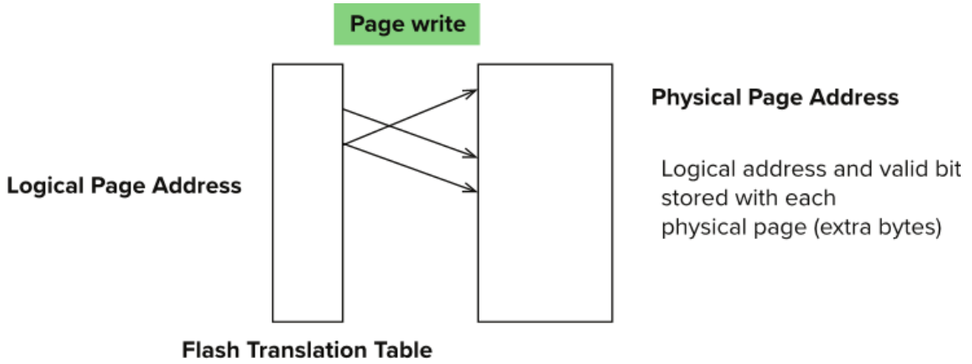
- Mean time to failure (MTTF) – the average time the disk is expected to run continuously without any failure.
  - Typically, 3 to 5 years
  - Probability of failure of new disks is quite low, corresponding to a "theoretical MTTF" of 500,000 to 1,200,000 hours for a new disk
    - E.g., an MTTF of 1,200,000 hours for a new disk means that given 1000 relatively new disks, on an average one will fail every 1200 hours
  - MTTF decreases as disk ages

Flash Storage

- NOR flash vs NAND flash
- NAND flash
  - Used widely for storage, cheaper than NOR flash
  - Requires page-at-a-time read (page: 512 bytes to 4 KB)
    - 20 to 100 microseconds for a page read
    - Not much difference between sequential and random read
  - Page can only be written once
    - Must be erased to allow rewrite
- Solid state disks (SSD)
  - Use standard block-oriented disk interfaces, but store data on multiple flash storage devices internally
  - Transfer rate of up to 500 MB/sec using SATA, and up to 3 GB/sec using NVMe PCIe

Flash Storage

- Erase happens in units of erase block
  - Takes 2 to 5 millisecs
  - Erase block typically 256 KB to 1 MB (128 to 256 pages)
- Remapping of logical page addresses to physical page addresses avoids waiting for erase
- Flash translation table tracks mapping
  - Also stored in a label field of flash page
  - Remapping carried out by flash translation layer



- After 100,000 to 1,000,000 erases, erase block becomes unreliable and cannot be used
  - Wear leveling

SSD Performance Metrics

- Random reads/writes per second
  - Typical 4 KB reads: 10,000 reads per second (10,000 IOPS)
  - Typical 4KB writes: 40,000 IOPS
  - SSDs support parallel reads
    - Typical 4KB reads:
      - 100,000 IOPS with 32 requests in parallel (QD-32) on SATA
      - 350,000 IOPS with QD-32 on NVMe PCIe
    - Typical 4KB writes:
      - 100,000 IOPS with QD-32, even higher on some models
- Data transfer rate for sequential reads/writes
  - 400 MB/sec for SATA3, 2 to 3 GB/sec using NVMe PCIe

- Hybrid disks: Combine small amount of flash cache with larger magnetic disk

Storage Class Memory

- 3D-XPoint memory technology pioneered by Intel
- Available as Intel Optane
  - SSD interface shipped from 2017
    - Allows lower latency than flash SSDs
  - Non-volatile memory interface announced in 2018
    - Supports direct access to words, at speeds comparable to main-memory speeds

Magnetic Tapes

- Hold large volumes of data and provide high transfer rates
  - Few GB for DAT (Digital Audio Tape) format, 10-40 GB with DLT (Digital Linear Tape) format, 100 GB+ with Ultrium format, and 330 GB with Ampex helical scan format
  - Transfer rates from few to 10s of MB/s
- Tapes are cheap, but cost of drives is very high
- Very slow access time in comparison to magnetic and optical disks
  - limited to sequential access.
  - Some formats (Accelis) provide faster seek (10s of seconds) at cost of lower capacity
- Used mainly for backup, for storage of infrequently used information, and as an off-line medium for transferring information from one system to another.
- Tape jukeboxes used for very large capacity storage
  - Multiple petabyes (1015 bytes)

RAID

- RAID: Redundant Arrays of Independent Disks
  - Disk organization techniques that manage a large numbers of disks, providing a view of a single disk of
    - high capacity and high speed by using multiple disks in parallel,
    - high reliability by storing data redundantly, so that data can be recovered even if a disk fails
- The chance that some disk out of a set of N disks will fail is much higher than the chance that a specific single disk will fail.
  - E.g., a system with 100 disks, each with MTTF of 100,000 hours (approx. 11 years), will have a system MTTF of 1000 hours (approx. 41 days)
  - Techniques for using redundancy to avoid data loss are critical with large numbers of disks

Improvement of Reliability via Redundancy

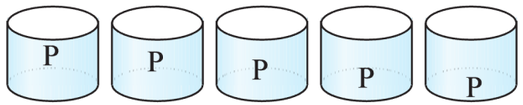
- Redundancy – store extra information that can be used to rebuild information lost in a disk failure
  - E.g., Mirroring (or shadowing)
    - Duplicate every disk. Logical disk consists of two physical disks.
    - Every write is carried out on both disks
      - Reads can take place from either disk
    - If one disk in a pair fails, data still available in the other
      - Data loss would occur only if a disk fails, and its mirror disk also fails before the system is repaired
      - Probability of combined event is very small
      - Except for dependent failure modes such as fire or building collapse or electrical power surges
  - Mean time to data loss depends on mean time to failure, and mean time to repair
    - E.g., MTTF of 100,000 hours, mean time to repair of 10 hours gives mean time to data loss of 500*106 hours (or 57,000 years) for a mirrored pair of disks (ignoring dependent failure modes)

RAID Levels

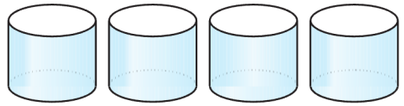
- Schemes to provide redundancy at lower cost by using disk striping combined with parity bits
  - Different RAID organizations, or RAID levels, have differing cost, performance and reliability characteristics

  - RAID 0: Block striping; non-redundant
  - RAID 1: Mirrored disks with block striping
  - RAID 10: Combination of striping and mirroring

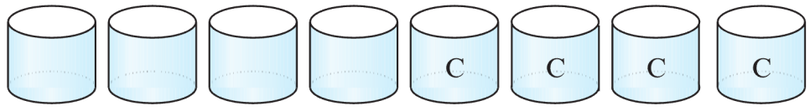
  - RAID 5: Block-interleaved distributed parity
  - RAID 6: P+Q Redundancy scheme



(c) RAID 5: block-interleaved distributed parity



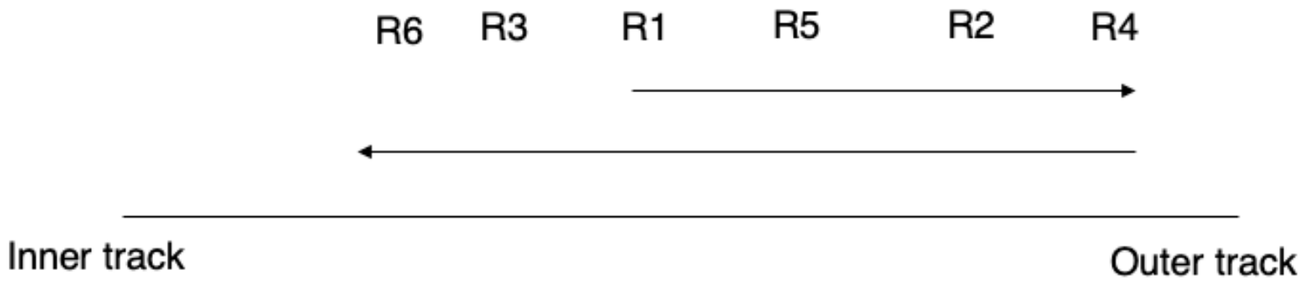
(a) RAID 0: nonredundant striping



(b) RAID 1: mirrored disks

Optimization of Disk-Block Access

- Buffering
  - In-memory buffer to cache disk blocks
- Read-ahead
  - Read extra blocks from a track in anticipation that they will be requested soon
- Disk-arm-scheduling algorithms
  - Re-order block requests so that disk arm movement is minimized
  - E.g., elevator algorithm



Optimization of Disk-Block Access

- File organization
  - Allocate blocks of a file in as contiguous a manner as possible
  - Allocation in units of extents
  - Files may get fragmented
    - E.g., if free blocks on disk are scattered, and newly created file has its blocks scattered over the disk
    - Sequential access to a fragmented file results in increased disk arm movement
    - Some systems have utilities to defragment the file system, in order to speed up file access
- Non-volatile write buffers
  - Temporarily store the written data
    - … and immediately notifies the OS that writing is completed without errors
  - Write data into the disk when idle
    - … with some optimizations

Storage 2

Data Storage Structure

File Organization

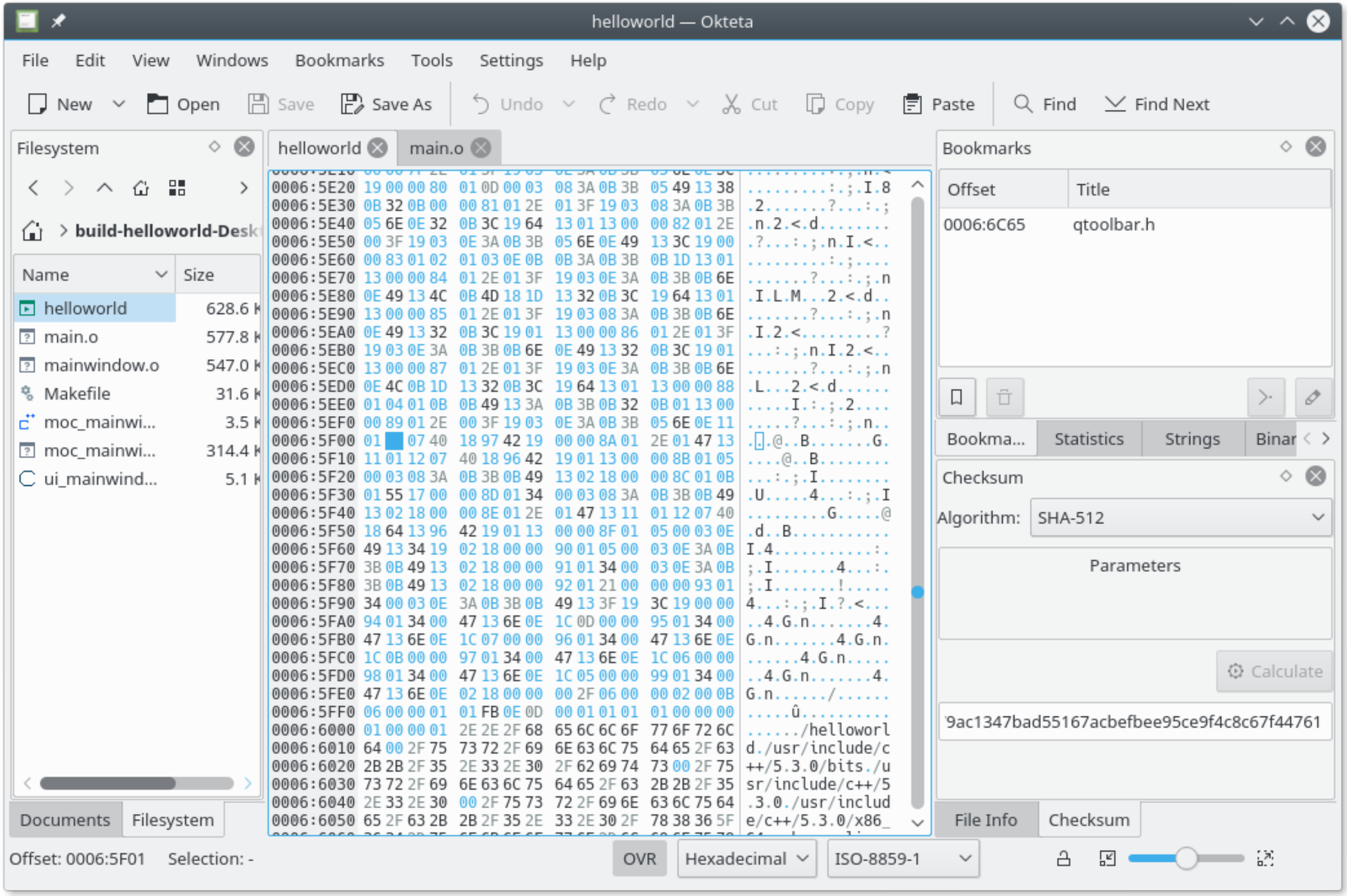
- The database is stored as a collection of files
  - Each file is a sequence of records
  - A record is a sequence of fields.
- One approach
  - Assume record size is fixed
  - Each file has records of one particular type only
  - Different files are used for different relations

  * This case is easiest to implement; will consider variable length records later

- We assume that records are smaller than a disk block

File Organization

- Bitmap of a file



https://apps.kde.org/okteta/

File Organization

- Goals: Time and Space
  - Support CURD operations as fast as possible
  - Save storage space as much as possible

  - Maintain data integrity

Fixed-Length Records

- Simple approach:
  - Store record i starting from byte $n*(i-1)$, where n is the size of each record
  - Record access is simple but records may cross blocks
    - Modification: do not allow records to cross block boundaries

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Fixed-Length Records

- Deletion of record i
  - Way #1: move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

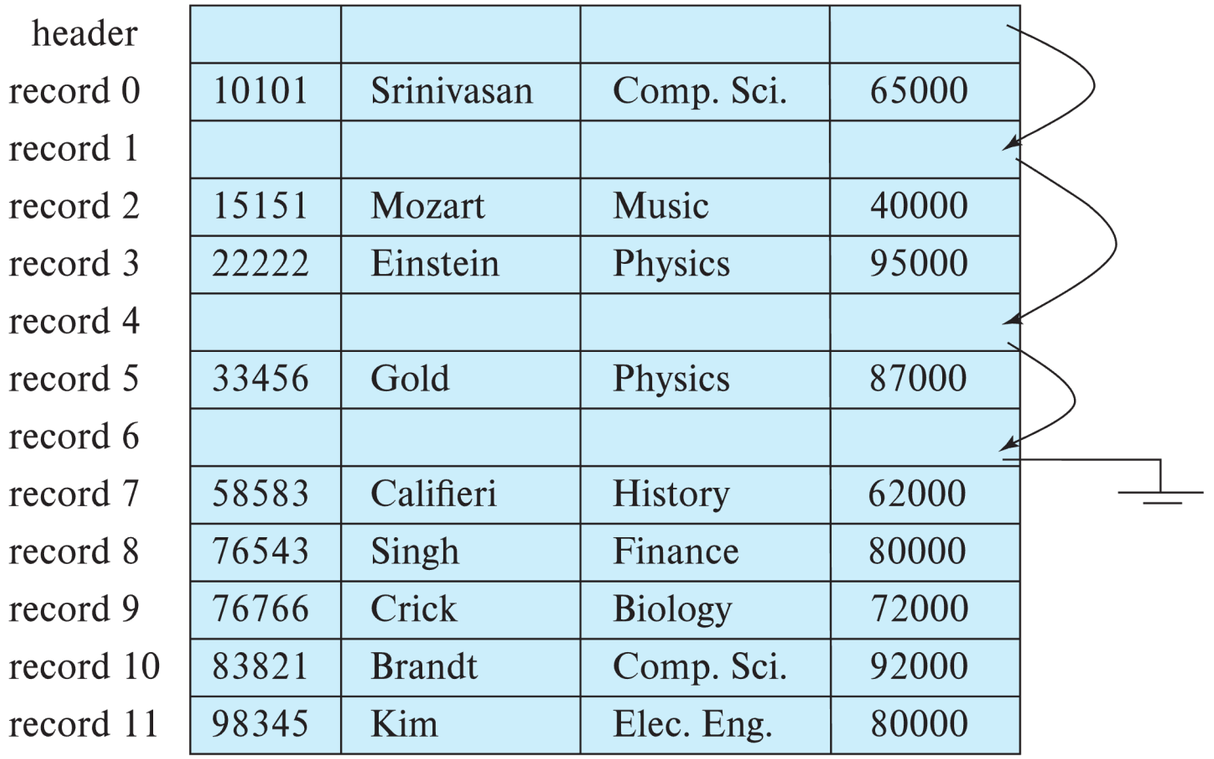
Fixed-Length Records

- Deletion of record i
  - Way #2: move record n to i
    - Record 3 is removed and replaced by record 11

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

Fixed-Length Records

- Deletion of record i
  - Way #3: Do not move records, but link all free records on a *free list*



header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

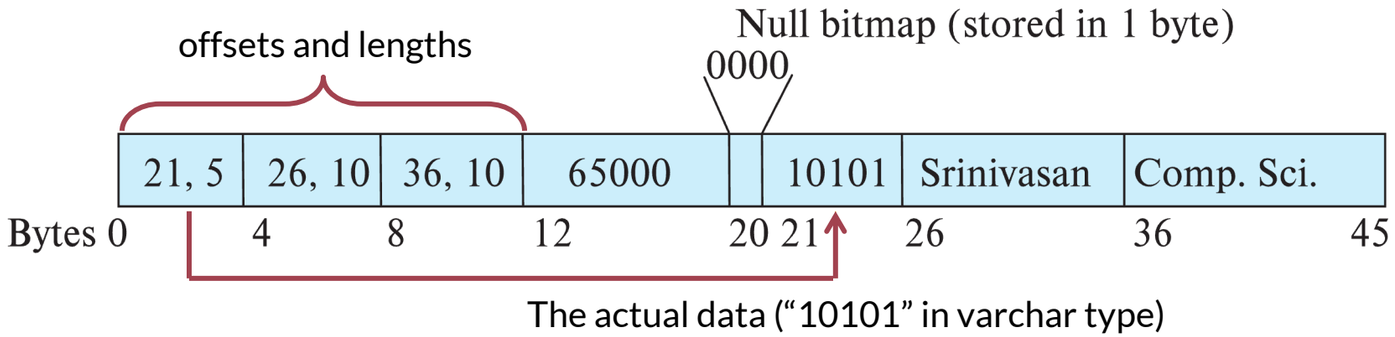
Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields such as strings (`varchar`)
  - Record types that allow repeating fields (used in some older data models).

