

CSC 261/461 – Database Systems

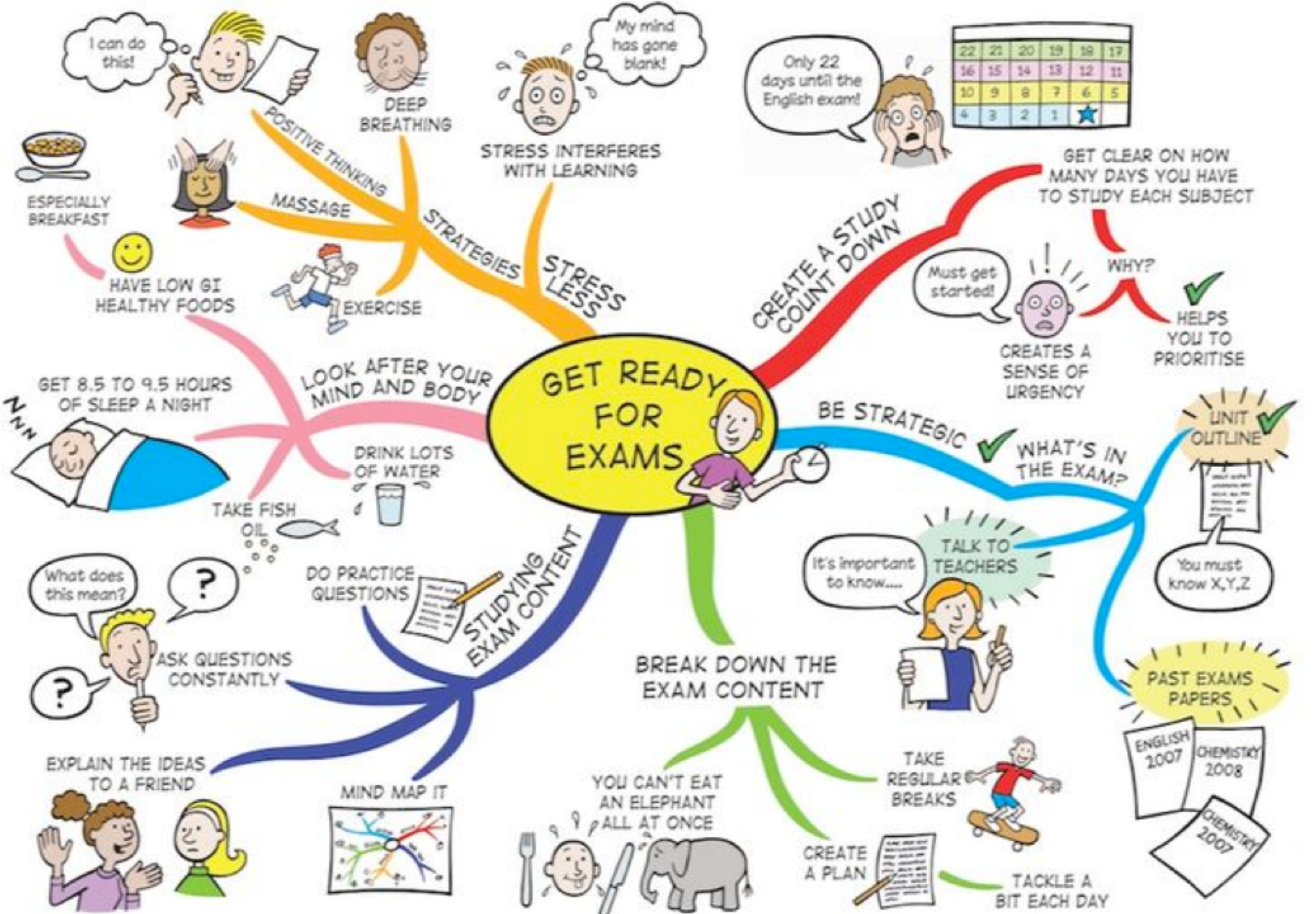
Lecture 14

Spring 2018

Announcement

- Midterm on Wednesday!





22	21	20	19	18	17
16	15	14	13	12	11
10	9	8	7	6	5
4	3	2	1	★	



Announcement

- Start practicing HTML, CSS, JavaScript, PHP + SQL
 - We will cover the basics in workshops
 - https://www.w3schools.com/php/php_mysql_intro.asp

- Project 1 Milestone 3 will be out soon.
 - Combination of **Theory** and **Application**
 - BCNF decomposition and PHP and MySQL

- Project 2 Part 2 will be out soon, too.

Announcement

Workshops from now onwards are going to be very important for doing well in your projects and save you plenty of time

We will cover:

1. Triggers
2. HTML, CSS, Java Script
3. PHP and MySQL together
4. MongoDB
5. Spark (if time permits)

Agenda

- Relational Algebra (Today)
 - Please read Chapter 8
- Relational Calculus (We will not cover)
- Also, Midterm Review

RELATIONAL ALGEBRA

Motivation

Relational Algebra provides a formal foundation for relational model operations

It is the basis for implementing and optimizing queries in any RDBMS

The core operations of most relational systems are based on Relational Algebra

The Relational Model: Schemata

- Relational Schema:

Students (sid: string, name: string, gpa: float)

**Relation
name**

String, float, int, etc.
are the **domains** of
the attributes

Attributes

The Relational Model: Data

An attribute (or column) is a typed data entry present in each tuple in the relation

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

The number of attributes is the arity of the relation

The Relational Model: Data

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

The number of tuples is the **cardinality** of the relation

A **tuple** or **row** (or *record*) is a single entry in the table having the attributes specified by the schema

The Relational Model: Data

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

Recall: In practice DBMSs relax the set requirement, and use multisets.

A **relational instance** is a ***set*** of tuples all conforming to the same ***schema***

Relation Instances

- *Relation DB Schema*

- Students(sid: *string*, name: *string*, gpa: *float*)
- Courses(cid: *string*, cname: *string*, credits: *int*)
- Enrolled(sid: *string*, cid: *string*, grade: *string*)

Note that the schemas impose effective domain / type constraints, i.e. Gpa can't be "Apple"

Sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Students

Relation
Instances

cid	cname	credits
564	564-2	4
308	417	2

Courses

sid	cid	Grade
123	564	A

Enrolled

Querying

```
SELECT S.name  
FROM Students S  
WHERE S.gpa > 3.5;
```

*“Find names of all students
with GPA > 3.5”*

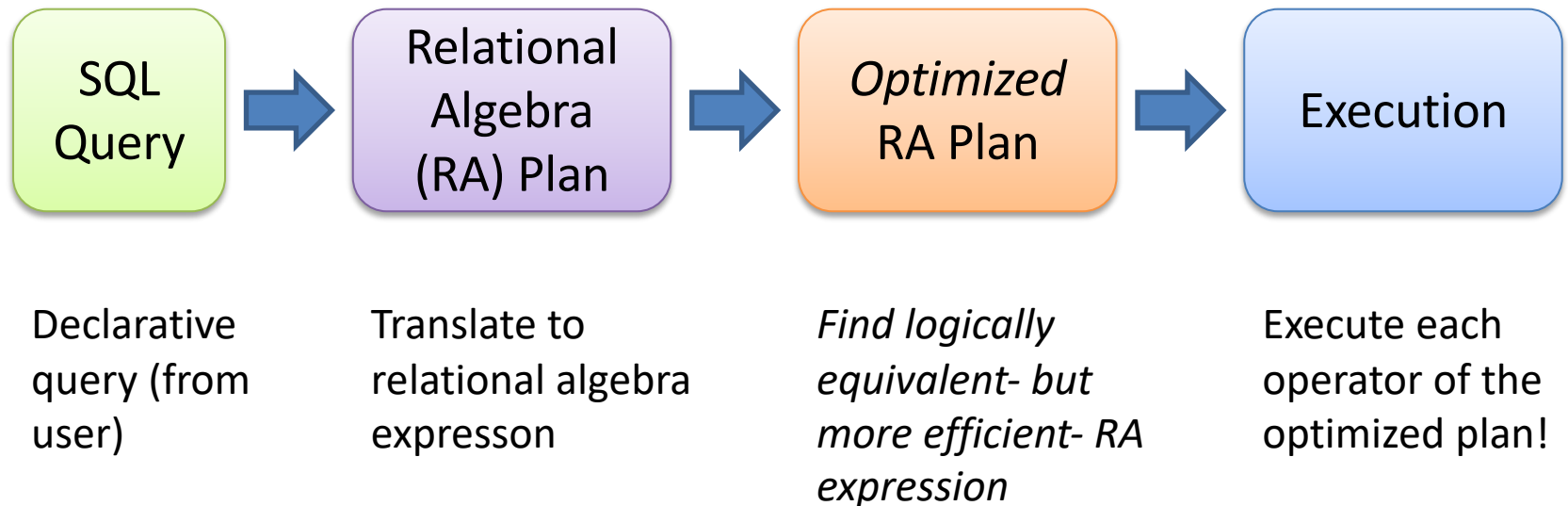
We don't tell the system *how* or *where* to get the data- **just what we want, i.e., Querying is declarative**

To make this happen, we need to translate the *declarative* query into a series of operators... we'll see this next!

Relational Algebra

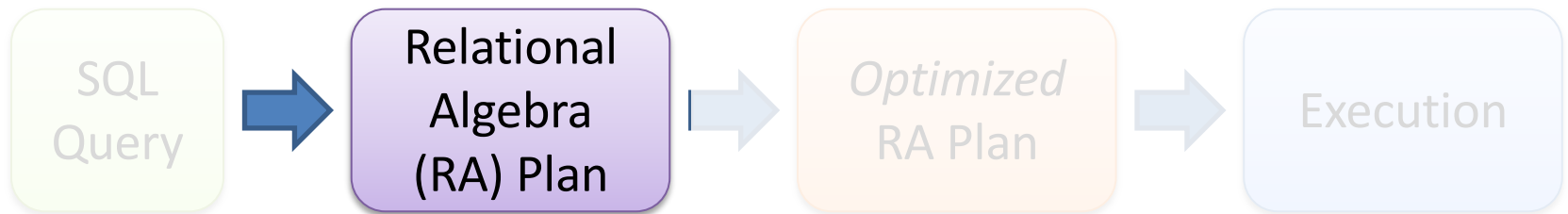
RDBMS Architecture

How does a SQL engine work ?



