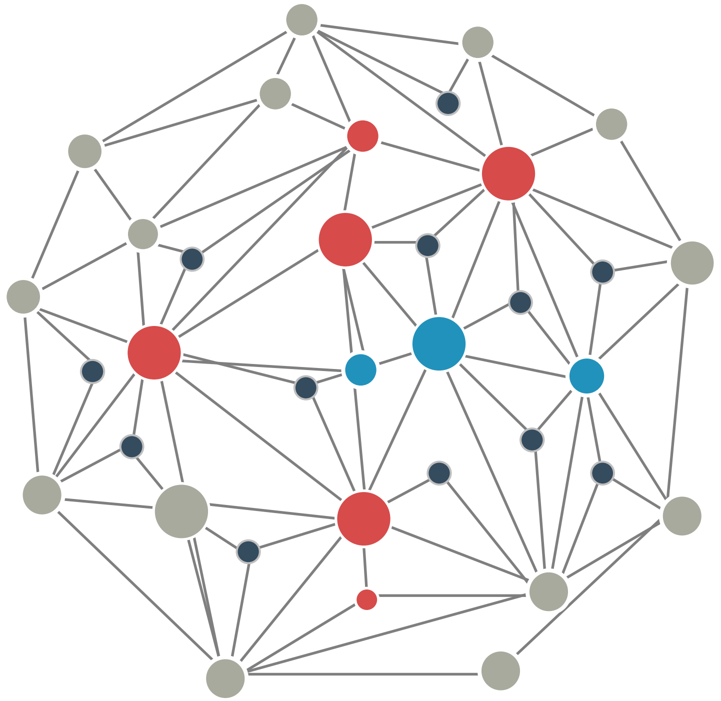
DST2 – Week 12

Database Normalization and Indexing



Zhaoyuan Fang

zhaoyuanfang@intl.zju.edu.cn

2020/12/17

Week 12 Learning Objectives

- Database design after ERD
- Database normalization: What is it?
- 1NF, 2NF, 3NF: What are they?
- Database index: Why we need indexing?

Database design

- Conceptual design (ERD)
  - Entity types: ordinary entities, associative entities, weak entities, …
  - Attribute types: unique attributes, composite attributes, multivalued attributes, derived attributes, optional attributes
  - Relationship types: one-to-one, one-to-many, many-to-one, many-to-many
- The significance of ERD
  - Identification of entities and relationships
  - Grouping of attributes into relations naturally
- Remaining tasks
  - How to evaluate whether the design is good enough
  - Information preservation + minimal redundancy => “Normalization”

Normalization

- **Normalization** - process to improve the design of relational databases
- **Normal form** - a set of particular conditions upon table structures
  - Main purpose: reducing data redundancy while preserving information
- There are several normal forms:
  - First normal form (1NF)
  - Second normal form (2NF)
  - Third normal form (3NF)
  - …

Normalization – action

- From a lower to a higher normal form, these conditions are increasingly stricter and leave less possibility for redundant data
  - 1NF -> 2NF -> 3NF -> …
- The normalization process involves examining each table and verifying if it satisfies a particular normal form
- If a table satisfies a particular normal form, then the next step is to verify if that relation satisfies the next **higher** normal form
- If a table does not satisfy a particular normal form, actions are taken to **convert** the table into a set of tables that satisfy the particular normal form

Normalization – action

- Normalizing to *first normal form* is done on **non-relational tables** in order to convert them to relational tables
- Normalizing to *subsequent normal forms* (e.g., second normal form, third normal form) **improves the design** of relational tables that contain redundant information and alleviates the problem of update anomalies

Normalization – 1NF

- **First Normal Form (1NF) -** *A table is in 1NF if each row is unique and atomic (not containing multiple values)*
  - *Each row must be unique and not duplicated*
  - *Within each row, each value in each column must be single valued*

- To satisfy 1NF, **non-relational table** should be converted to a **relation**
  - Duplicates in rows => remove duplicated rows
  - Multivalued columns => eliminate them (in several ways)

Normalization – 1NF (example)

- Example: **Related multivalued columns**
  - In this case, several columns in a table refer to the same entity
  - There can be multiple values per record

VET CLINIC CLIENT

ClientID	ClientName	PetNo	PetName	PetType
111	Lisa	1	Tofu	Dog
222	Lydia	1	Fluffy	Dog
		2	JoJo	Bird
		3	Ziggy	Snake
333	Jane	1	Fluffy	Cat
		2	Cleo	Cat

Normalization – 1NF (example)

- How to correct this non-relational table?