- Problem with variable-length records
  - How can we retrieve the data in an easy way without wasting too much space
    - `varchar(1000)`: do we really need to allocate 1000 bytes for this field, even if most of the actual data items only costs less than 10 bytes?

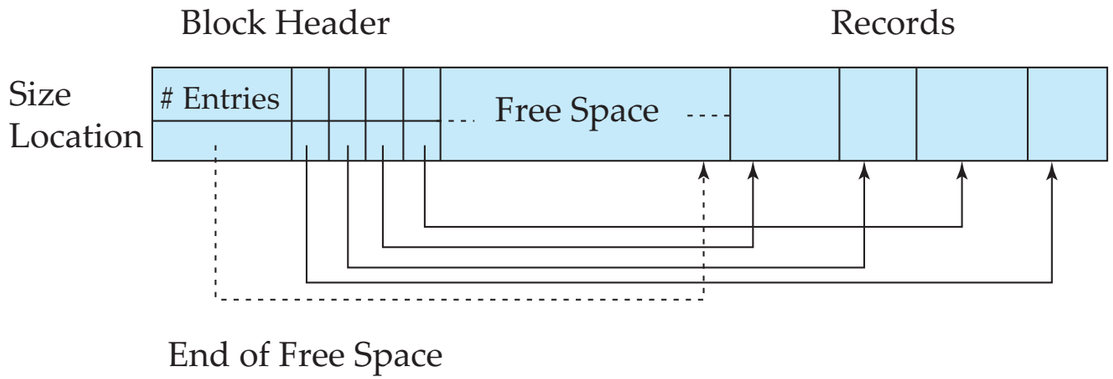
Variable-Length Records

- Attributes are stored in order
  - Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
  - Null values represented by null-value bitmap



Variable-Length Records

- Slotted page header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record
  - Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.
- Pointers should not point directly to record — instead they should point to the entry for the record in header.



Storing Large Objects

- E.g., blob/clob types
- Records must be smaller than pages
- Alternatives:
  - Store as files in file systems
  - Store as files managed by database
  - Break into pieces and store in multiple tuples in separate relation
    - PostgreSQL TOAST

Organization of Records in Files

- Heap – record can be placed anywhere in the file where there is space
- Sequential – store records in sequential order, based on the value of the search key of each record
- Multitable clustering file organization
  - Records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimize I/O
- B+-tree file organization
  - Ordered storage even with inserts/deletes
- Hashing – a hash function computed on search key; the result specifies in which block of the file the record should be placed

Heap File Organization

- Records can be placed anywhere in the file where there is free space
  - Records usually do not move once allocated

- Important to be able to efficiently find free space within file
- Free-space map
  - Array with 1 entry per block. Each entry is a few bits to a byte, and records fraction of block that is free
  - In example below, 3 bits per block, value divided by 8 indicates fraction of block that is free

4	2	1	4	7	3	6	5	1	2	0	1	1	0	5	6

  - Can have second-level free-space map
    - In example below, each entry stores maximum from 4 entries of first-level free-space map

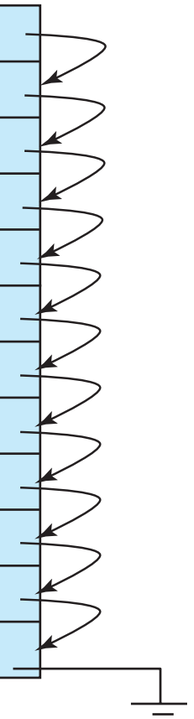
4	7	2	6

- Free space map written to disk periodically, OK to have wrong (old) values for some entries (will be detected and fixed)

Sequential File Organization

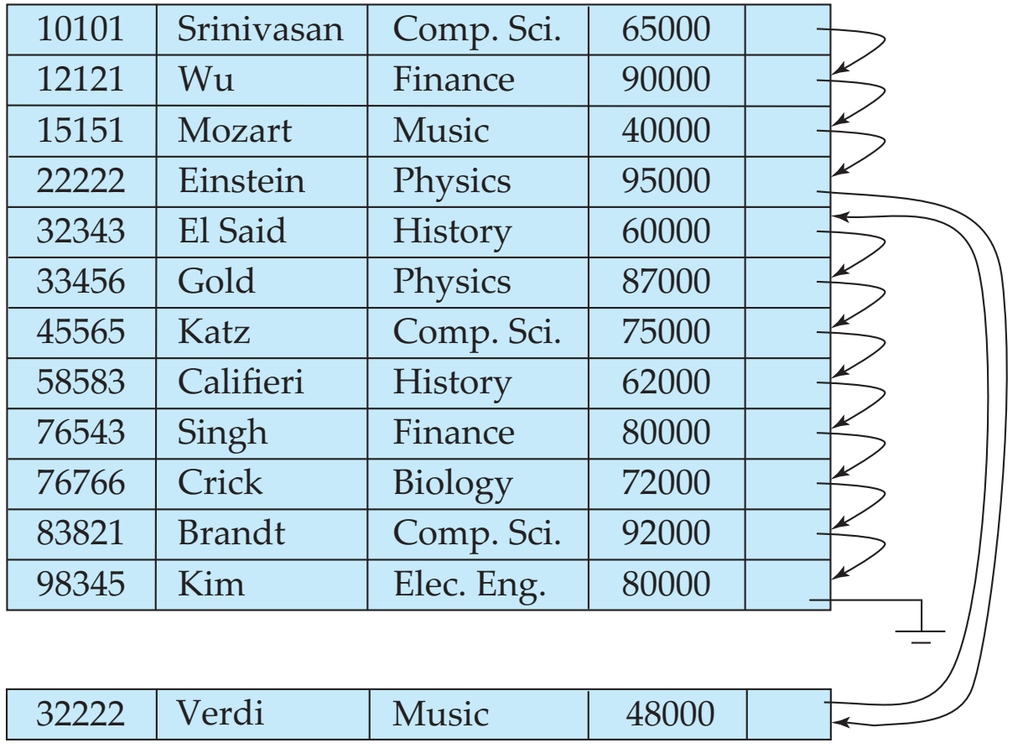
- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key

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



Sequential File Organization

- Deletion – Use pointer chains
- Insertion – Locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order



Multitable Clustering File Organization

- Store several relations in one file using a multitable clustering file organization
  - Good for queries involving:
    - *department* ⋈ *instructor*
    - or, one single department and its instructors
  - Bad for queries involving only *department*
  - Results in variable size records
  - Can add pointer chains to link records of a particular relation

department

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

Multitable clustering of department and instructor

Comp. Sci.	Taylor	100000	
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
Physics	Watson	70000	
33456	Gold	Physics	87000

Partitioning

- Table partitioning: Records in a relation can be partitioned into smaller relations that are stored separately
  - E.g., `transaction` relation may be partitioned into `transaction_2018`, `transaction_2019`, etc.
- Queries written on transaction must access records in all partitions
  - Unless query has a selection such as year=2019, in which case only one partition in needed
- Partitioning
  - Reduces costs of some operations such as free space management
  - Allows different partitions to be stored on different storage devices
    - E.g., transaction partition for current year on SSD, for older years on magnetic disk

Storage 3

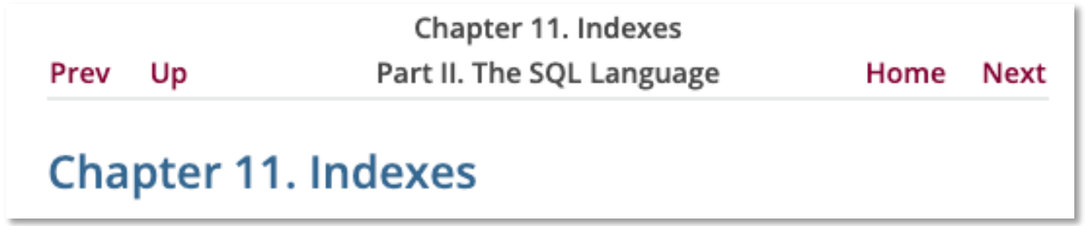
Indexing (in Database Systems)

Motivation

- Think about an example in a library:
  - How can we find a book?
    - Books are on the shelves in a sequential order
    - We had drawers where you could look for books by author, title or sometimes subject that were telling you what were the "***coordinates***" of a book.

Terminology

- Plural of index: indices, or indexes?
  - Both are correct in English
    - indices (Latin): Often used in scientific and mathematical context representing the places of an element in an array, vector, matrix, etc.
    - indexes (American English): Used in publishing for the books
  - What about database?
    - A good way: Follow the naming convention of the project or the DBMS



https://www.postgresql.org/docs/current/indexes.html

Searching for Record

- Remember searching algorithms in Data Structure?
  - Linear search
    - Scan all records from top to bottom
  - Binary search
    - Divide and conquer
    - Assumption: Records are sorted by the search key

```
1   1,12 stulyev,ru,1971,161
2   2,Al-mummia test,eg,1969,102
3   3,"Ali Zaoua, prince de la rue",ma,2000,90
4   4,Apariencias,ar,2000,94
5   5,Ardh Satya,in,1983,130
6   6,Armaan,in,2003,159
7   7,Armaan,pk,1966,
8   8,Babettes gæstebud,dk,1987,102
9   9,Banshun,jp,1949,108
10  10,Bidaya wa Nihaya,eg,1960,
11  11,Variety,us,2008,106
12  12,"Bon Cop, Bad Cop",ca,2006,
13  13,Brilliantovaja ruka,ru,1969,100
14  14,C'est arrivé près de chez vous,be,1992,95
15  15,Carlota Joaquina - Princesa do Brasil,br,1995,
16  16,Cicak-man,my,2006,107
```

Searching for Record

- Remember searching algorithms in Data Structure?
  - Linear search
    - Scan all records from top to bottom
  - Binary search
    - Divide and conquer
    - Assumption: Records are sorted by the search key
    - E.g., Find movies with IDs larger than 100 and smaller than 200

In the current storage structure, the records are sorted by `movieid`

- So, it will be easy to find a specific `movieid` with binary search

Searching for Record

- Remember searching algorithms in Data Structure?
  - Linear search
    - Scan all records from top to bottom
  - Binary search
    - Divide and conquer
    - Assumption: Records are sorted by the search key

  - However, how can we find data based on the non-sorted columns?
    - E.g., find all Chinese movies

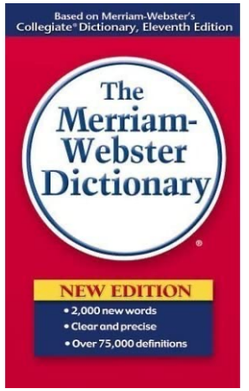
```
1,12 stulyev,ru,1971,161
2,Al-mummia test,eg,1969,102
3,"Ali Zaoua, prince de la rue",ma,2000,90
4,Apariencias,ar,2000,94
5,Ardh Satya,in,1983,130
6,Armaan,in,2003,159
7,Armaan,pk,1966,
8,Babettes gæstebud,dk,1987,102
9,Banshun,jp,1949,108
10,Bidaya wa Nihaya,eg,1960,
11,Variety,us,2008,106
12,"Bon Cop, Bad Cop",ca,2006,
13,Brilliantovaja ruka,ru,1969,100
14,C'est arrivé près de chez vous,be,1992,95
15,Carlota Joaquina - Princesa do Brasil,br,1995,
16,Cicak-man,my,2006,107
```

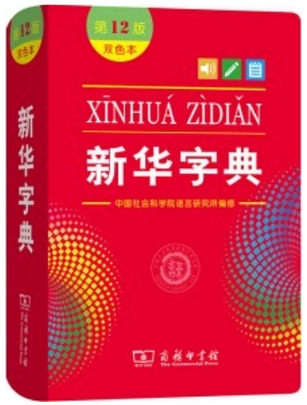
`country`

Find the rows `where country = 'cn'`

- The country codes are not sorted in the current storage structure, so the binary search algorithm cannot be used

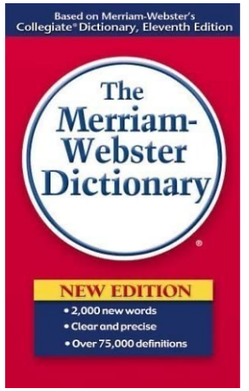
Searching for Record

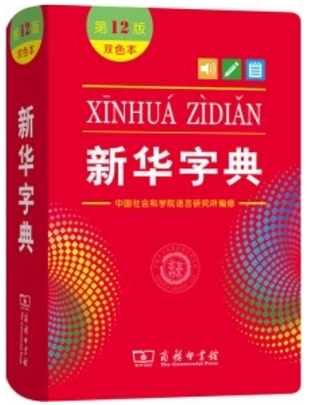




- This happens in real life too
  - English dictionary
    - The words are sorted in an alphabetical order
  - Chinese dictionary
    - The characters are sorted in the alphabetical order of Pinyin

Searching for Record





- This happens in real life too
  - English dictionary
    - The words are sorted in an alphabetical order
  - Chinese dictionary
    - The characters are sorted in the alphabetical order of Pinyin
    - However, we have other ways of looking up a character
      - Radicals (偏旁部首)
      - Number of strokes (数笔画)
      - Four-corner method (四角号码)

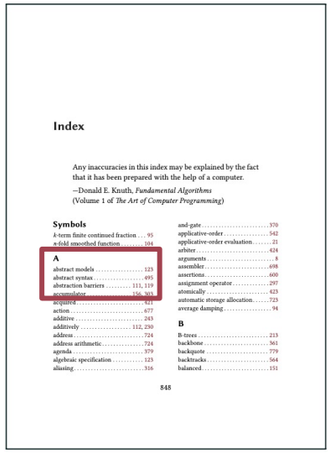
Index in Databases

- Concept
  - An **index** is a data structure which improves the efficiency of retrieving data with specific values from a database

  - Usually, indexes locate a row by a series of location indicators
    - E.g., (filename, block number, offset)

Index in Databases

- Concept
  - An **index** is a data structure which improves the efficiency of retrieving data with specific values from a database
  - Usually, indexes locate a row by a series of location indicators
    - E.g., (filename, block number, offset)
  - It is like indexes in books
    - Location indicator: (page, row)


Index

Any inaccuracies in this index may be explained by the fact that it has been prepared with the help of a computer.