RDBMS Architecture

How does a SQL engine work ?



Relational Algebra allows us to translate declarative (SQL) queries into precise and optimizable expressions!

Relational Algebra (RA)

- Five basic operators:

1. Selection: σ
2. Projection: Π
3. Cartesian Product: \times
4. Union: \cup
5. Difference: $-$

We'll look at these first!

- Derived or auxiliary operators:

- Intersection
- Joins (natural, equi-join, theta join, semi-join)
- Renaming: ρ
- Division

And also at one example of a derived operator (natural join) and a special operator (renaming)

Keep in mind: RA operates on sets!

- RDBMSs use **multisets**, however in relational algebra formalism we will consider **sets**!
- Also: we will consider the *named perspective*, where every attribute must have a unique name
 - → attribute order does not matter...

Now on to the basic RA operators...

1. Selection (σ)

Students(sid, sname, gpa)

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- Examples
 - $\sigma_{\text{Salary} > 40000}$ (Employee)
 - $\sigma_{\text{name} = \text{"Smith"}}$ (Employee)
- The condition c can be =, <, \leq , >, \geq , <>

SQL:

```
SELECT *  
FROM Students  
WHERE gpa > 3.5;
```



RA:

$\sigma_{\text{gpa} > 3.5}(\textit{Students})$

Another example:

SSN	Name	Salary
1234545	John	200000
5423341	Smith	600000
4352342	Fred	500000

$\sigma_{\text{Salary} > 40000}$ (Employee)



SSN	Name	Salary
5423341	Smith	600000
4352342	Fred	500000

2. Projection (Π)

- Eliminates columns, then removes duplicates
- Notation: $\Pi_{A1, \dots, An}(R)$
- Example: project social-security number and names:
 - $\Pi_{SSN, Name}(Employee)$
 - Output schema: Answer(SSN, Name)

Students(sid, sname, gpa)

SQL:

```
SELECT DISTINCT  
  sname,  
  gpa  
FROM Students;
```



RA:

$\Pi_{sname, gpa}(Students)$

Another example:

SSN	Name	Salary
1234545	John	200000
5423341	John	600000
4352342	John	200000

$\Pi_{\text{Name,Salary}}$ (Employee)



Name	Salary
John	200000
John	600000

Note that RA Operators are Compositional!

Students(sid, sname, gpa)

```
SELECT DISTINCT
  sname,
  gpa
FROM Students
WHERE gpa > 3.5;
```

How do we represent
this query in RA?



$\Pi_{sname, gpa}(\sigma_{gpa > 3.5}(Students))$



$\sigma_{gpa > 3.5}(\Pi_{sname, gpa}(Students))$

Are these logically
equivalent?

3. Cross-Product (×)

- Each tuple in R1 with each tuple in R2
- Notation: $R1 \times R2$
- Example:
 - Employee \times Dependents
- Rare in practice; mainly used to express joins

```
Students(sid, sname, gpa)  
People(ssn, pname, address)
```

SQL:

```
SELECT *  
FROM Students, People;
```



RA:

Students \times *People*

Another example: People

ssn	pname	address
1234545	John	216 Rosse
5423341	Bob	217 Rosse



Students

sid	sname	gpa
001	John	3.4
002	Bob	1.3

Students × People



ssn	pname	address	sid	sname	gpa
1234545	John	216 Rosse	001	John	3.4
5423341	Bob	217 Rosse	001	John	3.4
1234545	John	216 Rosse	002	Bob	1.3
5423341	Bob	216 Rosse	002	Bob	1.3

Renaming (ρ)

- Changes the schema, not the instance
- A 'special' operator- neither basic nor derived
- Notation: $\rho_{B1, \dots, Bn}(R)$

- **Note: this is shorthand for the proper form (since names, not order matters!):**

$$- \rho_{A1 \rightarrow B1, \dots, An \rightarrow Bn}(R)$$

Students(sid, sname, gpa)

SQL:

```
SELECT
  sid AS studId,
  sname AS name,
  gpa AS gradePtAvg
FROM Students;
```



RA:

$\rho_{studId, name, gradePtAvg}(Students)$

We care about this operator *because* we are working in a *named perspective*

Another example:

Students

sid	sname	gpa
001	John	3.4
002	Bob	1.3

$\rho_{studId,name,gradePtAvg}(Students)$



Students

studId	name	gradePtAvg
001	John	3.4
002	Bob	1.3

Natural Join (\bowtie)

Note: Textbook notation is *

```
Students(sid, name, gpa)
People(ssn, name, address)
```

- Notation: $R_1 \bowtie R_2$
- Joins R_1 and R_2 on *equality of all shared attributes*
 - If R_1 has attribute set A , and R_2 has attribute set B , and they share attributes $A \cap B = C$, can also be written: $R_1 \bowtie_C R_2$
- Our first example of a *derived* RA operator:
 - Meaning: $R_1 \bowtie R_2 = \Pi_{A \cup B}(\sigma_{C=D}(\rho_{C \rightarrow D}(R_1) \times R_2))$
 - Where:
 - The rename $\rho_{C \rightarrow D}$ renames the shared attributes in one of the relations
 - The selection $\sigma_{C=D}$ checks equality of the shared attributes
 - The projection $\Pi_{A \cup B}$ eliminates the duplicate common attributes

SQL:

```
SELECT DISTINCT
  sid, S.name, gpa,
  ssn, address
FROM
  Students S,
  People P
WHERE S.name = P.name;
```



RA:
Students \bowtie *People*

$$R_1 \bowtie R_2 = \Pi_{A \cup B}(\sigma_{C=D}(\rho_{C \rightarrow D}(R_1) \times R_2))$$

Another example:

Students S

sid	S.name	gpa
001	John	3.4
002	Bob	1.3



People P

ssn	P.name	address
1234545	John	216 Rosse
5423341	Bob	217 Rosse

Students ⋈ *People*



sid	S.name	gpa	ssn	address
001	John	3.4	1234545	216 Rosse
002	Bob	1.3	5423341	216 Rosse

Natural Join

- Given schemas $R(A, B, C, D)$, $S(A, C, E)$, what is the schema of $R \bowtie S$?
- Given $R(A, B, C)$, $S(D, E)$, what is $R \bowtie S$?
- Given $R(A, B)$, $S(A, B)$, what is $R \bowtie S$?

Example: Converting SFW Query -> RA

```
Students(sid,name,gpa)  
People(ssn,name,address)
```

```
SELECT DISTINCT  
  gpa,  
  address  
FROM Students S,  
     People P  
WHERE gpa > 3.5 AND  
      S.name = P.name;
```


$$\Pi_{gpa,address}(\sigma_{gpa>3.5}(S \bowtie P))$$

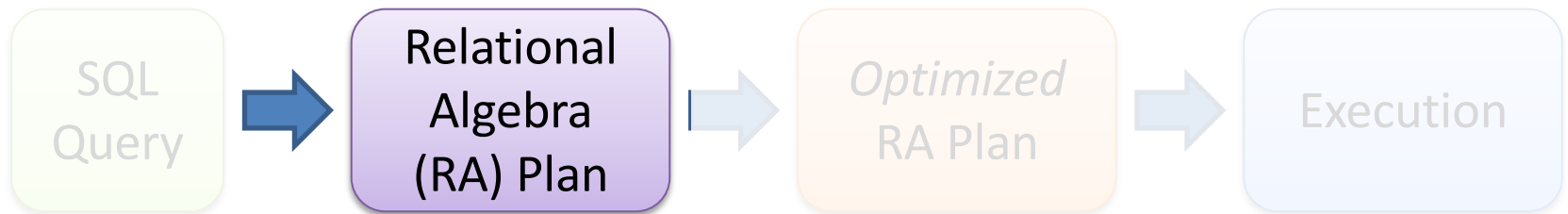
How do we represent
this query in RA?

Logical Equivalence of RA Plans

- Given relations $R(A,B)$ and $S(B,C)$:
 - Here, projection & selection commute:
 - $\sigma_{A=5}(\Pi_A(R)) = \Pi_A(\sigma_{A=5}(R))$
 - What about here?
 - $\sigma_{A=5}(\Pi_B(R)) \stackrel{?}{=} \Pi_B(\sigma_{A=5}(R))$

RDBMS Architecture

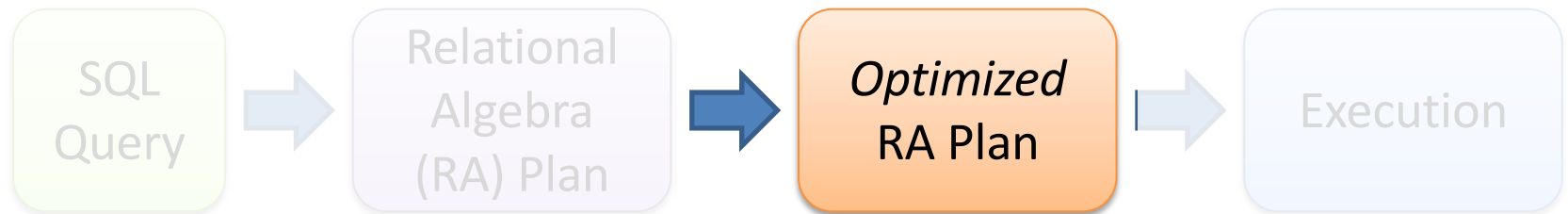
How does a SQL engine work ?