—Donald E. Knuth, *Fundamental Algorithms* (Volume 1 of *The Art of Computer Programming*)

Symbols

4-term finite continued fraction . . . 95
n-fold smoothed function . . . 104

A

abstract models . . . 123
abstract syntax . . . 495
abstraction barriers . . . 111, 119
accumulator . . . 156, 303
acquired . . . 617
action . . . 677
additive . . . 243
additively . . . 112, 230
address . . . 724
address arithmetic . . . 724
agenda . . . 379
algebraic specification . . . 123
aliasing . . . 316
and-gate . . . 370
applicative-order . . . 542
applicative-order evaluation . . . 21
arbiter . . . 424
arguments . . . 8
assembler . . . 698
assertions . . . 600
assignment operator . . . 297
atomically . . . 423
automatic storage allocation . . . 723
average damping . . . 94

B

B-trees . . . 233
backbone . . . 361
backquote . . . 779
backtracks . . . 564
balanced . . . 151

848

A

abstract models 123
abstract syntax 495
abstraction barriers 111, 119
accumulator 156, 303

Index in Databases

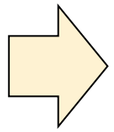
- Actually, we have been benefited from indexes off-the-shelf

  - In PostgreSQL, indexes are built automatically on columns with `primary key` or `unique` constraints

Experiment on Using Indexes

- Duplicate a table with no index

```
create table movies_no_index as select * from movies;
```



```
-- auto-generated definition
create table movies_no_index
(
    movieid       integer,
    title         varchar(100),
    country       char(2),
    year_released integer,
    runtime       integer,
    user_name     varchar(20)
);
```

Experiment on Using Indexes

- Check the performance on retrieving data
  - Significant difference between queries on the two tables

```sql
-- Query 1
explain analyze
    select *
    from movies
    where movieid > 100 and movieid < 300;

-- Query 2
explain analyze
    select *
    from movies_no_index
    where movieid > 100 and movieid < 300;
```

Query 1
(on `movies`)

```
QUERY PLAN
Bitmap Heap Scan on movies  (cost=10.32..136.35 rows=199 width=40) (actual time=0.162..0.440 rows=199 loops=1)
  Recheck Cond: ((movieid > 100) AND (movieid < 300))
  Heap Blocks: exact=6
  ->  Bitmap Index Scan on movies_pkey  (cost=0.00..10.28 rows=199 width=0) (actual time=0.136..0.136 rows=199 loops=1)
        Index Cond: ((movieid > 100) AND (movieid < 300))
Planning Time: 0.413 ms
Execution Time: 0.507 ms
```

Query 2
(on `movies_no_index`)

```
QUERY PLAN
Seq Scan on movies_no_index  (cost=0.00..217.06 rows=199 width=40) (actual time=0.039..5.075 rows=199 loops=1)
  Filter: ((movieid > 100) AND (movieid < 300))
  Rows Removed by Filter: 9005
Planning Time: 0.444 ms
Execution Time: 5.156 ms
```

Experiment on Using Indexes

- If there is no index on a column (or several columns), we can create one manually

```
-- SQL Syntax for creating indexes
create index index_name
on table_name (column_name [, ...]);
```

Index Taxonomy

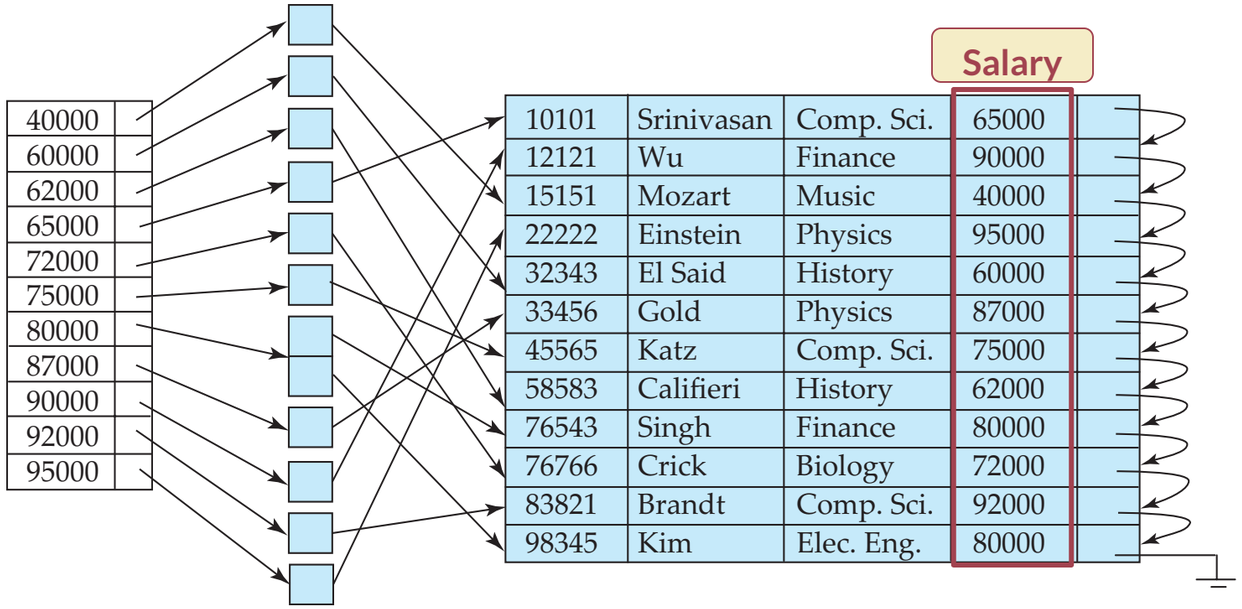
- 1) In terms of storage structure, is the index completely separated with the data records?
  - No ⇒ **Integrated index**
    - PK index in a MySQL InnoDB database
    - PK index in a SQL Server database
  - Yes ⇒ **External index**
    - Indexes in a PostgreSQL database
    - Indexes in a MySQL MyISAM database

Index Taxonomy

- 2) Does the index specify the order in which records are stored in the data file?
  - Yes ⇒ **Clustered index** (a.k.a. primary index)
  - No ⇒ **Non-clustered index** (a.k.a. secondary index)

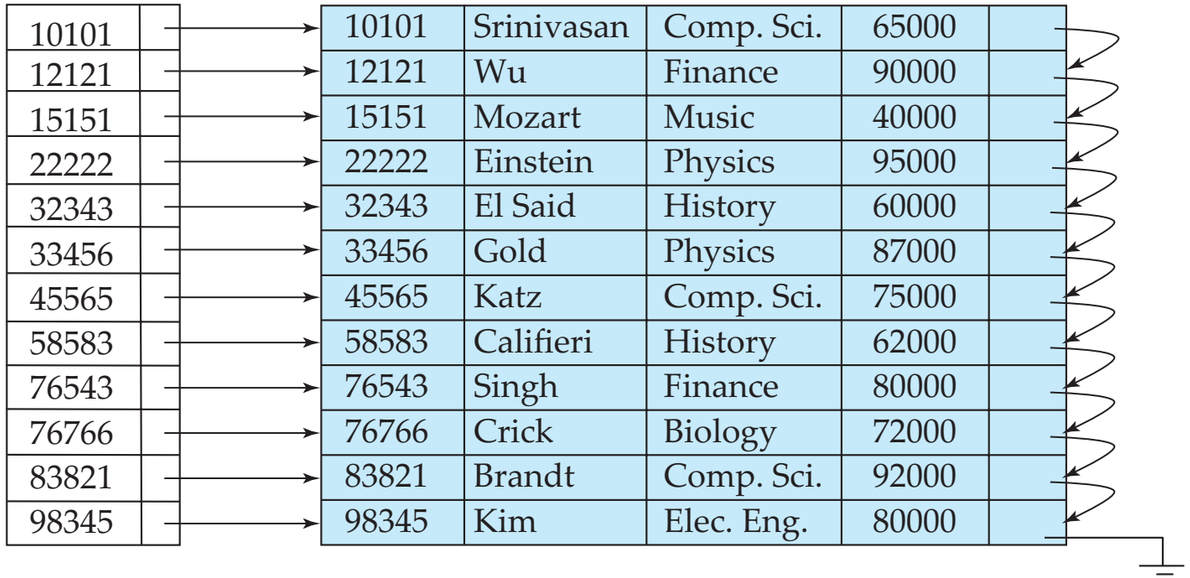
A secondary index on the column "salary"

- Index record points to a bucket that contains pointers to all the actual records with that particular search-key value
- Secondary indices have to be dense

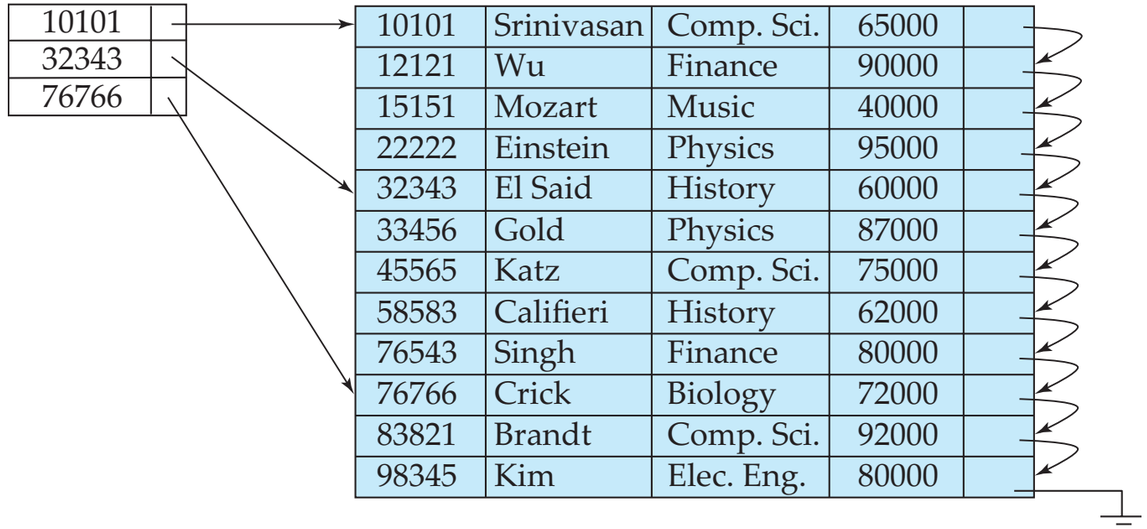


Index Taxonomy

- 3) Does every search key in the data file correspond to an index entry?
  - Yes ⇒ **Dense Index**
  - No ⇒ **Sparse Index**



Dense Index



Sparse Index

Index Taxonomy

- 4) Does the search key contain more than one attribute?
  - Yes ⇒ **Multi-key index** (Multi-column index)
  - No ⇒ **Single-key index** (Single-column index)
    - *We mainly focus on single-key index for now*

Index Implementation

- Data Structures for Indexes
  - B-tree, B+-tree
    - Very famous data structures for building indexes
  - Hash table

B-tree

- A B-tree of order m satisfies that
  - For every node, # of children = # of keys + 1
  - (**Ordered**) For a node containing n keys ($K_1 < K_2 < K_3 < \cdots < K_n$) with n+1 children (pointed by $P_0, P_1, P_2, \ldots, P_n$), any key $k_{sub\ i}$ in the sub-tree pointed by P_i satisfies that $K_i < k_{sub\ i} < K_{i+1}$
  - (**Multiway**) For an internal node, $\lceil m/2 \rceil \leq$ # of children $\leq m$
    - … except that a root node may have less than $\lceil m/2 \rceil$ children
  - (**Always balanced**) All leaves appear on the same level

B-tree

- A B-tree of order m satisfies that
  - For every node, # of children = # of keys + 1

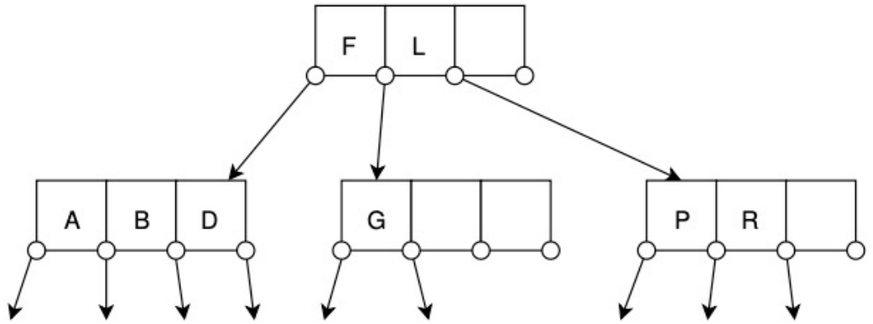


Figure: An example of B-tree (2-4 tree, a.k.a. 2-3-4 tree)

  - (**Ordered**) For a node containing n keys ($K_1 < K_2 < K_3 < \cdots < K_n$) with n+1 children (pointed by $P_0, P_1, P_2, \ldots, P_n$), any key $k_{sub\ i}$ in the sub-tree pointed by P_i satisfies that $K_i < k_{sub\ i} < K_{i+1}$
  - (**Multiway**) For an internal node, $\lceil m/2 \rceil \leq$ # of children $\leq m$
    - … except that a root node may have less than $\lceil m/2 \rceil$ children
  - (**Always balanced**) All leaves appear on the same level

B-tree

- A B-tree of order m satisfies that
  - For every node, # of children = # of keys + 1
  - (**Ordered**) For a node containing n keys ($K_1 < K_2 < K_3 < \cdots < K_n$) with n+1 children (pointed by $P_0, P_1, P_2, \ldots, P_n$), any key $k_{sub\ i}$ in the sub-tree pointed by P_i satisfies that $K_i < k_{sub\ i} < K_{i+1}$
  - (**Multiway**) For an internal node, $\lceil m/2 \rceil \leq$ # of children $\leq m$
    - … except that a root node may have less than $\lceil m/2 \rceil$ children
  - (**Always balanced**) All leaves appear on the same level

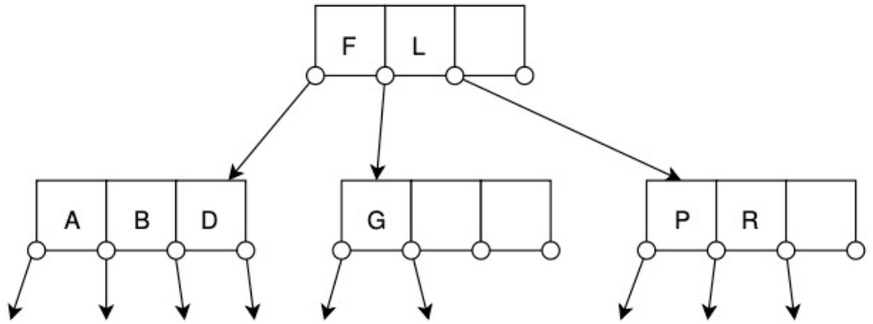


Figure: An example of B-tree (2-4 tree, a.k.a. 2-3-4 tree)

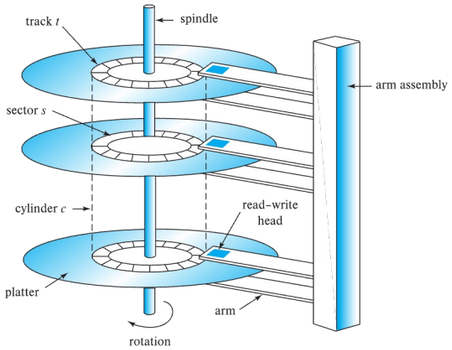
- $\lceil m/2 \rceil$ is the called the minimum branching factor (a.k.a. minimum degree) of the tree
- A B-tree of order m is usually called a "$\lceil m/2 \rceil$-m tree", like 2-3 tree, 2-4 tree, 3-5 tree, 3-6 tree, . . .
  - In practice, the order m is much larger (~100)

B-tree

- Height of a *B*-tree: $h \leq 1 + \log_{\lceil m/2 \rceil}\left(\frac{n+1}{2}\right)$
  - If we take an 50-100 tree with 1M records:
    - $h \leq 1+\log_{100/2}(1000000/2) = 4.354$ (i.e., 4 levels)

B-tree

- Height of a *B*-tree: $h \leq 1 + \log_{\lceil m/2 \rceil}\left(\frac{n+1}{2}\right)$
  - If we take an 50-100 tree with 1M records:
    - h ≤ $1+\log_{100/2}(1000000/2)$ = 4.354 (i.e., 4 levels)
  - Why do we use B-trees?
    - We can set the size of a B-tree node as the disk page size
      - i.e., m can chosen with consideration on the page size
    - The height of the tree -> Number of disk I/Os
      - The number of disk I/Os can be relatively small