We saw how we can transform declarative SQL queries into precise, compositional RA plans

RDBMS Architecture

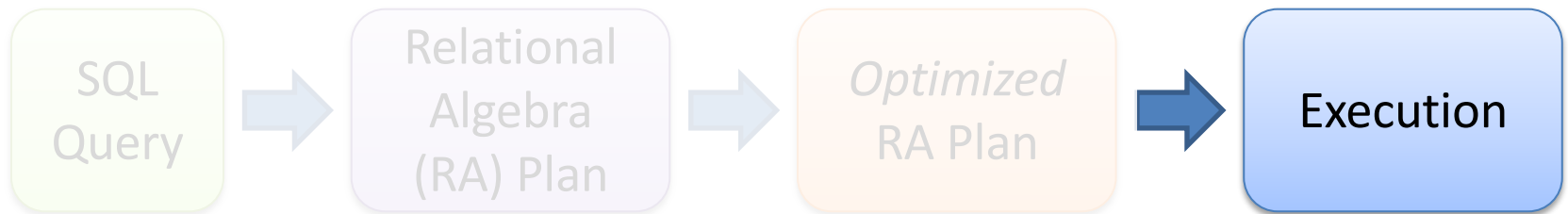
How does a SQL engine work ?



We'll look at how to then optimize these plans later in this lecture

RDBMS Architecture

How is the RA “plan” executed?



We already know how to execute all the basic operators!

2. ADV. RELATIONAL ALGEBRA

What you will learn about in this section

1. Set Operations in RA
2. Fancier RA

Relational Algebra (RA)

- Five basic operators:

1. Selection: σ

2. Projection: Π

3. Cartesian Product: \times

4. Union: \cup

5. Difference: $-$

- Derived or auxiliary operators:

- Intersection

- Joins (natural, equi-join, theta join, semi-join)

- Renaming: ρ

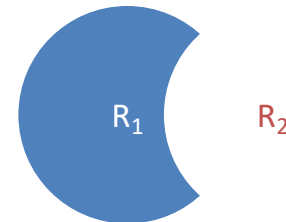
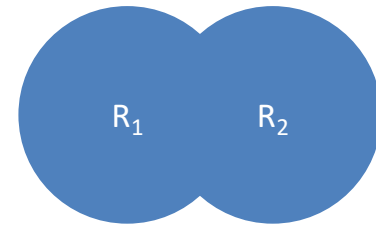
- Division

We'll look at these

*And also at some of
these derived
operators*

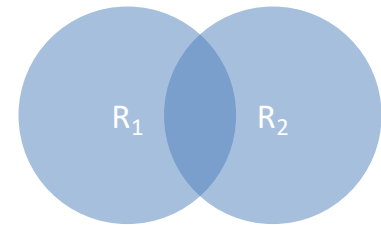
1. Union (\cup) and 2. Difference ($-$)

- $R_1 \cup R_2$
- Example:
 - ActiveEmployees \cup RetiredEmployees
- $R_1 - R_2$
- Example:
 - AllEmployees -- RetiredEmployees



What about Intersection (\cap) ?

- It is a derived operator
- $R1 \cap R2 = R1 - (R1 - R2)$
- Also expressed as a join!
- Example
 - `UnionizedEmployees` \cap `RetiredEmployees`



Fancier RA

Theta Join (\bowtie_{θ})

- A join that involves a predicate
- $R1 \bowtie_{\theta} R2 = \sigma_{\theta} (R1 \times R2)$
- Here θ can be any condition

Note that natural join is a theta join + a projection.

```
Students(sid, sname, gpa)  
People(ssn, pname, address)
```

SQL:

```
SELECT *  
FROM  
  Students, People  
WHERE  $\theta$ ;
```



RA:
Students \bowtie_{θ} *People*

Equi-join ($\bowtie_{A=B}$)

- A theta join where θ is an equality
- $R1 \bowtie_{A=B} R2 = \sigma_{A=B} (R1 \times R2)$
- Example:
 - Employee $\bowtie_{SSN=SSN}$ Dependents

Most common join
in practice!

```
Students(sid, sname, gpa)
People(ssn, pname, address)
```

SQL:

```
SELECT *
FROM
  Students S,
  People P
WHERE sname = pname;
```



RA:

$S \bowtie_{sname=pname} P$

Semijoin (\bowtie)

- $R \bowtie S = \Pi_{A_1, \dots, A_n} (R \bowtie S)$
- Where A_1, \dots, A_n are the attributes in R
- Example:
 - Employee \bowtie Dependents

```
Students(sid, sname, gpa)  
People(ssn, pname, address)
```

SQL:

```
SELECT DISTINCT  
  sid, sname, gpa  
FROM  
  Students, People  
WHERE  
  sname = pname;
```



RA:

Students \bowtie People

Division (\div)

- $T(Y) = R(Y,X) \div S(X)$
- Y is the set of attributes of R that are not attributes of S.
- For a tuple **t** to appear in the result T of the Division, the values in **t** must appear in R in combination with *every* tuple in S.

Example

R(Y,X)

÷

S(X)

=

T(Y)

```
PilotSkills
pilot_name  plane_name
=====
'Celko'     'Piper Cub'
'Higgins'   'B-52 Bomber'
'Higgins'   'F-14 Fighter'
'Higgins'   'Piper Cub'
'Jones'     'B-52 Bomber'
'Jones'     'F-14 Fighter'
'Smith'     'B-1 Bomber'
'Smith'     'B-52 Bomber'
'Smith'     'F-14 Fighter'
'Wilson'    'B-1 Bomber'
'Wilson'    'B-52 Bomber'
'Wilson'    'F-14 Fighter'
'Wilson'    'F-17 Fighter'
```

```
Hangar
plane_name
=====
'B-1 Bomber'
'B-52 Bomber'
'F-14 Fighter'
```

```
PilotSkills DIVIDED BY Hangar
pilot_name
=====
'Smith'
'Wilson'
```

```
SELECT PS1.pilot_name
       FROM PilotSkills AS PS1, Hangar AS H1
       WHERE PS1.plane_name = H1.plane_name
       GROUP BY PS1.pilot_name
       HAVING COUNT(PS1.plane_name) =
       (SELECT COUNT(plane_name) FROM Hangar);
```

Multisets

Recall that SQL uses Multisets

$\lambda(X)$ = "Count of tuple in X "
(Items not listed have implicit count 0)

Multiset X

Tuple
(1, a)
(1, a)
(1, b)
(2, c)
(2, c)
(2, c)
(1, d)
(1, d)



Equivalent
Representations
of a **Multiset**

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	1
(2, c)	3
(1, d)	2

Note: In a set all counts are $\{0,1\}$.

Generalizing Set Operations to Multiset Operations

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

\cap

Multiset Y

Tuple	$\lambda(Y)$
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

$=$

Multiset Z

Tuple	$\lambda(Z)$
(1, a)	2
(1, b)	0
(2, c)	2
(1, d)	0

$$\lambda(Z) = \min(\lambda(X), \lambda(Y))$$

For sets, this is
intersection

Generalizing Set Operations to Multiset Operations

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

U

Multiset Y

Tuple	$\lambda(Y)$
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

=

Multiset Z

Tuple	$\lambda(Z)$
(1, a)	7
(1, b)	1
(2, c)	5
(1, d)	2

$$\lambda(Z) = \lambda(X) + \lambda(Y)$$

For sets,
this is **union**

Operations on Multisets

- $\sigma_C(R)$: preserve the number of occurrences
- $\Pi_A(R)$: no duplicate elimination
- Cross-product, join: no duplicate elimination

This is important-
relational engines work on multisets, not sets!

Complete Set of Relational Operations

- The set of operations including
 - Select σ ,
 - Project π
 - Union \cup
 - Difference \ominus
 - Rename ρ , and
 - Cartesian Product \times
- is called a *complete set*
- because any other relational algebra expression can be expressed by a combination of these five operations.
- For example:
 - $R \cap S = (R \cup S) - ((R - S) \cup (S - R))$
 - $R \bowtie_{\langle \text{join condition} \rangle} S = \sigma_{\langle \text{join condition} \rangle} (R \times S)$

Table 8.1 Operations of Relational Algebra

Table 8.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R .	$\sigma_{\langle \text{selection condition} \rangle}(R)$
PROJECT	Produces a new relation with only some of the attributes of R , and removes duplicate tuples.	$\pi_{\langle \text{attribute list} \rangle}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$, OR $R_1 \bowtie_{(\langle \text{join attributes 1} \rangle), (\langle \text{join attributes 2} \rangle)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1^*_{\langle \text{join condition} \rangle} R_2$, OR $R_1^*_{(\langle \text{join attributes 1} \rangle), (\langle \text{join attributes 2} \rangle)} R_2$ OR $R_1 * R_2$

continued on next slide

Table 8.1 Operations of Relational Algebra (continued)

Table 8.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2 ; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2 ; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2 ; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2 .	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Query Tree Notation

- Query Tree
 - An internal data structure to represent a query
 - Standard technique for estimating the work involved in executing the query, the generation of intermediate results, and the optimization of execution
 - Nodes stand for operations like selection, projection, join, renaming, division,
 - Leaf nodes represent base relations
 - A tree gives a good visual feel of the complexity of the query and the operations involved
 - Algebraic Query Optimization consists of rewriting the query or modifying the query tree into an equivalent tree.

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

DEPT_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum
-------	----------------	-----------	------

WORKS_ON

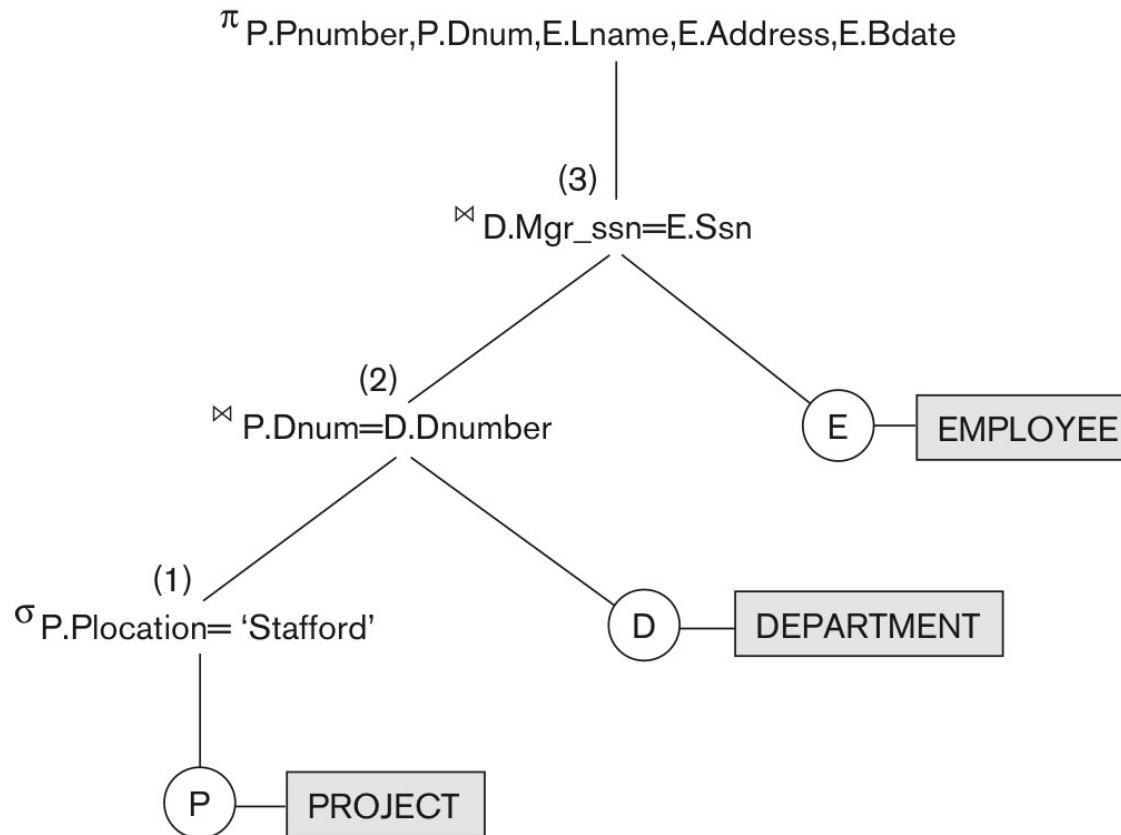
<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

Example of Query Tree

- For every project located in Stafford, list the project number, dept. number, manager's last name, address, and birth date
- (page 258)



Summary

- Total 8 basic operators:
 - Unary relational operators (3)
 - Selection: σ
 - Projection: Π
 - Renaming: ρ
 - Binary relational operators (5)
 - Union: \cup
 - Intersect: \cap
 - Set difference: $-$
 - Cartesian Product (Join): \times , \bowtie
 - Natural Join, Theta Join, Equi-Join, Semi-Join.
 - Division: \div
- Tell us: How the query may be executed.

Acknowledgement

- Some of the slides in this presentation are taken from the slides provided by the authors.
- Many of these slides are taken from cs145 course offered by Stanford University.

MIDTERM REVIEW

Chapters to Read

- Chapter 1
- Chapter 2
- Chapter 3
- Chapter 4 (Just the basics. Only ISA relationships. Even studying the slides is fine)
- Chapter 5, 6, 7 (SQL)
- Chapter 9
- Chapter 14 (14.1-14.5)
- Chapter 15 (15.1-15.3)

Exam Structure

- Problem 1 (20 pts)
 - Short answers, True/False, or One liner
- Other problems (55 pts)
- Plenty of SQL queries, ER diagram, BCNF decomposition questions.
- Total: 75 pts

Time distribution

- Time crunch.
 - Not as relaxed as the quiz

Notes:

I may curve the grades.

For example, If I decide 70 pts is equivalent to 100% score credit, I will scale all scores accordingly. If you have have scored more than 70, for example, 75, those 5 points will be treated as extra-credit for future.

MATERIALS COVERED

Questions to ponder

- Why not Lists? Why database?
- How related tables avoid problems associated with lists?

Problems with Lists

- Multiple Concepts or Themes:
 - Microsoft Excel vs Microsoft Access
- Redundancy
- Anomalies:
 - Deletion anomalies
 - Update anomalies
 - Insertion anomalies

List vs Database

- Lists do not provide information about relations!
- Break lists into tables
- Facilitates:
 - Insert
 - Delete
 - Update
- Input and Output interface (Forms and Reports)
- Query!

Again, Why database?

- To store data
- To provide structure
- Mechanism for querying, creating, modifying and deleting data.
- CRED (Create, Read, Update, Delete)
- Store information and relationships

- Database Schema vs. Database State

Simplified Database System Environment

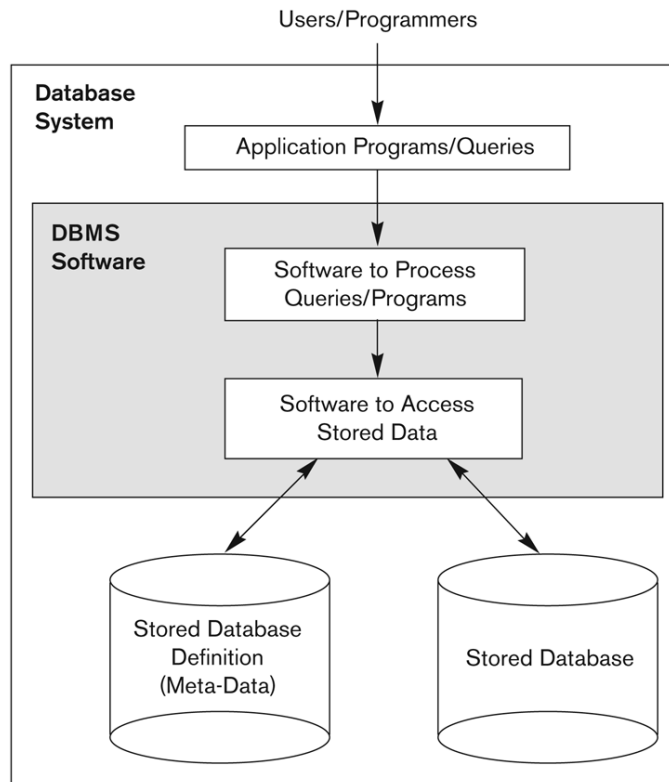


Figure 1.1
A simplified database system environment.

SQL

General form SQL

SELECT	S
FROM	R_1, \dots, R_n
WHERE	C_1
GROUP BY	a_1, \dots, a_k
HAVING	C_2

Evaluation steps:

1. Evaluate **FROM-WHERE**: apply condition C_1 on the attributes in R_1, \dots, R_n
2. **GROUP BY** the attributes a_1, \dots, a_k
3. **Apply condition C_2 to each group (may have aggregates)**
4. Compute aggregates in S and return the result

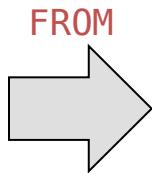
Grouping and Aggregation

Semantics of the query:

1. Compute the **FROM** and **WHERE** clauses
2. Group by the attributes in the **GROUP BY**
3. Compute the **SELECT** clause: grouped attributes and aggregates

1. Compute the **FROM** and **WHERE** clauses

```
SELECT product, SUM(price*quantity) AS TotalSales
FROM Purchase
WHERE date > '10/1/2005'
GROUP BY product
```



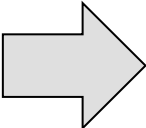
Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10

2. Group by the attributes in the **GROUP BY**

```
SELECT product, SUM(price*quantity) AS TotalSales
FROM Purchase
WHERE date > '10/1/2005'
GROUP BY product
```

Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10

GROUP BY



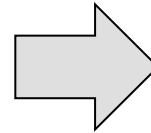
Product	Date	Price	Quantity
Bagel	10/21	1	20
	10/25	1.50	20
Banana	10/3	0.5	10
	10/10	1	10

3. Compute the **SELECT** clause: grouped attributes and aggregates

```
SELECT product, SUM(price*quantity) AS TotalSales
FROM Purchase
WHERE date > '10/1/2005'
GROUP BY product
```

Product	Date	Price	Quantity
Bagel	10/21	1	20
	10/25	1.50	20
Banana	10/3	0.5	10
	10/10	1	10

SELECT



Product	TotalSales
Bagel	50
Banana	15

HAVING Clause

```
SELECT product, SUM(price*quantity)
FROM Purchase
WHERE date > '10/1/2005'
GROUP BY product
HAVING SUM(quantity) > 100
```

HAVING clauses contains conditions on **aggregates**

*Whereas WHERE clauses condition on **individual tuples...***

Same query as before, except that we consider only products that have more than 100 buyers

Null Values

Unexpected behavior:

```
SELECT *  
FROM Person  
WHERE age < 25 OR age >= 25
```

Some Persons are not included !

Null Values

Can test for NULL explicitly:

- x IS NULL
- x IS NOT NULL

```
SELECT *  
FROM Person  
WHERE age < 25 OR age >= 25  
      OR age IS NULL
```

Now it includes all Persons!

Inner Joins

By default, joins in SQL are “inner joins”:

```
Product(name, category)
Purchase(prodName, store)
```

```
SELECT Product.name, Purchase.store
FROM Product
JOIN Purchase ON Product.name = Purchase.prodName
```

```
SELECT Product.name, Purchase.store
FROM Product, Purchase
WHERE Product.name = Purchase.prodName
```

Both equivalent:
Both INNER JOINS!

Inner Joins + NULLS = Lost data?