Access time: 5-20ms

1ns = 10^{-6}ms

Access time: 50-70ns

Seconds:

100000

Hours:

27.7777777778

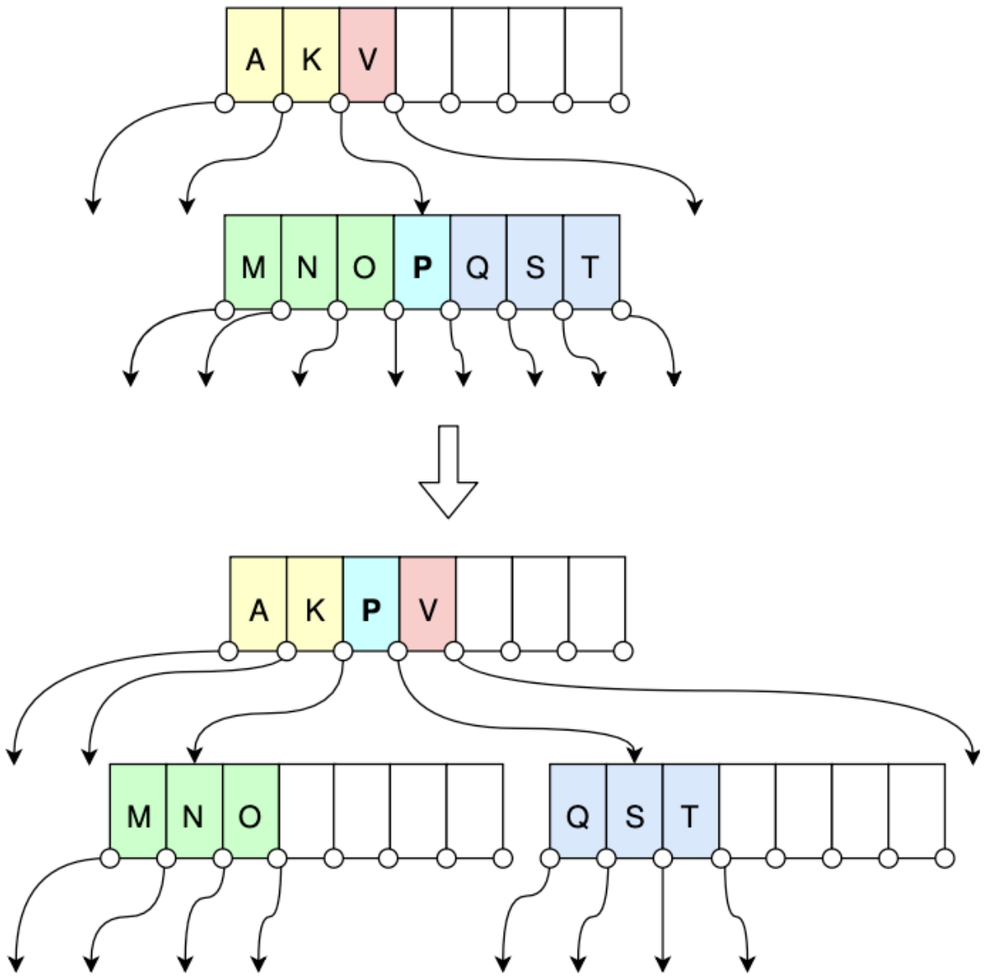
B-tree

- Tree operations:
  - Search, Insert, Delete
  - Update (Delete + Insert)

- What is special in B-tree
  - Split and merge nodes

B-tree

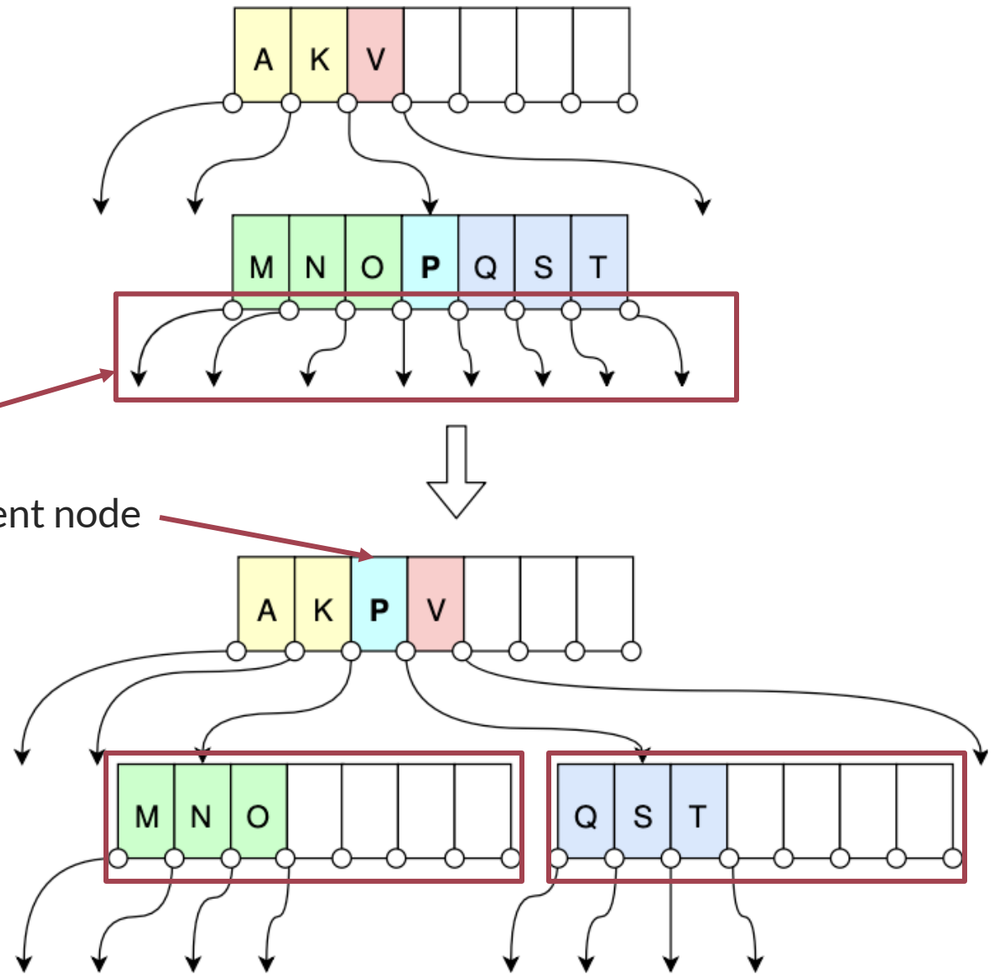
- Split a node in a B-tree
  - Example: when m=7



B-tree

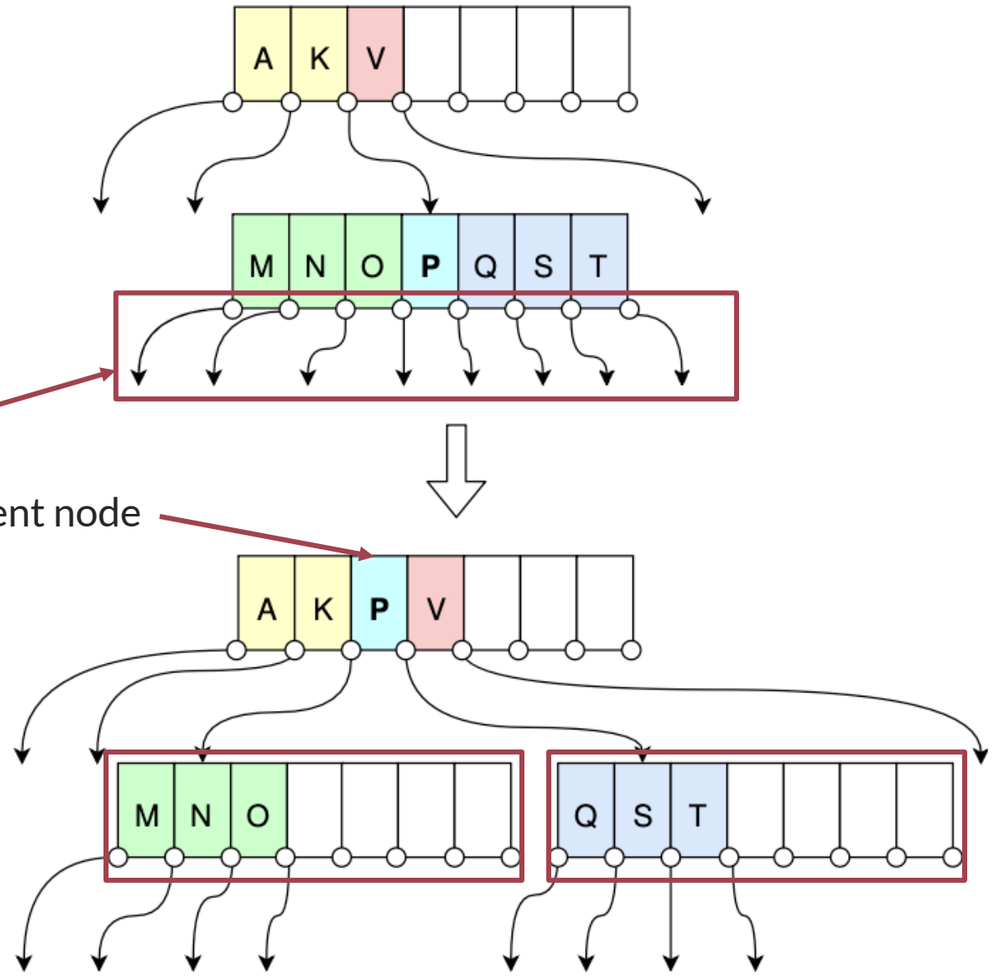
- Split a node in a B-tree
  - Example: when m=7

The number of children is larger than m (7)
- This node will be split into two nodes
- The pivot key will be elevated into the parent node



B-tree

- Split a node in a B-tree
  - Example: when m=7



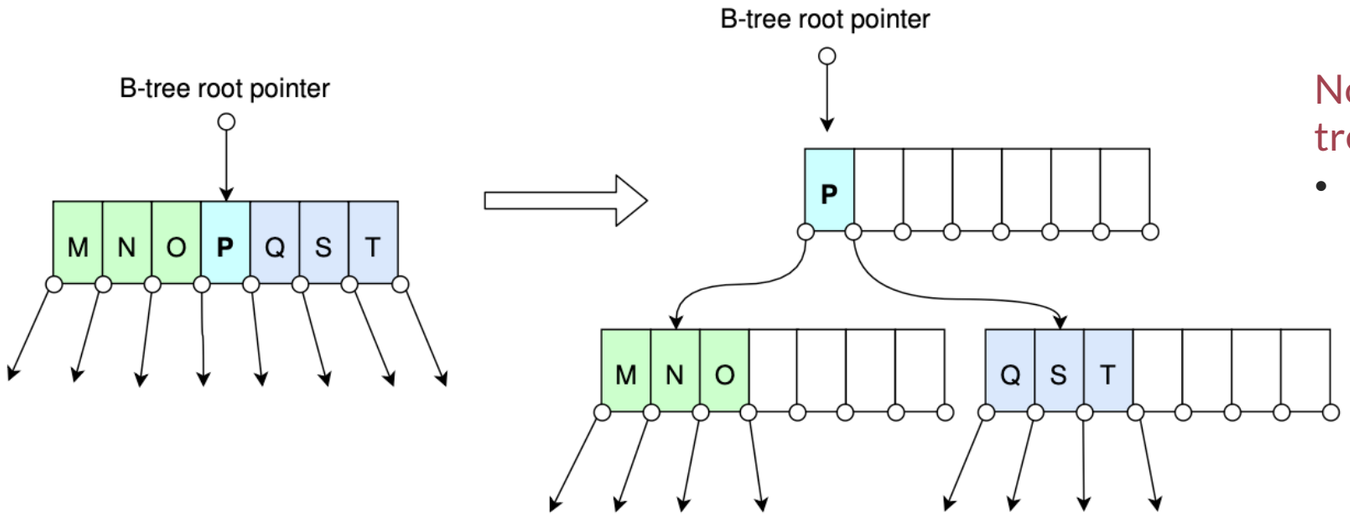
The number of children is larger than m (7)
- This node will be split into two nodes
- The pivot key will be elevated into the parent node

  - What if the parent (or even the root) node is also full?

B-tree

- Split a node in a B-tree
  - Example: when m=7

- Split the root node of the B-tree

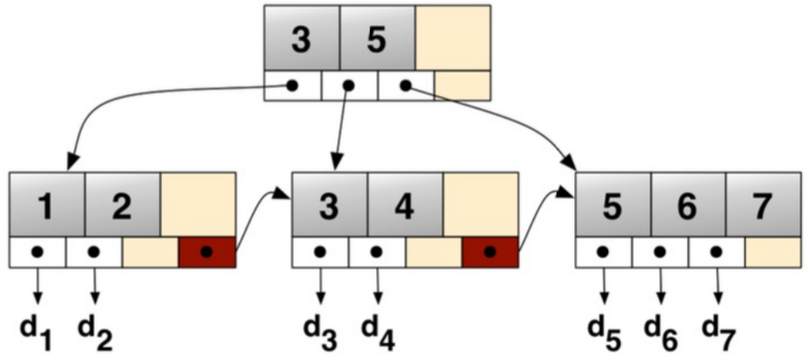


Note that the height of the B-tree is increased by 1

- This is the only way that a B-tree increases its height

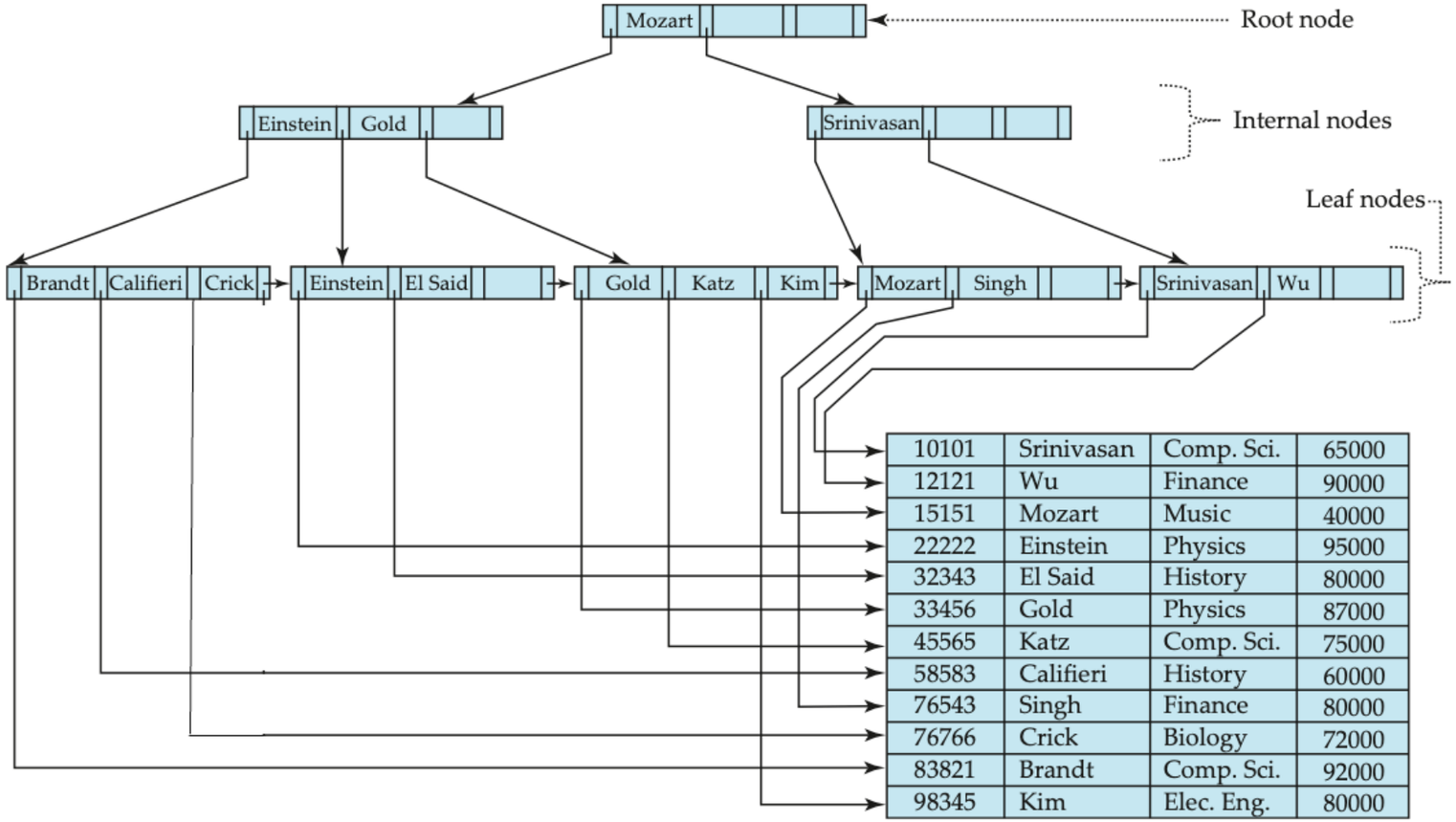
B+-tree

- Features of a B+-tree
  - Data stored only in leaves
  - Leaves are linked sequentially



B+-tree

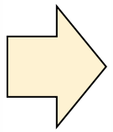
- A complete example of a B+-tree
  - Data stored only in leaves
    - No need to squeeze data into nodes
  - Leaves are linked sequentially
    - Faster table traverse from top to bottom
    - Better support for range queries



Index It or Not: Where Indexing May Help

- Check whether the PK / Unique index helps first
- Index those columns frequently appeared as search criteria
  - `=`
  - `<`, `<=`, `>`, `>=`, `between`
  - `in`
  - `exists`
  - `like` (prefix matching)
- Be cautious when the indexed columns need frequent writing operations
  - insert, update, delete
- Functions

```
SELECT attr1, attr2
FROM table
WHERE function(column) = search_key
```



```
-- Create an index on the return values of the function
-- instead of the original values
create index idx_name ON table1(function(col1));
```

Note: The expression should be deterministic. For detailed usage, please refer to:
`https://www.postgresql.org/docs/14/indexes-expressional.html`

Index It or Not: Where Indexing May Help

- Be cautious when using indexes on a small table
  - Full scan ≠ Bad scheme
  - Index retrieval ≠ Good scheme

Hashing

- A **bucket** is a unit of storage containing one or more entries
  - A bucket is typically a disk block
  - We obtain the bucket of an entry from its search-key value using a hash function
    - Hash function h is a function from the set of all search-key values K to the set of all bucket addresses B
    - Hash function is used to locate entries for access, insertion as well as deletion.

- Entries with different search-key values may be mapped to the same bucket
  - … thus, the entire bucket has to be searched sequentially to locate an entry.

Hashing Index & Hashing File Organization

- In a hash index, buckets store entries with pointers to records

bucket1

```
15151: file 1, offset 100
```

bucket2

```
32343: file 3, offset 20
58583: file 3, offset 72
```

- In a hash file-organization, buckets store records

bucket 0

bucket 1

15151	Mozart	Music	40000

bucket 2

32343	El Said	History	80000
58583	Califieri	History	60000

bucket 3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000

bucket 4

12121	Wu	Finance	90000
76543	Singh	Finance	80000

bucket 5

76766	Crick	Biology	72000

bucket 6

10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

bucket 7

How `join` Works

- Some widely used `join` algorithms
  - Nested-loop join
  - Hash join
  - Sort-merge join

How `join` Works

- Nested loop join
  - Straight-forward linking between records from two tables in a nested-loop manner

```
for each row in t1 match C1(t1)
    for each row in t2 match P(t1, t2)
        if C2(t2)
            add t1|t2 to the result
```

How `join` Works

- Hash join
  - Build a set of buckets for a smaller table to speed up the data lookup

- Procedure:
  - 1. Create a hash table for the smaller table `t1` in the memory
  - 2. Scan the larger table `t2`. For each record `r`,
    - 2.1 Compute the hash value of `r.join_attribute`
    - 2.2 Map to corresponding rows in `t1` using the hash table

How `join` Works

- Sort-merge join (a.k.a. merge join)
  - Zipper-like joining

- Procedure:
  - 1. Sort tables t1 and t2 respectively according to the join attributes
  - 2. Perform an interleaved scan of t1 and t2. When encountering a matched value, join the related rows together.

When there are indexes on the join attributes, step 1, the most expensive operation, can be skipped because t1 and t2 are already sorted in this scenario.

Advanced Topic 1

Beyond Tables: More Data Types

Semi-Structured Data

- Many applications require storage of complex data, whose schema changes often
- The relational model's requirement of atomic data types may be **an overkill**
  - E.g., storing set of interests as a set-valued attribute of a user profile may be simpler than normalizing it
- Data exchange can benefit greatly from semi-structured data
  - Exchange can be between applications, or between back-end and front-end of an application
  - Web-services are widely used today, with complex data fetched to the front-end and displayed using a mobile app or JavaScript
- **JSON** and **XML** are widely used semi-structured data models

Features of Semi-Structured Data Models

- Flexible schema
  - Wide column representation: allow each tuple to have a different set of attributes, can add new attributes at any time
  - Sparse column representation: schema has a fixed but large set of attributes, by each tuple may store only a subset

Features of Semi-Structured Data Models

- Multivalued data types
  - Sets, multisets
    - E.g.,: set of interests: {‘basketball, ‘cooking’, ‘anime’, ‘jazz’}
  - Key-value map (or just map for short)
    - Store a set of key-value pairs
    - E.g.,
      - {(brand, Apple), (ID, MacBook Air), (size, 13), (color, silver)}
    - Operations on maps
      - put(key, value)
      - get(key)
      - delete(key)

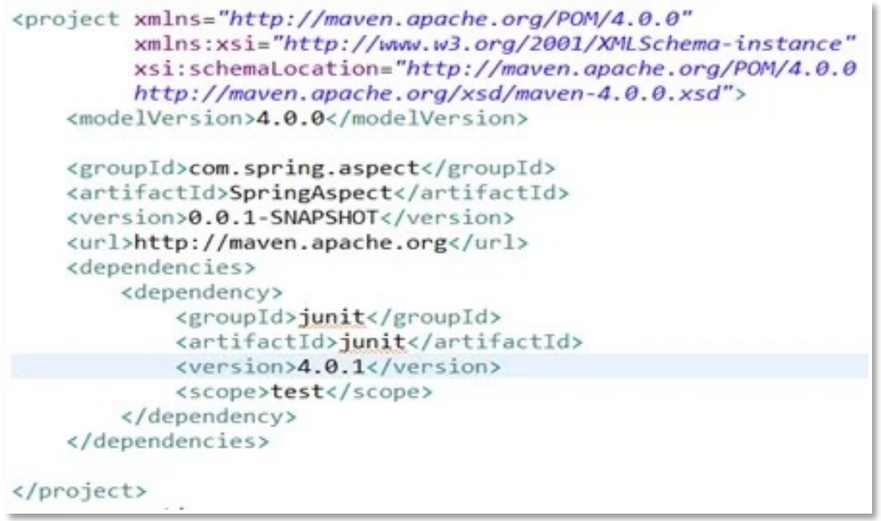
Features of Semi-Structured Data Models

- Arrays
  - Widely used for scientific and monitoring applications
  - E.g., readings taken at regular intervals can be represented as array of values instead of (time, value) pairs
    - [5, 8, 9, 11] instead of {(1,5), (2, 8), (3, 9), (4, 11)}
- Array database: a database that provides specialized support for arrays
  - E.g., compressed storage, query language extensions, etc.
  - Oracle GeoRaster, PostGIS, SciDB, etc

Nested Data Types

- Hierarchical data is common in many applications

- **JSON** (JavaScript Object Notation)
  - Widely used today
- **XML** (eXtensible Markup Language)
  - Earlier generation notation, still used extensively



JSON

- Textual representation widely used for data exchange

- Types: integer, real, string, and
  - Objects: key-value maps, i.e. sets of (attribute name, value) pairs
  - Arrays: also key-value maps (from offset to value)

```
{
    "ID": "22222",
    "name": {
        "firstname: "Albert",
        "lastname: "Einstein"
    },
    "deptname": "Physics",
    "children": [
        {"firstname": "Hans", "lastname": "Einstein" },
        {"firstname": "Eduard", "lastname": "Einstein" }
    ]
}
```

JSON

- JSON is ubiquitous in data exchange today
  - Widely used for web services
    - Most modern applications are architected around on web services
- PostgreSQL supports JSON format columns

```
create table json_test (
    id serial not null primary key,
    student json not null
);

insert into json_test (student) values ('{"name": "aaa", "age": 20, "major": {"primary": "cs", "minor":
"math"}}');
insert into json_test (student) values ('{"name": "bbb", "major": {"primary": "math", "minor": "physics"}}');
insert into json_test (student) values ('{"name": "ccc", "age": 19, "major": {"primary": "biology"}}');
```

JSON

- JSON is ubiquitous in data exchange today
  - Widely used for web services
    - Most modern applications are architected around on web services
- PostgreSQL supports JSON format columns

```
-- select all content from the column
select * from json_test;

-- select a value of a key with "->"
select student -> 'major' -> 'minor' from json_test;
```

	id	student
1	1	{"name": "aaa", "age": 20, "major": {"primary": "cs", "minor": "math"}}
2	2	{"name": "bbb", "major": {"primary": "math", "minor": "physics"}}
3	3	{"name": "ccc", "age": 19, "major": {"primary": "biology"}}

	?column?
1	"math"
2	"physics"
3	<null>

XML

- XML uses tags to mark up text
  - Tags make the data self-documenting
  - Tags can be hierarchical

```
<course>
    <course id>CS-101</course id>
    <title>Intro. to Computer Science</title>
    <dept name>Comp. Sci.</dept name>
    <credits>4</credits>
</course>
```

```
<info>
    <name>aaa</name>
    <age>20</age>
    <major>
        <primary>cs</primary>
        <minor>math</minor>
    </major>
</course>
```

Textual Data

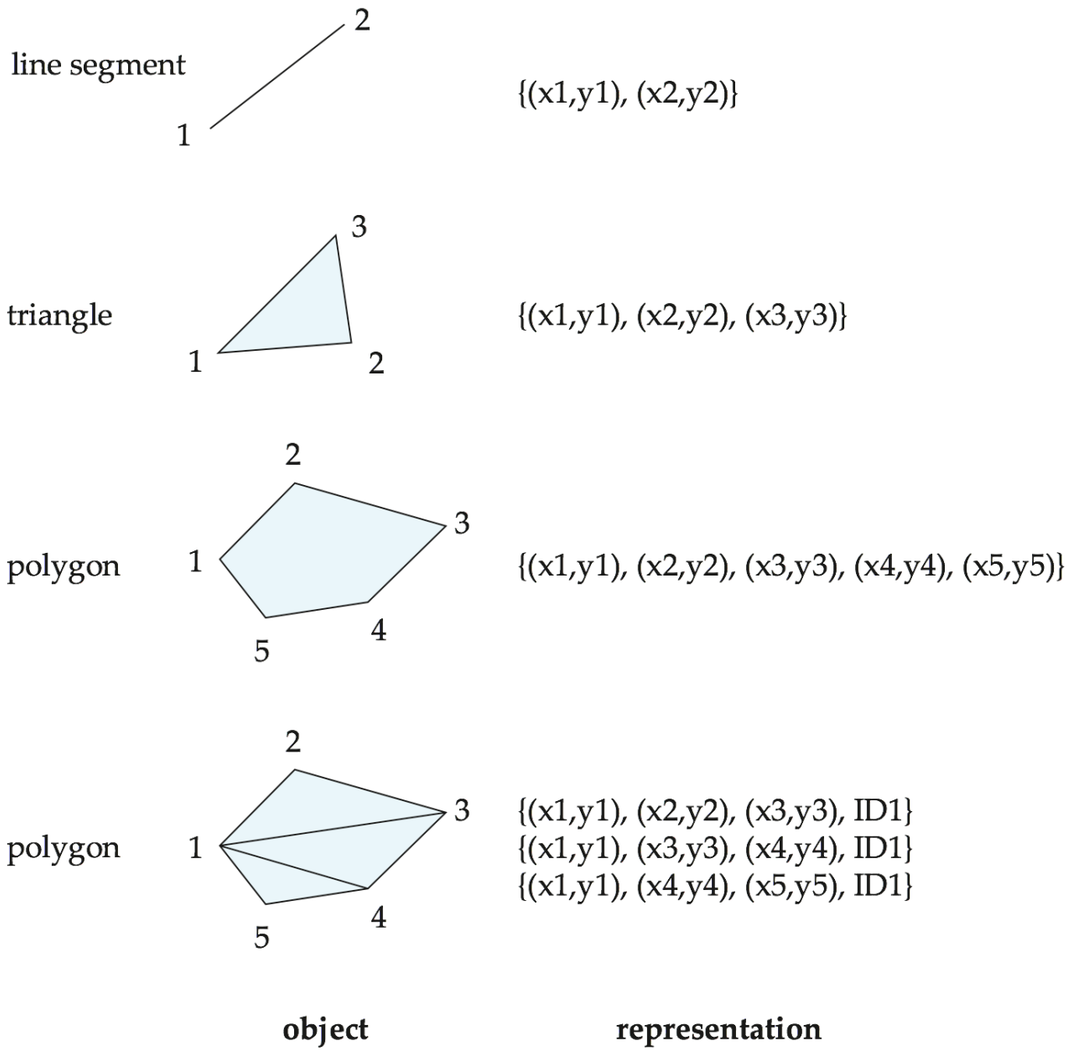
- Information retrieval: querying of unstructured data
  - Simple model of keyword queries: given query keywords, retrieve documents containing all the keywords
  - More advanced models rank relevance of documents
  - Today, keyword queries return many types of information as answers
    - E.g., a query “cricket” typically returns information about ongoing cricket matches
- Relevance ranking
  - Essential since there are usually many documents matching keywords

Spatial Data

- **Spatial databases** store information related to spatial locations, and support efficient storage, indexing and querying of spatial data.
  - **Geographic data**: road maps, land-usage maps, topographic elevation maps, political maps showing boundaries, land-ownership maps, and so on.
    - Geographic information systems (GIS) are special-purpose databases tailored for storing geographic data.
    - Round-earth coordinate system may be used
      - (Latitude, longitude, elevation)
  - **Geometric data**: design information about how objects are constructed
    - E.g., designs of buildings, aircraft, layouts of integrated-circuits.
    - 2 or 3 dimensional Euclidean space with (X, Y, Z) coordinates

Representation of Geometric Information

- Various geometric constructs can be represented in a database in a normalized fashion



Representation of Geometric Information

- Representation of points and line segment in 3-D similar to 2-D, except that points have an extra “z” component
  - Represent arbitrary polyhedra by dividing them into tetrahedrons
    - Similar to triangulating polygons
  - Alternative
    - List their faces, each of which is a polygon, along with an indication of which side of the face is inside the polyhedron

Representation of Geometric Information

- Geometry and geography data types supported by many databases
  - E.g. PostGIS
    - point, linestring, curve, polygons
    - Collections: multipoint, multilinestring, multicurve, multipolygon
    - LINESTRING(1 1, 2 3, 4 4)
    - POLYGON((1 1, 2 3, 4 4, 1 1))
    - Type conversions: ST GeometryFromText() and ST GeographyFromText()
    - Operations: ST Union(), ST Intersection(), …

NoSQL Database

- “Not Only SQL”
  - Useful when working with a huge quantity of data when nature of data does not require a relational model
  - Usually not built on tables and queried by SQL
- Examples
  - Document store – MongoDB
  - Graph structure – Neo4j
  - Key-value storage – Redis, LevelDB
  - Tabular – Apache Hbase (Hadoop-based)

Advanced Topic 2

Beyond Storage: Big Data Analytics

What is Data, By the Way?

data noun, plural in form but singular or plural in construction, often attributive

Save Word

da·ta | \ ˈdā-tə, ˈda- *also* ˈdä- \

Definition of *data*

1 : factual information (such as measurements or statistics) used as a basis for reasoning, discussion, or calculation

// the *data* is plentiful and easily available
— H. A. Gleason, Jr.