By default, joins in SQL are “inner joins”:

```
Product(name, category)
Purchase(prodName, store)
```

```
SELECT Product.name, Purchase.store
FROM   Product
       JOIN Purchase ON Product.name = Purchase.prodName
```

```
SELECT Product.name, Purchase.store
FROM   Product, Purchase
WHERE  Product.name = Purchase.prodName
```

However: Products that never sold (with no Purchase tuple) will be lost!

INNER JOIN:

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
SELECT Product.name, Purchase.store
FROM Product
INNER JOIN Purchase
ON Product.name = Purchase.prodName
```



name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Note: another equivalent way to write an INNER JOIN!

LEFT OUTER JOIN:

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
SELECT Product.name, Purchase.store
FROM Product
LEFT OUTER JOIN Purchase
ON Product.name = Purchase.prodName
```



name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz
OneClick	NULL

Other Outer Joins

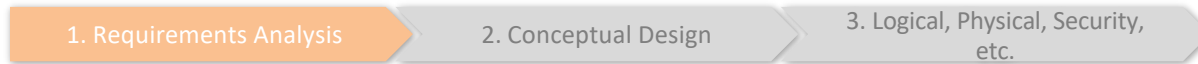
- Left outer join:
 - Include the left tuple even if there's no match
- Right outer join:
 - Include the right tuple even if there's no match
- Full outer join:
 - Include the both left and right tuples even if there's no match

Also read

- Triggers
- Views
- Operations on Databases
 - Create, Drop, and Alter
- Operations of Tables
 - Insert, Delete and Update
- Constraint:
 - Key constraint
 - Foreign key
 - On Delete cascade, Set Null, Set Default;

DATABASE DESIGN

Database Design Process

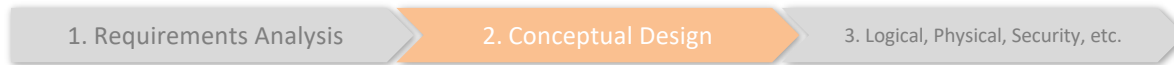


1. Requirements analysis

- What is going to be stored?
- How is it going to be used?
- What are we going to do with the data?
- Who should access the data?

Technical and non-technical people are involved

Database Design Process

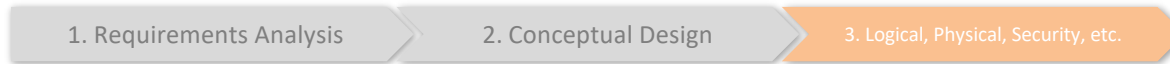


2. Conceptual Design

- A high-level description of the database
- Sufficiently precise that technical people can understand it
- But, not so precise that non-technical people can't participate

This is where E/R fits in.

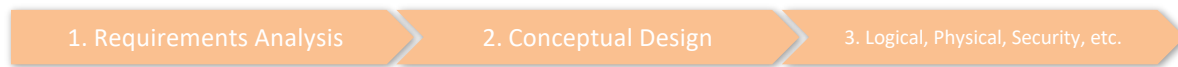
Database Design Process



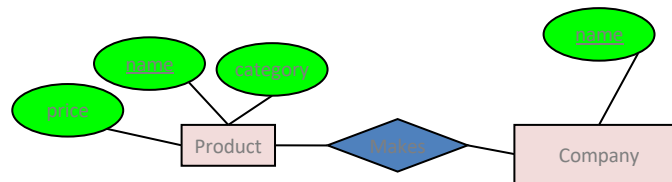
3. Implementation:

- Logical Database Design
- Physical Database Design
- Security Design

Database Design Process



E/R Model & Diagrams used



This process is iterated **many** times

E/R is a *visual syntax* for DB design which is ***precise enough*** for technical points, but ***abstracted enough*** for non-technical people

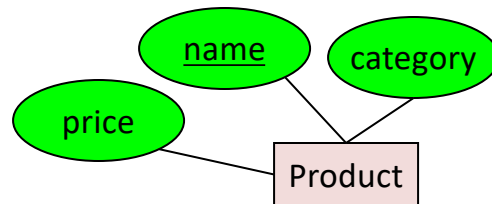
Requirements Become the E-R Data Model

- After the requirements have been gathered, they are transformed into an Entity Relationship (E-R) Data Model
- E-R Models consist of
 1. Entities
 2. Attributes
 - a) Identifiers (Keys)
 - b) Non-key attributes
 3. Relationships

1. E/R BASICS: ENTITIES & RELATIONS

Entities and Entity Sets

- An entity set has **attributes**
 - Represented by ovals
attached to an entity set

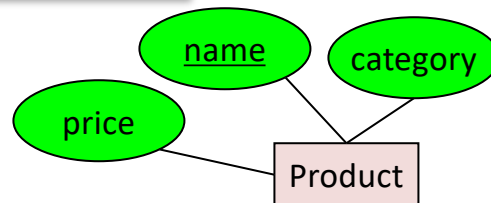


Shapes are important. Colors are not.

Keys

- A key is a **minimal** set of attributes that uniquely identifies an entity.

Denote elements of the primary key by underlining.



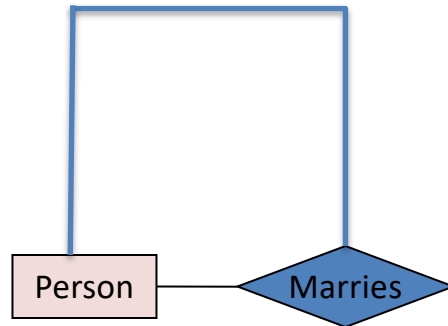
Here, {name, category} is **not** a key (it is not *minimal*).

The E/R model forces us to designate a single **primary** key, though there may be multiple candidate keys

Relationships

- A **relationship** connects two or more entity sets.
- It is represented by a **diamond**, with lines to each of the entity sets involved.
- The *degree* of the relationship defines the number of entity classes that participate in the relationship
 - Degree 1 is a **unary** relationship
 - Degree 2 is a **binary** relationship
 - Degree 3 is a **ternary** relationship

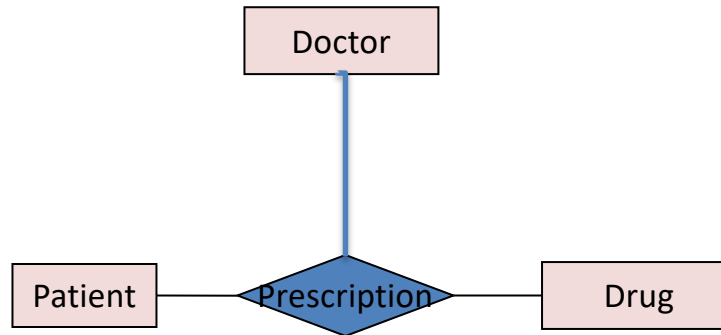
Conceptual Unary Relationship



Conceptual Binary Relationship

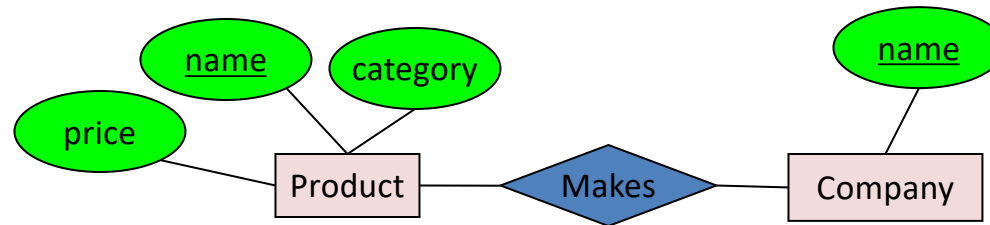


Conceptual Ternary Relationship

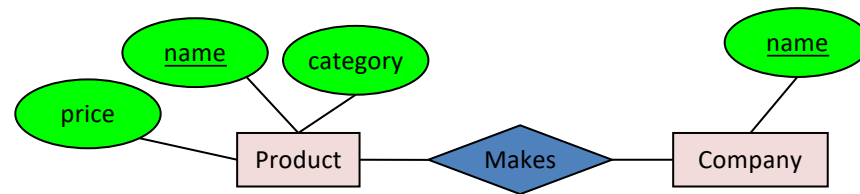


The R in E/R: Relationships

- E, R and A together:



What is a Relationship?



A **relationship** between entity sets **P** and **C** is a **subset of all possible pairs of entities in P and C**, with tuples uniquely identified by **P and C's keys**

What is a Relationship?

Company

<u>name</u>
GizmoWorks
GadgetCorp

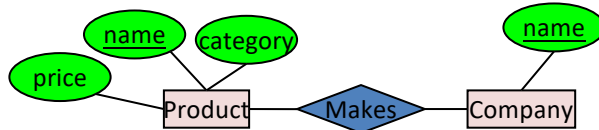
Product

<u>name</u>	category	price
Gizmo	Electronics	\$9.99
GizmoLite	Electronics	\$7.50
Gadget	Toys	\$5.50



Company C × Product P

<u>C.name</u>	<u>P.name</u>	P.category	P.price
GizmoWorks	Gizmo	Electronics	\$9.99
GizmoWorks	GizmoLite	Electronics	\$7.50
GizmoWorks	Gadget	Toys	\$5.50
GadgetCorp	Gizmo	Electronics	\$9.99
GadgetCorp	GizmoLite	Electronics	\$7.50
GadgetCorp	Gadget	Toys	\$5.50



A **relationship** between **entity sets P and C** is a **subset of all possible pairs of entities in P and C**, with tuples uniquely identified by **P and C's keys**

What is a Relationship?

Company

<u>name</u>
GizmoWorks
GadgetCorp

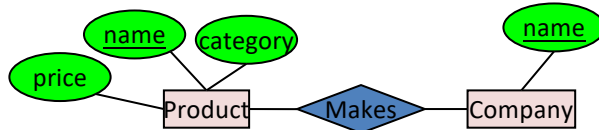
Product

<u>name</u>	category	price
Gizmo	Electronics	\$9.99
GizmoLite	Electronics	\$7.50
Gadget	Toys	\$5.50



Company C × Product P

<u>C.name</u>	<u>P.name</u>	P.category	P.price
GizmoWorks	Gizmo	Electronics	\$9.99
GizmoWorks	GizmoLite	Electronics	\$7.50
GizmoWorks	Gadget	Toys	\$5.50
GadgetCorp	Gizmo	Electronics	\$9.99
GadgetCorp	GizmoLite	Electronics	\$7.50
GadgetCorp	Gadget	Toys	\$5.50



A **relationship** between entity sets **P** and **C** is a **subset of all possible pairs of entities in P and C**, with tuples uniquely identified by **P and C's keys**

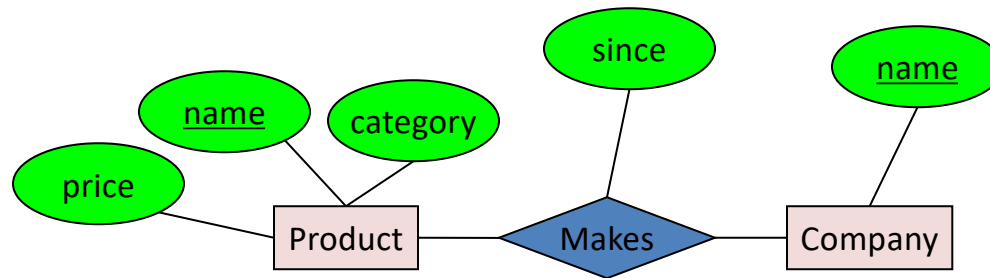


Makes

<u>C.name</u>	<u>P.name</u>
GizmoWorks	Gizmo
GizmoWorks	GizmoLite
GadgetCorp	Gadget

Relationships and Attributes

- Relationships may have attributes as well.



For example: "since" records when company started making a product

Note: "since" is implicitly unique per pair here! Why?

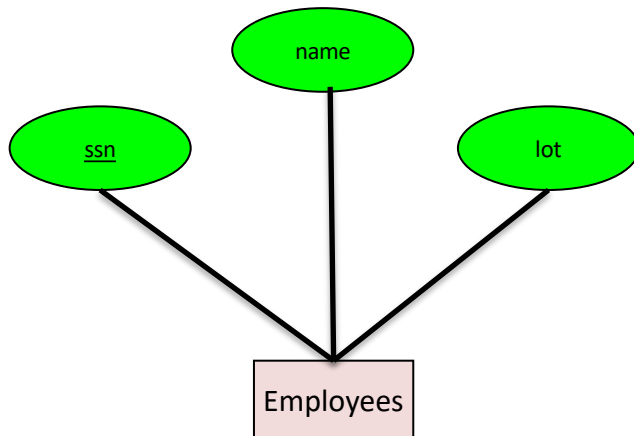
Figure 3.14
Summary of the
notation for ER
diagrams.

- Summary

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1: N for $E_1:E_2$ in R
	Structural Constraint (min, max) on Participation of E in R

Design Theory (ER model to Relations)

Entity Sets to Tables



ssn	name	lot
123-22-3333	Alex	23
234-44-6666	Bob	44
567-88-9787	John	12

```
CREATE TABLE Employees (
    ssn char(11),
    name varchar(30),
    lot Integer,
    PRIMARY KEY (ssn))
```

Other Conversions (ER model to Tables)

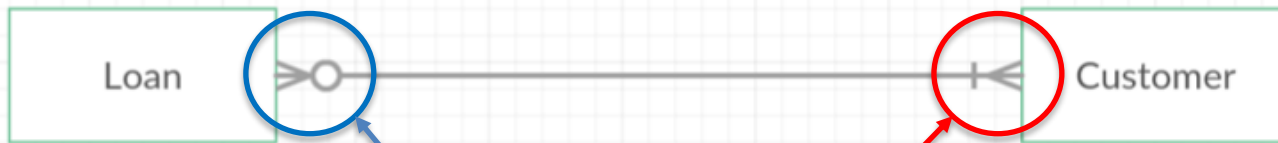
- Relationships:
- Many to Many
- One to Many
- One to One

Scenario

- One customer can have **at max 2 loans**. One loan can be given to **multiple** customers.

What it really means:

- One customer can have **(0,2)** loans
- One loan can be given to **(1,n)** customer
- This is a many to many scenario



Crow's foot Notation



Chen Notation

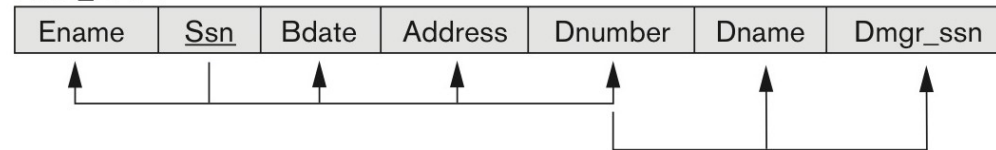
FUNCTIONAL DEPENDENCIES

Prime and Non-prime attributes

- A **Prime attribute** must be a member of *some* candidate key
- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.

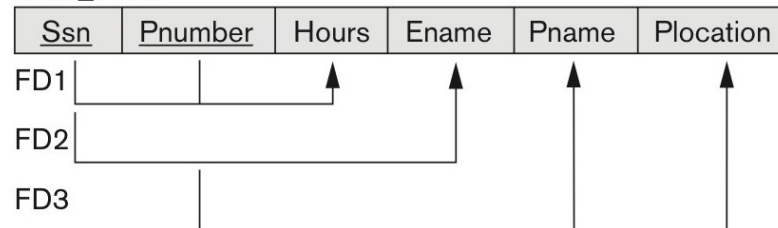
(a)

EMP_DEPT



(b)

EMP_PROJ



Back to Conceptual Design

Now that we know how to find FDs, it's a straight-forward process:

1. Search for “bad” FDs
2. If there are any, then *keep decomposing the table into sub-tables* until no more bad FDs
3. When done, the database schema is *normalized*

Boyce-Codd Normal Form (BCNF)

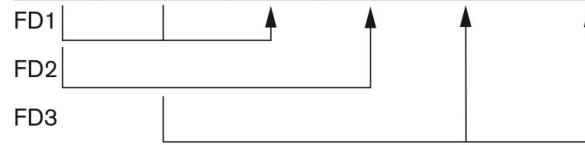
- Main idea is that we define “good” and “bad” FDs as follows:
 - $X \rightarrow A$ is a “good FD” if X is a (super)key
 - In other words, if A is the set of all attributes
 - $X \rightarrow A$ is a “bad FD” otherwise
- We will try to eliminate the “bad” FDs!
 - Via normalization

Normalizing into 2NF and 3NF

(a)

EMP_PROJ

<u>Ssn</u>	Pnumber	Hours	Ename	Pname	Plocation
------------	---------	-------	-------	-------	-----------



2NF Normalization

EP1

<u>Ssn</u>	Pnumber	Hours
------------	---------	-------



EP2

<u>Ssn</u>	Ename
------------	-------



EP3

Pnumber	Pname	Plocation
---------	-------	-----------



(b)

EMP_DEPT

Ename	<u>Ssn</u>	Bdate	Address	Dnumber	Dname	Dmgr_ssn
-------	------------	-------	---------	---------	-------	----------



3NF Normalization

ED1

Ename	<u>Ssn</u>	Bdate	Address	Dnumber
-------	------------	-------	---------	---------



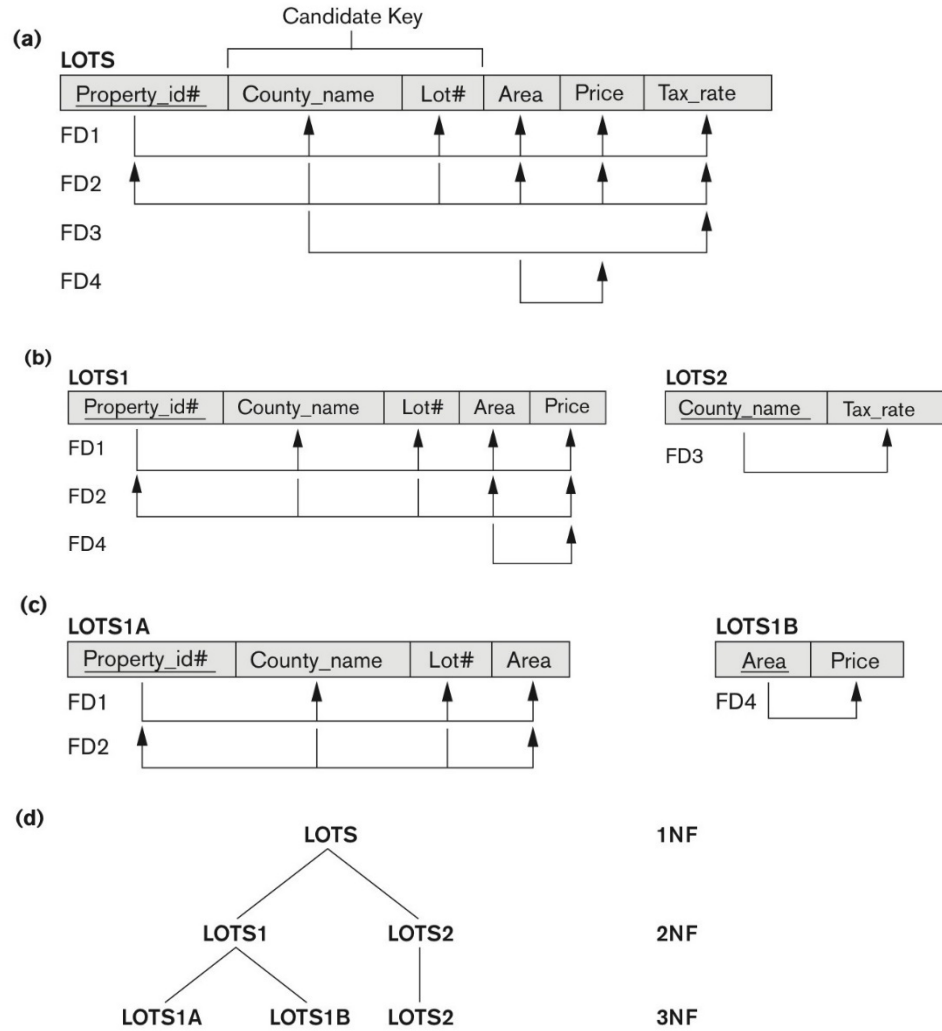
ED2

<u>Dnumber</u>	Dname	Dmgr_ssn
----------------	-------	----------



Figure 14.12 Normalization into 2NF and 3NF

Figure 14.12
 Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Progressive normalization of LOTS into a 3NF design.