// comprehensive *data* on economic growth have been published
— N. H. Jacoby

2 : information in digital form that can be transmitted or processed

3 : information output by a sensing device or organ that includes both useful and irrelevant or redundant information and must be processed to be meaningful

factual information (such as measurements or statistics) used as a basis for reasoning, discussion, or calculation

https://www.merriam-webster.com/dictionary/data

Data Attribute

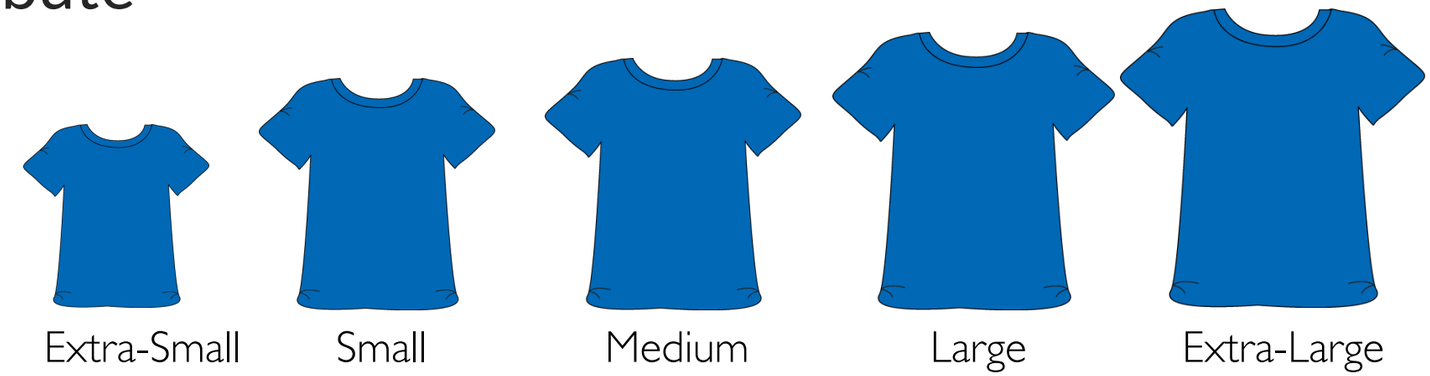
- Characteristics or feature of a data object
  - Other names:
    - Feature
    - Dimension
    - Variable

- Set of attributes: attribute vector

Attribute Types

- Nominal (Categorical) attribute

- Ordinal attribute



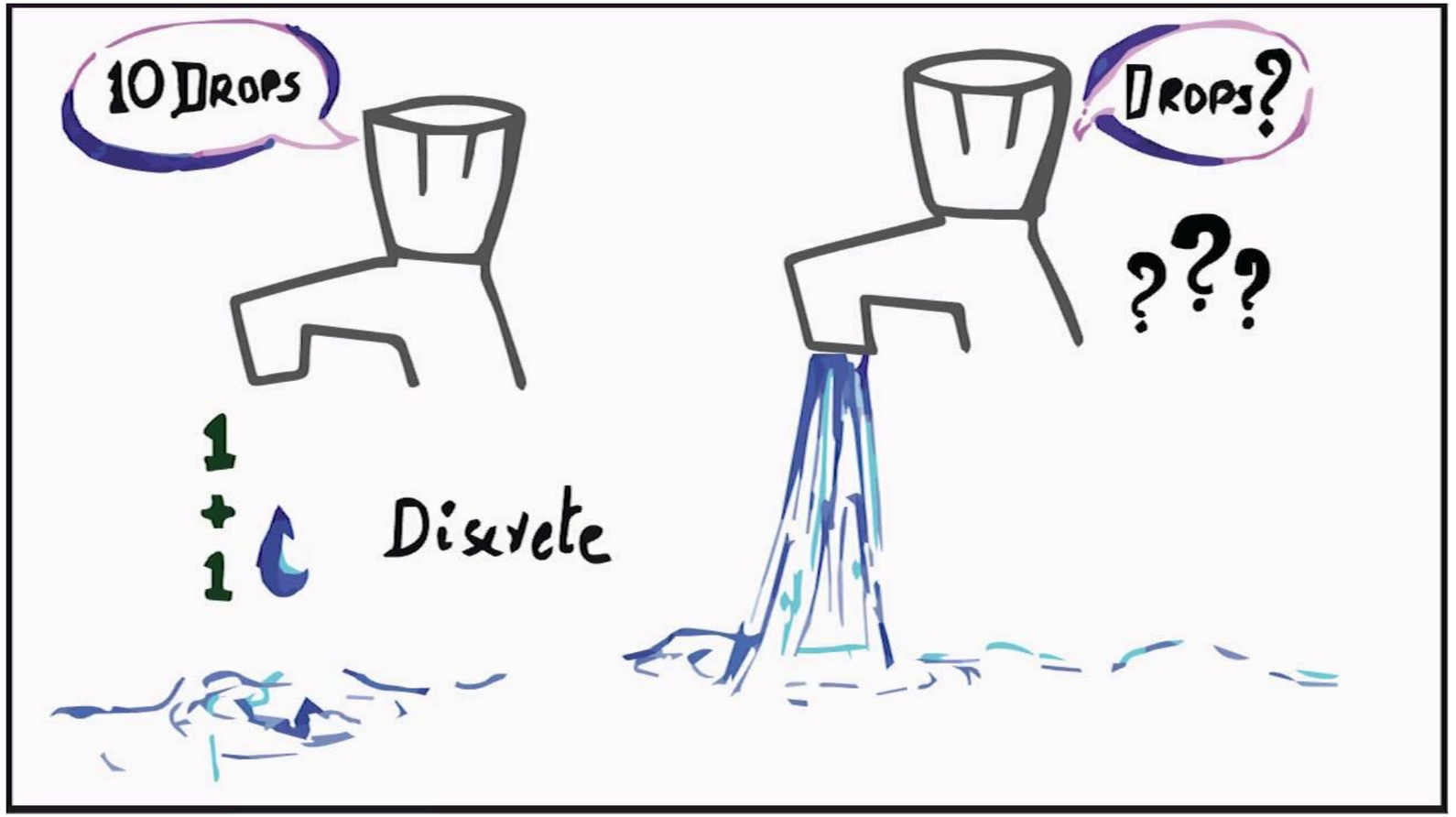
Attribute Types

- Numeric attribute

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa
5.4	3.4	1.7	0.2	setosa
5.1	3.7	1.5	0.4	setosa

https://archive.ics.uci.edu/ml/datasets/iris

Discrete vs. Continuous



Basic Statistical Descriptions

- Overall picture of your data
- Basis of exploratory data analysis

- Mean

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} = \frac{x_1 + x_2 + \cdots + x_N}{N}.$$

- Median

$$\mathrm{Q}_{\frac{1}{2}}(x) = \begin{cases} x'_{\frac{n+1}{2}}, & \text{if } n \text{ is odd.} \\ \frac{1}{2}\left(x'_{\frac{n}{2}} + x'_{\frac{n}{2}+1}\right), & \text{if } n \text{ is even.} \end{cases}$$

- Variance

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 = \left(\frac{1}{N}\sum_{i=1}^{N} x_i^2\right) - \bar{x}^2,$$

Relationship between Data Objects: Data (Dis)Similarity

- Measurement of relationships
  - Commonly used in many statistical methods and data mining algorithms

- Dissimilarity Matrix & Distance Measures

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \cdots & \cdots & 0 \end{bmatrix}$$

Euclidean	$d(x,y) = \sqrt{\sum (x_i - y_i)^2}$
Squared Euclidean	$d(x,y) = \sum (x_i - y_i)^2$
Manhattan	$d(x,y) = \sum \lvert (x_i - y_i) \rvert$
Canberra	$d(x,y) = \sum \frac{\lvert x_i - y_i \rvert}{\lvert x_i + y_i \rvert}$
Chebychev	$d(x,y) = \max(\lvert x_i - y_i \rvert)$
Bray Curtis	$d(x,y) = \frac{\sum \lvert x_i - y_i \rvert}{\sum x_i + y_i}$
Cosine Correlation	$d(x,y) = \frac{\sum (x_i y_i)}{\sqrt{\sum (x_i)^2 \sum (y_i)^2}}$
Pearson Correlation	$d(x,y) = \frac{\sum (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum (y_i - \overline{y})^2} \sqrt{\sum (y_i - \overline{y})^2}}$
Uncentered Peason Correlation	$d(x,y) = \frac{\sum x_i y_i}{\sqrt{\sum (y_i - \overline{y})^2} \sqrt{\sum (y_i - \overline{y})^2}}$
Euclidean Nullweighted	Same as Euclidean, but only the indexes where both x and y have a value (not NULL) are used, and the result is weighted by the number of values calculated. Nulls must be replaced by the missing value calculator (in dataloader).

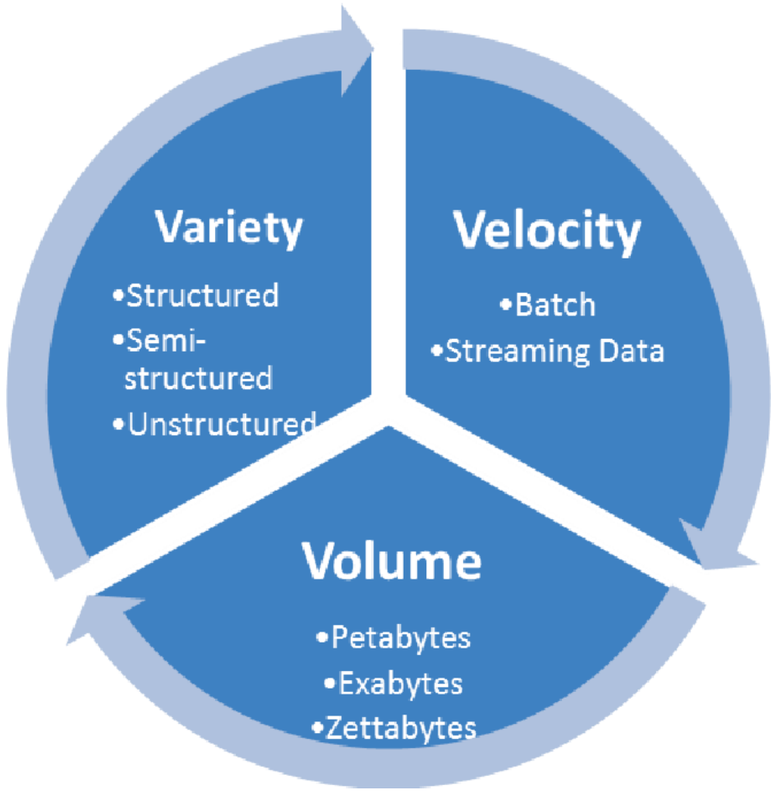
What is Big Data?

- A collection of data sets so large and complex

Three Dimensions of Big Data

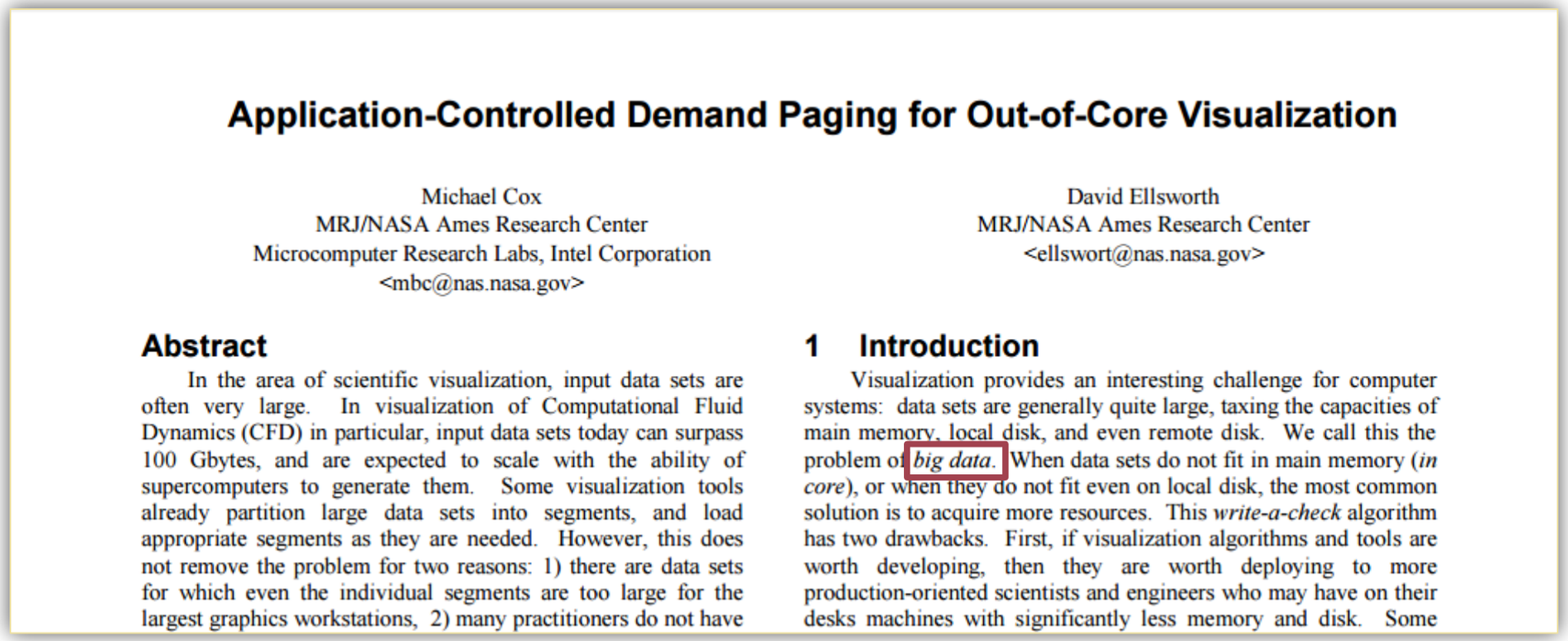
- **Volume**
  - From GB to TB, PB, or higher
- **Velocity**
  - Processing speed
- **Variety**
  - Text, sensor data, multimedia, …

- Other (new) aspects:
  - Veracity: Trustworthiness
  - Value: Worth of data



The Emergence of Data Science

- 1997: The first article to use the term “big data” in the ACM digital library.



Application-Controlled Demand Paging for Out-of-Core Visualization

Michael Cox
MRJ/NASA Ames Research Center
Microcomputer Research Labs, Intel Corporation
<mbc@nas.nasa.gov>

David Ellsworth
MRJ/NASA Ames Research Center
<ellswort@nas.nasa.gov>

Abstract

In the area of scientific visualization, input data sets are often very large. In visualization of Computational Fluid Dynamics (CFD) in particular, input data sets today can surpass 100 Gbytes, and are expected to scale with the ability of supercomputers to generate them. Some visualization tools already partition large data sets into segments, and load appropriate segments as they are needed. However, this does not remove the problem for two reasons: 1) there are data sets for which even the individual segments are too large for the largest graphics workstations, 2) many practitioners do not have

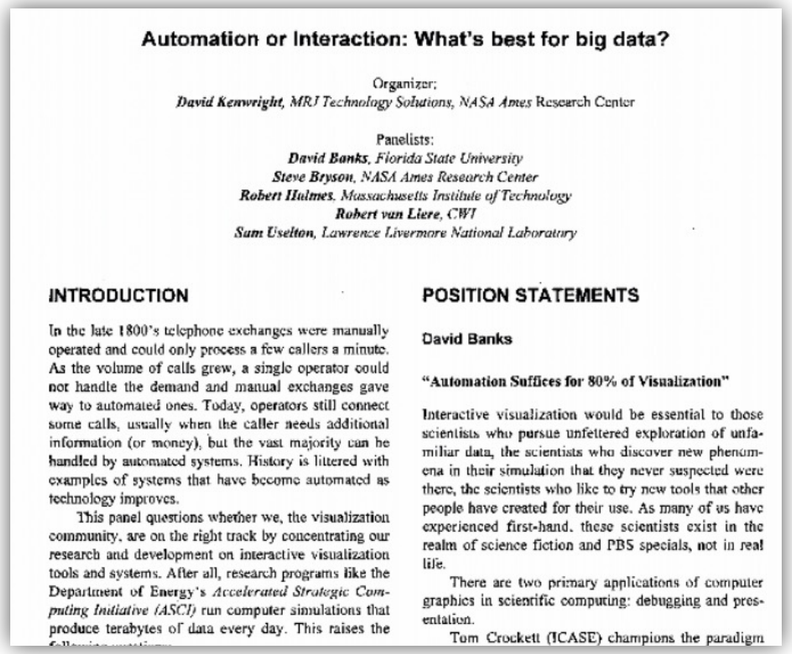
1 Introduction

Visualization provides an interesting challenge for computer systems: data sets are generally quite large, taxing the capacities of main memory, local disk, and even remote disk. We call this the problem of *big data*. When data sets do not fit in main memory (*in core*), or when they do not fit even on local disk, the most common solution is to acquire more resources. This *write-a-check* algorithm has two drawbacks. First, if visualization algorithms and tools are worth developing, then they are worth deploying to more production-oriented scientists and engineers who may have on their desks machines with significantly less memory and disk. Some

Proceedings of the Visualization ‘97

The Emergence of Data Science

- 1999: Bryson, Kenwright and Haimes join David Banks, Robert van Liere, and Sam Uselton on a panel titled “Automation or interaction: what’s best for big data?” at the IEEE Conference on Visualization in 1999



Automation or Interaction: What's best for big data?