Normal Forms Defined Informally

- 1st normal form
 - All attributes depend on **the key**
- 2nd normal form
 - All attributes depend on **the whole key**
- 3rd normal form
 - All attributes depend on **nothing but the key**

General Definition of 2NF and 3NF (For Multiple Candidate Keys)

- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on **every** key of R
- A relation schema R is in **third normal form (3NF)** if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on **any** key of R

1. BOYCE-CODD NORMAL FORM

Figure 14.14 A relation TEACH that is in 3NF but not in BCNF

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

- Two FDs exist in the relation TEACH:
 - fd₁: { student, course } -> instructor
 - fd₂: instructor -> course
- {student, course} is a candidate key for this relation
- So this relation is in 3NF *but not in BCNF*
- A relation **NOT** in BCNF should be decomposed so as to meet this property,
 - while possibly forgoing the preservation of all functional dependencies in the decomposed relations.

Achieving the BCNF by Decomposition (2)

- Three possible decompositions for relation TEACH
 - D1: {student, instructor} and {student, course}
 - D2: {course, instructor } and {course, student}
 - D3: {instructor, course } and {instructor, student} ✓

4.3 Interpreting the General Definition of Third Normal Form (2)

■ **ALTERNATIVE DEFINITION of 3NF:** We can restate the definition as:

A relation schema R is in **third normal form (3NF)** if, whenever a nontrivial FD $X \rightarrow A$ holds in R , either

- a) X is a superkey of R or
- b) A is a prime attribute of R

The condition (b) takes care of the dependencies that “slip through” (are allowable to) 3NF but are “caught by” BCNF which we discuss next.

1. BOYCE-CODD NORMAL FORM

What you will learn about in this section

1. Boyce-Codd Normal Form
2. The BCNF Decomposition Algorithm

5. BCNF (Boyce-Codd Normal Form)

- **Definition of 3NF:**
- A relation schema R is in **3NF** if, whenever a nontrivial FD $X \rightarrow A$ holds in R, either
 - a) X is a superkey of R or
 - b) A is a prime attribute of R
- A relation schema R is in **Boyce-Codd Normal Form (BCNF)** if whenever an FD $X \rightarrow A$ holds in R, then
 - a) X is a superkey of R
 - ~~b) There is no b~~
- Each normal form is strictly stronger than the previous one
 - Every 2NF relation is in 1NF
 - Every 3NF relation is in 2NF
 - Every BCNF relation is in 3NF

Boyce-Codd normal form

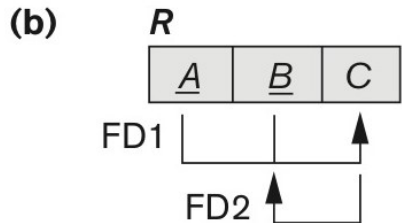
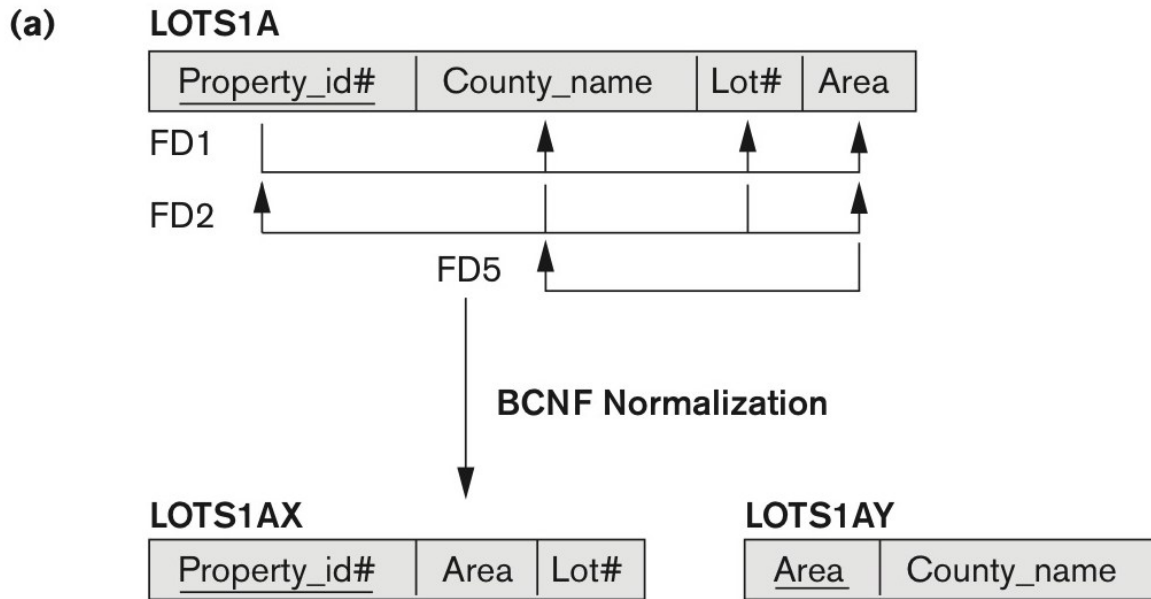


Figure 14.13
 Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF due to the f.d. $C \rightarrow B$.

A relation TEACH that is in 3NF but not in BCNF

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

- Two FDs exist in the relation TEACH:

X

→ A

- {student, course} → instructor
- instructor → course

- {student, course} is a candidate key for this relation
- So this relation is in **3NF** *but not in BCNF*
- A relation **NOT** in **BCNF** should be decomposed

Achieving the BCNF by Decomposition

- Three possible decompositions for relation TEACH
 - D1: {student, instructor} and {student, course}
 - D2: {course, instructor } and {course, student}
 - ✓ D3: {instructor, course } and {instructor, student}

Boyce-Codd Normal Form

BCNF is a simple condition for removing anomalies from relations:

A relation R is in BCNF if:

if $\{X_1, \dots, X_n\} \rightarrow A$ is a *non-trivial* FD in R

then $\{X_1, \dots, X_n\}$ is a **superkey** for R

In other words: there are no “bad” FDs

Example

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

$\{SSN\} \rightarrow \{Name, City\}$

This FD is *bad*
because it is **not** a
superkey

\Rightarrow **Not** in BCNF

What is the key?
{SSN, PhoneNumber}

Example

Name	<u>SSN</u>	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Madison

<u>SSN</u>	<u>PhoneNumber</u>
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

{SSN} → {Name, City}

This FD is now
good because it is
the key

Let's check anomalies:

- Redundancy ?
- Update ?
- Delete ?

Now in BCNF!

BCNF Decomposition

BCNFDecomp(R):

If $X \rightarrow A$ causes BCNF violation:

Decompose R into

$R_1 = XA$

$R_2 = R - A$

(Note: X is present in both R1 and R2)

Return BCNFDecomp(R1), BCNFDecomp(R2)

Example

BCNFDecomp(R):

If $X \rightarrow A$ causes BCNF violation:

Decompose R into

$R_1 = XA$

$R_2 = R - A$

(Note: X is present in both R_1 and R_2)

Return BCNFDecomp(R_1),
BCNFDecomp(R_2)