Organizer:
David Kenwright, MRJ Technology Solutions, NASA Ames Research Center

Panelists:
***David Banks**, Florida State University*
***Steve Bryson**, NASA Ames Research Center*
***Robert Haimes**, Massachusetts Institute of Technology*
***Robert van Liere**, CWI*
***Sam Uselton**, Lawrence Livermore National Laboratory*

INTRODUCTION

In the late 1800's telephone exchanges were manually operated and could only process a few callers a minute. As the volume of calls grew, a single operator could not handle the demand and manual exchanges gave way to automated ones. Today, operators still connect some calls, usually when the caller needs additional information (or money), but the vast majority can be handled by automated systems. History is littered with examples of systems that have become automated as technology improves.

This panel questions whether we, the visualization community, are on the right track by concentrating our research and development on interactive visualization tools and systems. After all, research programs like the Department of Energy's *Accelerated Strategic Computing Initiative (ASCI)* run computer simulations that produce terabytes of data every day. This raises the

POSITION STATEMENTS

David Banks

"Automation Suffices for 80% of Visualization"

Interactive visualization would be essential to those scientists who pursue unfettered exploration of unfamiliar data, the scientists who discover new phenomena in their simulation that they never suspected were there, the scientists who like to try new tools that other people have created for their use. As many of us have experienced first-hand, these scientists exist in the realm of science fiction and PBS specials, not in real life.

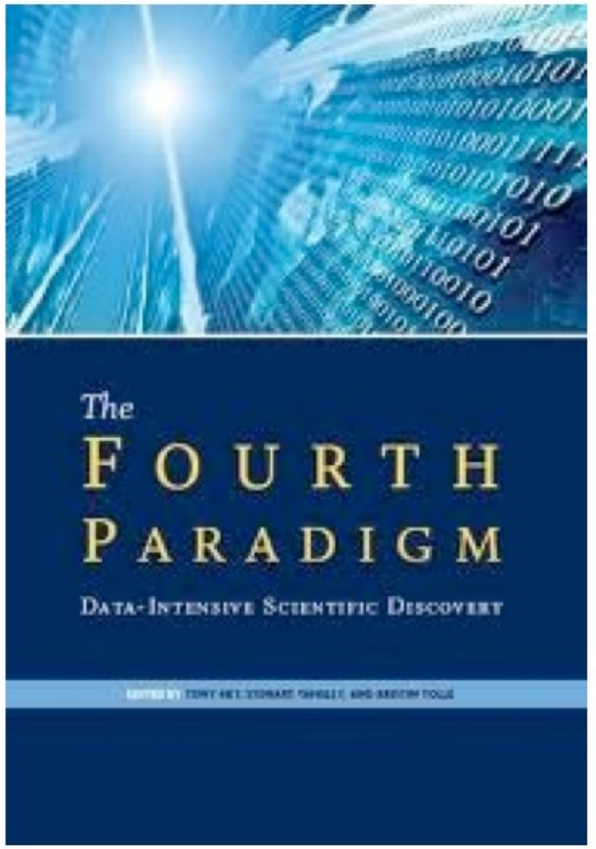
There are two primary applications of computer graphics in scientific computing: debugging and presentation.

Tom Crockett (ICASE) champions the paradigm

Proceedings of the IEEE Conference on Visualization in 1999

The Emergence of Data Science

- 2007: “The Fourth Paradigm”
  - “Data-Intensive Scientific Discovery”



The Emergence of Data Science

- 2012: Data Scientist: The Sexiest Job of the 21st Century

Harvard Business Review

DATA

Data Scientist: The Sexiest Job of the 21st Century

by Thomas H. Davenport and D.J. Patil

FROM THE OCTOBER 2012 ISSUE

The Emergence of Data Science

- 2013: The IEEE Task Force on Data Science and Advanced Analytics was launched.
- 2014: The first international conference: IEEE International Conference on Data Science and Advanced Analytics was launched.
- 2015: The International Journal on Data Science and Analytics was launched by Springer to publish original work on data science and big data analytics.
- 2016: The American Statistical Association section on Statistical Learning and Data Mining renamed to "Statistical Learning and Data Science".

The Emergence of Data Science

- 2016: “Trump vs. Clinton: How Big Data and scientists helped Trump win the election”

Digital campaigning

The role of technology in the presidential election

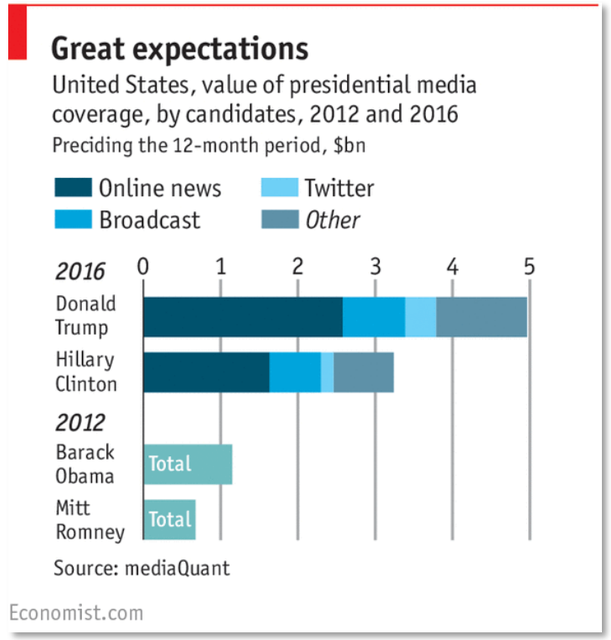
All latest updates

From fake news to big data, a post mortem is under way

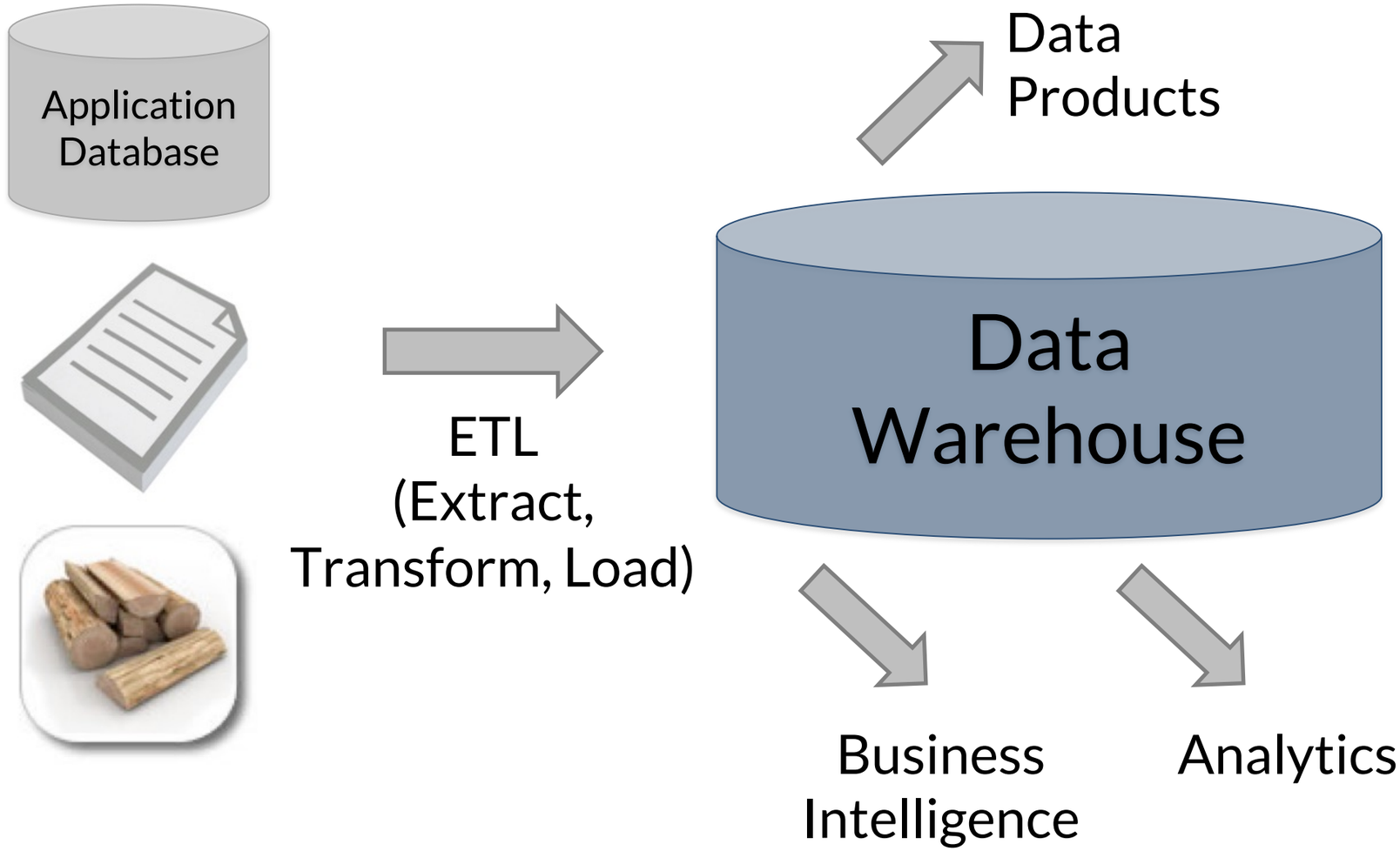
Nov 20th 2016 | United States

Timekeeper Like 916 Tweet

EARLY in America’s presidential campaign, pundits compared the contest between Hillary Clinton and Donald Trump to a fight between a large tanker and Somali pirates. This turned out to be particularly true of the digital campaigns: a massive data battleship lost to a chaotic flotilla of social-media speedboats. The big question now is what this means for future elections, both in America and abroad.



Standard Architecture



Instantiations(1) - Businesspersons

- Data Sources
  - Web pages
  - Excel
- Extract-Transform-Load (ETL)
  - Copy & paste
- Data Warehouse
  - Excel
- BI and Analytics
  - Excel functions
  - Excel charts
  - VB scripts?
  - Visualization tools: Power BI, Tableau

Instantiations(2) - Programmers

- Data Sources
  - Web scraping, web services API
  - CSV files
  - Database queries
- ETL
  - wget, curl, Beautiful Soup, lxml, …
- Data Warehouse
  - Files
- Analytics
  - Numpy, pandas, Matplotlib, R, Octave, …

Instantiations(3) - Enterprises

- Data Sources
  - Application databases(Oracle, IBM, …)
  - Intranet files
  - Application log files
- ETL
  - Infomatica, IBM DataStage, …
- Data Warehouse
  - Teradata, Oracle, IBM DB2, …
- Business Intelligence & Analytics
  - SAS, SPSS, R, …
  - Power BI, Tableau, Spotfire, …

Instantiations(4) – Web Companies

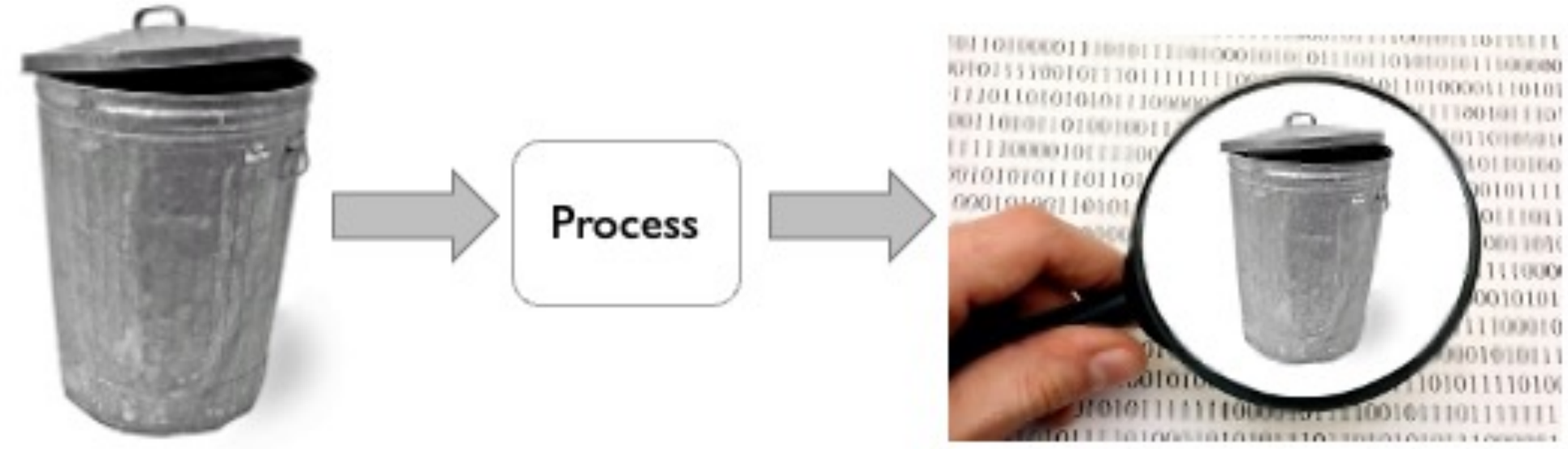
- Data Sources
  - Application databases
  - Logs
  - Web crawl data
- ETL
  - Apache Flume, Apache Sqoop, …
- Data Warehouse
  - Hadoop-based: Hive, Hbase
  - Microsoft Azure, Amazon Redshift
- Business Intelligence & Analytics
  - Argus, R, …

ETL: What is Inside

- Data Cleaning
- Data Integration

“Garbage in, garbage out.”

- Raw data can always be **DIRTY**!



Data Quality

- Data quality: data has quality if it satisfies the requirements of its intended use
  - Accuracy
  - Completeness
  - Consistency
  - Timeliness
  - Believability
  - Interpretability

Data Cleaning

- Deals with detecting and removing errors and inconsistencies from data in order to improve the quality of data

Data Cleaning

- Examples of data quality issues
  - Schema-level

Scope/Problem		Dirty Data	Reasons/Remarks
Attribute	Illegal values	bdate=30.13.70	values outside of domain range
Record	Violated attribute dependencies	age=22, bdate=12.02.70	age = (current date – birth date) should hold
Record type	Uniqueness violation	emp_1=(name="John Smith", SSN="123456") emp_2=(name="Peter Miller", SSN="123456")	uniqueness for SSN (social security number) violated
Source	Referential integrity violation	emp=(name="John Smith", deptno=127)	referenced department (127) not defined

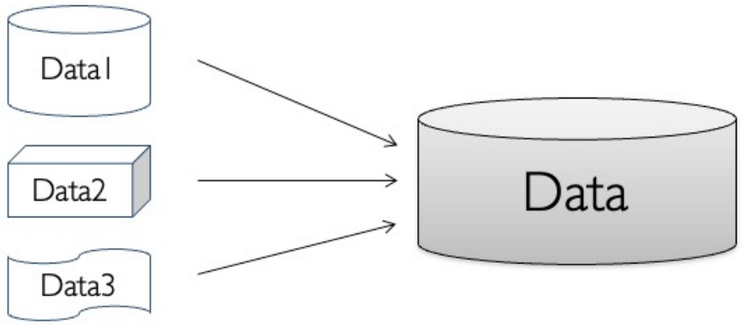
  - Instance-level

Scope/Problem		Dirty Data	Reasons/Remarks
Attribute	Missing values	phone=9999-999999	unavailable values during data entry (dummy values or null)
	Misspellings	city="Liipzig"	usually typos, phonetic errors
	Cryptic values, Abbreviations	experience="B"; occupation="DB Prog."	
	Embedded values	name="J. Smith 12.02.70 New York"	multiple values entered in one attribute (e.g. in a free-form field)
	Misfielded values	city="Germany"	
Record	Violated attribute dependencies	city="Redmond", zip=77777	city and zip code should correspond
Record type	Word transpositions	$name_1$= "J. Smith", $name_2$="Miller P."	usually in a free-form field
	Duplicated records	emp_1=(name="John Smith",...); emp_2=(name="J. Smith",...)	same employee represented twice due to some data entry errors
	Contradicting records	emp_1=(name="John Smith", bdate=12.02.70); emp_2=(name="John Smith", bdate=12.12.70)	the same real world entity is described by different values
Source	Wrong references	emp=(name="John Smith", deptno=17)	referenced department (17) is defined but wrong

Data Integration

- Data integration involves combining data residing in different sources and providing users with a unified view of these data.
  - Remember “views” in DBMS?

- Management of data from multiple sources



Customer (source 1)

CID	*Name*	*Street*	*City*	*Sex*
11	Kristen Smith	2 Hurley Pl	South Fork, MN 48503	0
24	Christian Smith	Hurley St 2	S Fork MN	1

Client (source 2)

Cno	*LastName*	*FirstName*	*Gender*	*Address*	*Phone/Fax*
24	Smith	Christoph	M	23 Harley St, Chicago IL, 60633-2394	333-222-6542 / 333-222-6599
493	Smith	Kris L.	F	2 Hurley Place, South Fork MN, 48503-5998	444-555-6666

Customers (integrated target with cleaned data)

No	*LName*	*FName*	*Gender*	*Street*	*City*	*State*	*ZIP*	*Phone*	*Fax*	*CID*	*Cno*
1	Smith	Kristen L.	F	2 Hurley Place	South Fork	MN	48503-5998	444-555-6666		11	493
2	Smith	Christian	M	2 Hurley Place	South Fork	MN	48503-5998			24	
3	Smith	Christoph	M	23 Harley Street	Chicago	IL	60633-2394	333-222-6542	333-222-6599		24

Typical Data Cleaning and Integration Workflow

- Data analysis
  - Detailed inspection before operations
- Conflicts resolution
  - Resolve data confliction between data sources to be integrated
- Definition of transformation workflow and mapping rules
  - Workflow methods for schema adaption and transformation
- Verification of Workflow
  - Verify each steps
- Transformation
  - start the process

Load and Store Data

- File-based Storage
  - Simplest way & easy to manage
  - Scalability is low

- **Database & DBMS**
  - What we have learned for 10+ weeks

- Data Warehouse

Data Warehouse

A data warehouse is a **subject-oriented**, **integrated**, **time-varient**, and **nonvolatile** collection of data in support of management's decision making process.

-- W. H. Inmon, "Building the Data Warehouse". 1996.

Loosely Speaking, a data warehouse refers to a data repository that is **maintained separately** from an organization's operational databases.

-- J. Han and M. Kamber, "Data Mining: Concepts and Techniques", 3rd ed., 2011.

Differences between Databases and Data Warehouses

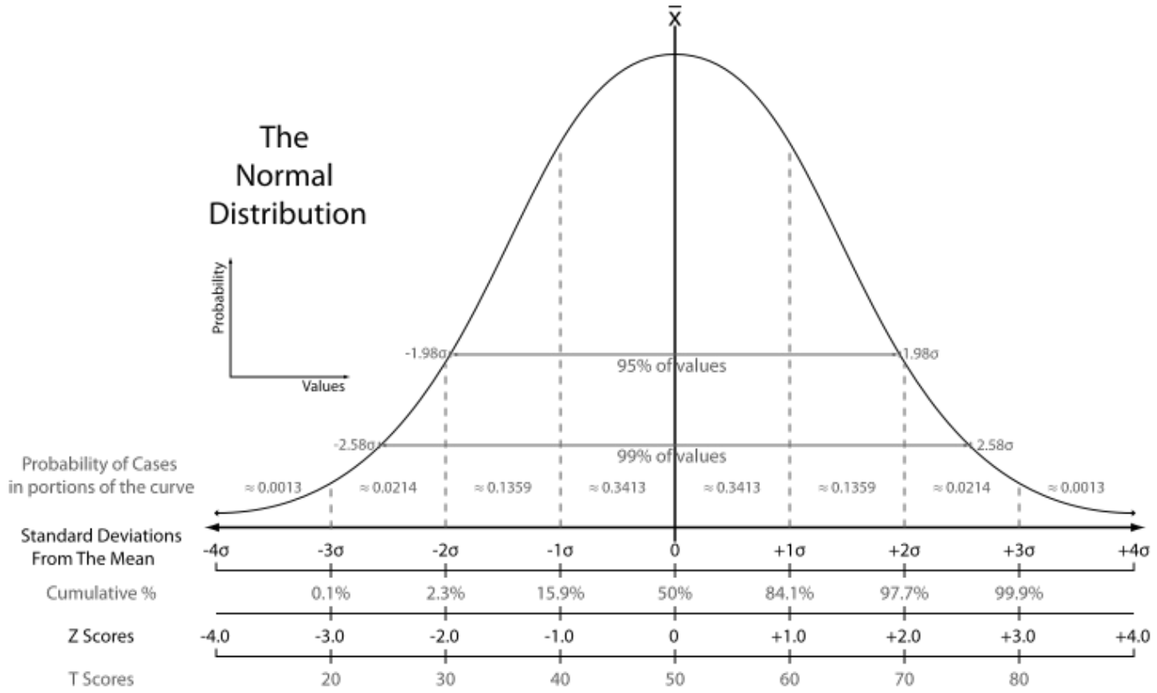
	DB	DW
Characteristics	operational processing	Informational processing
Orientation	transaction	analysis
User	terminal users: clerk, database administrator(DBA)	knowledge workers: manager, analyst, executive
Function	everyday operations	long-term informational requirements decision support
Data	current, up-to-date	historic, accuracy maintained over time
Access	read/write	mostly read
Focus	data in	information/knowledge out
Size	GB to high-order GB	>=TB

Data Analysis

- Exploratory Data Analysis
- Data Mining

Something Basic: Statistics

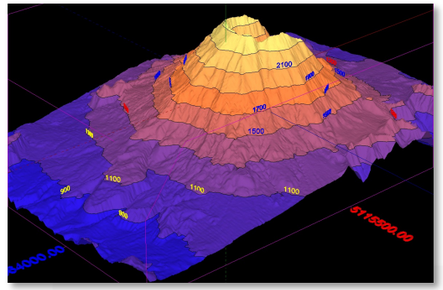
- (Probably) Foundation of modern data analysis
- (Also) Foundation of machine learning, data mining, etc.

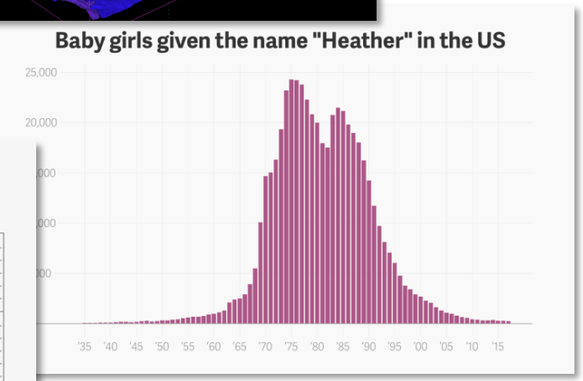


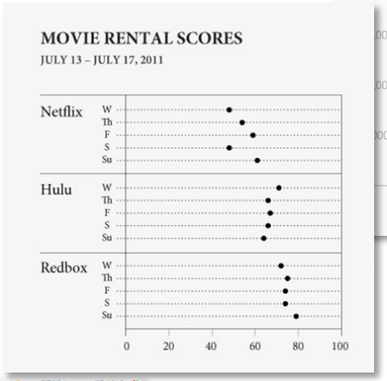
Exploratory Data Analysis (EDA)

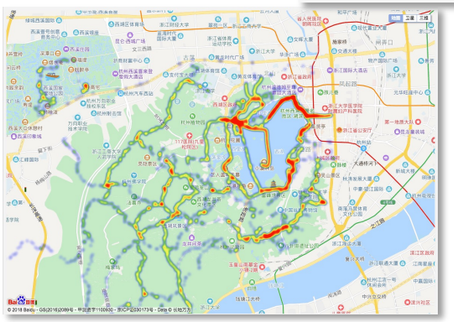
- Based on statistics
  - Data visualization-driven method
  - Summary of main characteristics in easy-to-understand form

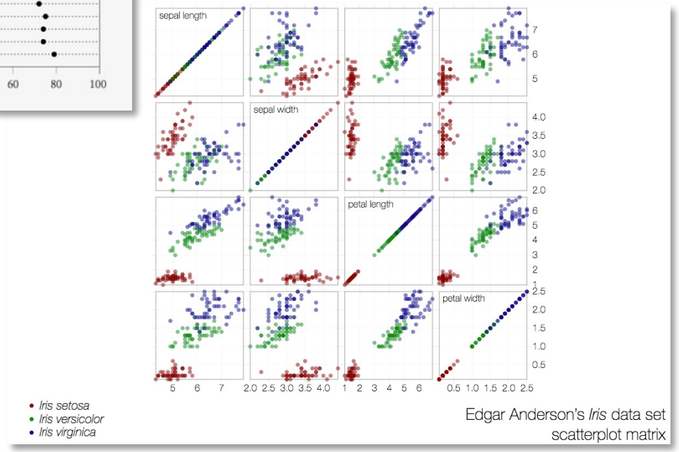
- Types of data visualization methods in EDA:
  - Plotting of raw data
  - Plotting of statistical values
  - Multiple coordinated views (Dashboard)











Exploratory Data Analysis (EDA)

"Some of my friends felt that I should be very explicit in warning you of how much time and money can be wasted on computing, how much clarity and insight can be lost in great stacks of computer output. In fact, I ask you to remember only two points:

1. The tool that is so dull that you cannot cut yourself on it is not likely to be sharp enough to be either useful or helpful.

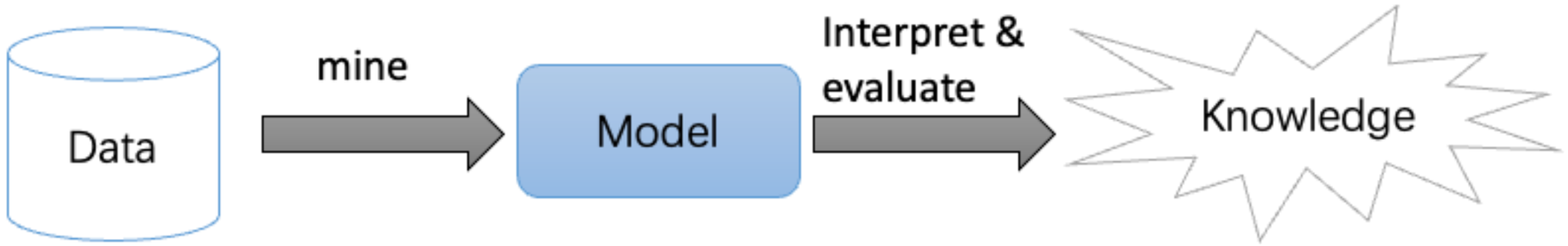
2. Most uses of the classical tools of statistics have been, are, and will be , made by those who know not what they do."

-- John W. Tukey, "The Technical Tools of Statistics",
at the 125th Anniversary Meeting of American Statistical Association, 1964

Data Mining

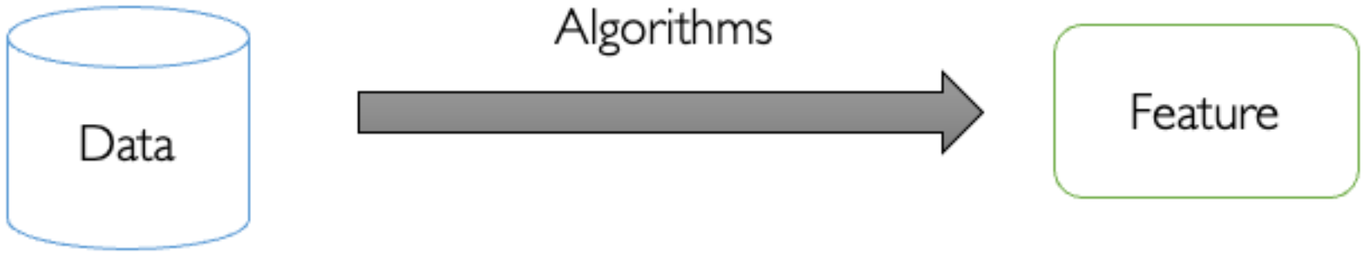
"**Data Mining**, also popularly referred to as **knowledge discovery** from data (KDD), is the automated or convenient **extraction of patterns representing knowledge** implicitly stored or captured in large databases, data warehouses, the Web, other massive repositories, or data streams."

-- H. Jiawei and M. Kamber, "Data Mining: Concepts and Techniques", 3rd ed., 2011.

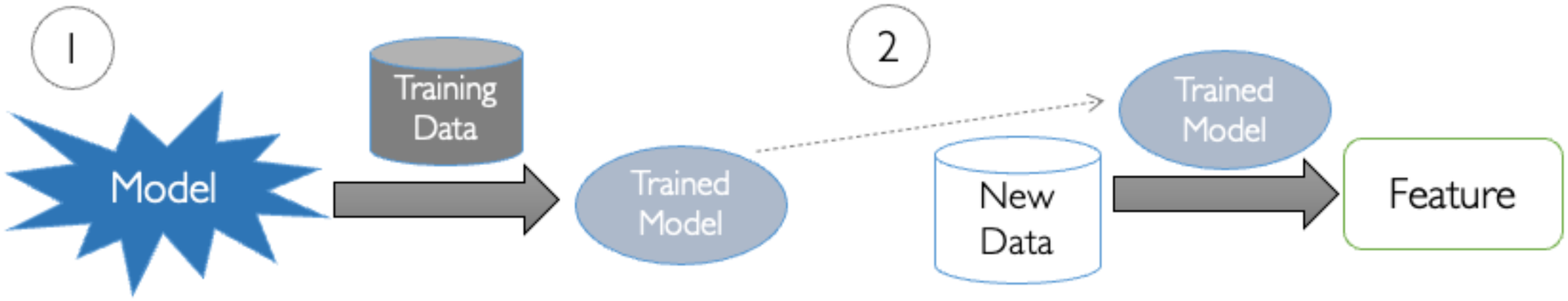


Tasks in Data Mining

- Descriptive Tasks



- Predictive Tasks



Descriptive Tasks

- Concept Description
  - Describe features of data directly
- Association Analysis
  - Analyze “feature-value” pairs that occur frequently in data
- Clustering
  - Group data on the principle of maximizing the intra-class similarity and minimizing the inter-class similarity
- Outlier Detection
  - Analyze objects that do not comply with the general behavior or model of the data

Predictive Tasks

- Regression
  - Model the relationship between a scalar response and a number of variables
- Classification
  - Find a model/function that describes and distinguish data classes or concepts based on analysis of a set of training data
- Evolution Analysis
  - Analyze temporal and spatial patterns in dataset, model these patterns and predict data in unknown spatio-temporal positions

Data Visualization

- **Visualization is the creation and study of the visual representation of data**

Input: data

Output: visual form

Goal: insight

Why Do We Need Visualization?

- Sometimes, statistics may not work

Set A		Set B		Set C		Set D	
X	Y	X	Y	X	Y	X	Y
10	8.04	10	9.14	10	7.46	8	6.58
8	6.95	8	8.14	8	6.77	8	5.76
13	7.58	13	8.74	13	12.74	8	7.71
9	8.81	9	8.77	9	7.11	8	8.84
11	8.33	11	9.26	11	7.81	8	8.47
14	9.96	14	8.1	14	8.84	8	7.04
6	7.24	6	6.13	6	6.08	8	5.25
4	4.26	4	3.1	4	5.39	19	12.5
12	10.84	12	9.11	12	8.15	8	5.56
7	4.82	7	7.26	7	6.42	8	7.91
5	5.68	5	4.74	5	5.73	8	6.89

Summary Statistics
$u_X = 9.0$ $\sigma_X = 3.317$
$u_Y = 7.5$ $\sigma_Y = 2.03$

Linear Regression
$Y = 3 + 0.5\,X$
$R^2 = 0.67$

[Anscombe 73]