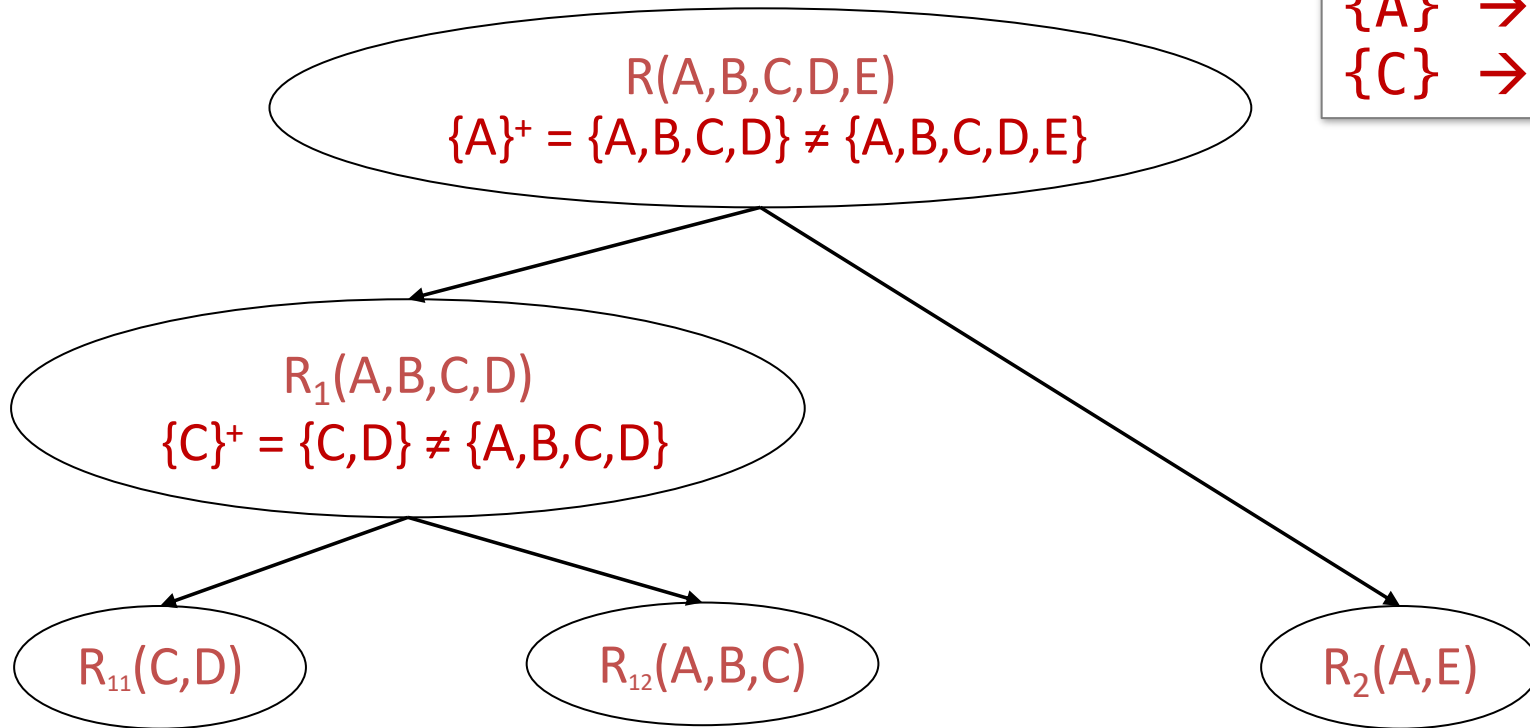
$R(A, B, C, D, E)$

$\{A\} \rightarrow \{B, C\}$
 $\{C\} \rightarrow \{D\}$

Example

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

$\{A\} \rightarrow \{B, C\}$
 $\{C\} \rightarrow \{D\}$



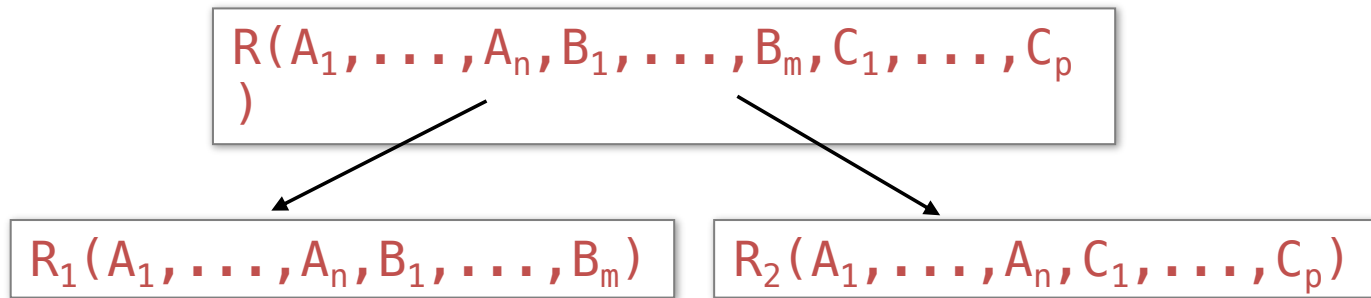
2. DECOMPOSITIONS

Recap: Decompose to remove redundancies

1. We saw that **redundancies** in the data (“bad FDs”) can lead to data anomalies
2. We developed mechanisms to **detect and remove redundancies by decomposing tables into BCNF**
 1. BCNF decomposition is *standard practice*- very powerful & widely used!
3. However, sometimes decompositions can lead to **more subtle unwanted effects...**

When does this happen?

Decompositions in General



R_1 = the *projection* of R on $A_1, \dots, A_n, B_1, \dots, B_m$


R_2 = the *projection* of R on $A_1, \dots, A_n, C_1, \dots, C_p$

Theory of Decomposition


Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Sometimes a decomposition is “correct”

I.e. it is a Lossless decomposition



Name	Price
Gizmo	19.99
OneClick	24.99
Gizmo	19.99



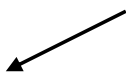
Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Lossy Decomposition

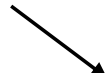
Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

*However
sometimes it isn't*

What's wrong
here?

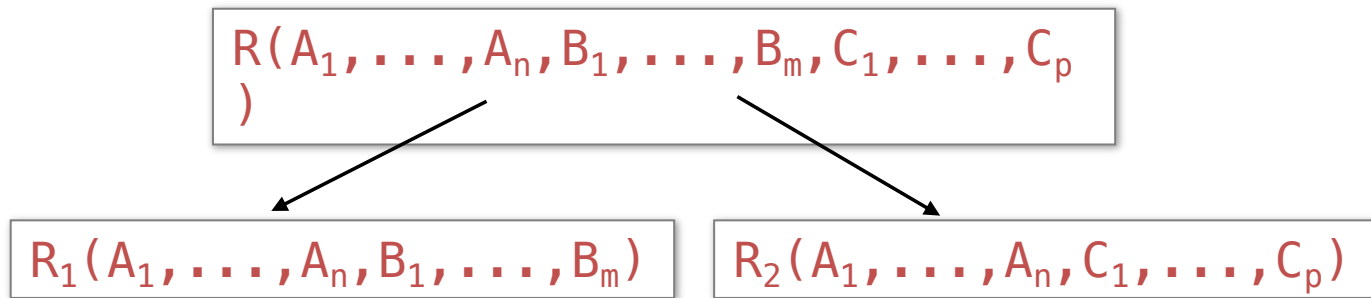


Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera



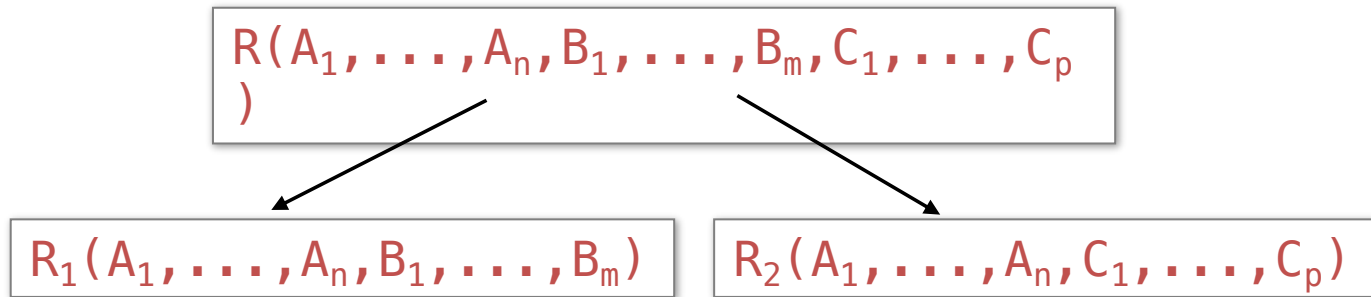
Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

Lossless Decompositions



A decomposition R to (R_1, R_2) is **lossless** if $R = R_1 \text{ Join } R_2$

Lossless Decompositions



If $\{A_1, \dots, A_n\} \rightarrow \{B_1, \dots, B_m\}$
Then the decomposition is lossless

Note: don't need
 $\{A_1, \dots, A_n\} \rightarrow \{C_1, \dots, C_p\}$

BCNF decomposition is always lossless. Why?

A relation TEACH that is in 3NF but not in BCNF

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

- Two FDs exist in the relation TEACH:

X

→ A

- {student, course} → instructor
- instructor → course

- {student, course} is a candidate key for this relation
- So this relation is in 3NF *but not in* BCNF
- A relation **NOT** in BCNF should be decomposed

Achieving the BCNF by Decomposition (2)

- Three possible decompositions for relation TEACH
 - D1: {student, instructor} and {student, course}
 - D2: {course, instructor } and {course, student}
 - ✓ D3: {instructor, course } and {instructor, student}

A problem with BCNF

Problem: To enforce a FD, must reconstruct original relation—*on each insert!*

A Problem with BCNF

Unit	Company	Product
...

$\{\text{Unit}\} \rightarrow \{\text{Company}\}$
 $\{\text{Company}, \text{Product}\} \rightarrow \{\text{Unit}\}$

<u>Unit</u>	Company
...	...

Unit	Product
...	...

We do a BCNF decomposition
on a “bad” FD:
 $\{\text{Unit}\}^+ = \{\text{Unit}, \text{Company}\}$

$\{\text{Unit}\} \rightarrow \{\text{Company}\}$

We lose the FD $\{\text{Company}, \text{Product}\} \rightarrow \{\text{Unit}\}!!$

So Why is that a Problem?

<u>Unit</u>	Company
Galaga99	UW
Bingo	UW

Unit	Product
Galaga99	Databases
Bingo	Databases

No problem so far.
All *local* FD's are satisfied.

$\{\text{Unit}\} \rightarrow \{\text{Company}\}$

Unit	Company	Product
Galaga99	UW	Databases
Bingo	UW	Databases

Let's put all the data back into a single table again:

Violates the FD $\{\text{Company, Product}\} \rightarrow \{\text{Unit}\}!!$

The Problem

- We started with a table R and FDs F
- We decomposed R into BCNF tables R_1, R_2, \dots with their own FDs F_1, F_2, \dots
- We insert some tuples into each of the relations—which satisfy their local FDs but when reconstruct it violates some FD **across** tables!

Practical Problem: To enforce FD, must reconstruct R —*on each insert!*

Possible Solutions

- Various ways to handle so that decompositions are all lossless / no FDs lost
 - For example 3NF- stop short of full BCNF decompositions.
- Usually a tradeoff between redundancy / data anomalies and FD preservation...

BCNF still most common- with additional steps to keep track of lost FDs...

Other Topics

- **Problem Set 5** (Really important)
 - Cover
 - Minimal Cover
 - BCNF violations and Decomposition