VET CLINIC CLIENT

ClientID	ClientName	PetNo	PetName	PetType
111	Lisa	1	Tofu	Dog
222	Lydia	1 2 3	Fluffy JoJo Ziggy	Dog Bird Snake
333	Jane	1 2	Fluffy Cleo	Cat Cat

VET CLINIC CLIENT

ClientID	ClientName	PetNo	PetName	PetType
111	Lisa	1	Tofu	Dog
222	Lydia	1	Fluffy	Dog
222	Lydia	2	JoJo	Bird
222	Lydia	3	Ziggy	Snake
333	Jane	1	Fluffy	Cat
333	Jane	2	Cleo	Cat

DEMO

Solution 1:
Separate each multivalued record into individual rows

Normalization – 1NF (example)

- How to correct this non-relational table?

VET CLINIC CLIENT

ClientID	ClientName	PetNo	PetName	PetType
111	Lisa	1	Tofu	Dog
222	Lydia	1 2 3	Fluffy JoJo Ziggy	Dog Bird Snake
333	Jane	1 2	Fluffy Cleo	Cat Cat

VET CLINIC CLIENT

ClientID	ClientName
111	Lisa
222	Lydia
333	Jane

PET

ClientID	PetNo	PetName	PetType
111	1	Tofu	Dog
222	1	Fluffy	Dog
222	2	JoJo	Bird
222	3	Ziggy	Snake
333	1	Fluffy	Cat
333	2	Cleo	Cat

DEMO

Solution 2:
Create a new separate table for each group of related multivalued columns (convert to a new **weak entity**)

Normalization – 1NF (task)

Task 1 :
How to fix the table based on 1NF rule?

STU_ID	STU_Name	STU_Phone	STU_STATE	STU_Nation
1	Ram	971621721, 98717178	Haryana	India
2	Ram	9898297281	Punjab	India
3	Suresh		Punjab	India

STU_ID	STU_Name	STU_Phone	STU_STATE	STU_Nation
1	Ram	971621721	Haryana	India
1	Ram	98717178	Haryana	India
2	Ram	9898297281	Punjab	India
3	Suresh		Punjab	India

Normalization – 1NF (task)

Task 2 : Non-relational table with two groups of related multivalued columns. What are the two groups? How to normalize the table to 1NF?

VET CLINIC CLIENT

Client’s household members

ClientID	ClientName	PetNo	PetName	PetType	HHMember	Name	Relation
111	Lisa	1	Tofu	Dog	1 2 3	Joe Sally Clyde	Husband Daughter Son
222	Lydia	1 2 3	Fluffy JoJo Ziggy	Dog Bird Snake	1 2	Bill Lilly	Husband Daughter
333	Jane	1 2	Fluffy Cleo	Cat Cat	1	Jill	Sister

Normalization – 1NF (task solution)

Task 2 : Non-relational table with two groups of related multivalued columns. What are the two groups? How to normalize the table to 1NF?

Group 1 (pet) columns: PetNo, PetName, and PetType
Group 2 (household member) columns: HHMember, Name, and Relation

VET CLINIC CLIENT

ClientID	ClientName
111	Lisa
222	Lydia
333	Jane

PET

ClientID	PetNo	PetName	PetType
111	1	Tofu	Dog
222	1	Fluffy	Dog
222	2	JoJo	Bird
222	3	Ziggy	Snake
333	1	Fluffy	Cat
333	2	Cleo	Cat

HOUSEHOLD MEMBER

ClientID	HHMember	Name	Relation
111	1	Joe	Husband
111	2	Sally	Daughter
111	3	Clyde	Son
222	1	Bill	Husband
222	2	Lilly	Daughter
333	1	Jill	Sister

Normalization – 2NF

- Second Normal Form (2NF)
  - A table is in 1NF and does not contain partial dependencies
- Partial dependencies
  - A column of a relation is functionally dependent on a portion of a composite primary key

DEMO:

<u>Stu_ID</u>	<u>Course_ID</u>	Score	Stu_Name	Course_Name
S00001	MA00003	90	Smith	Mathematics II
S00001	PH00002	87	Smith	Physics I
S00002	PH00002	81	Bob	Physics I
S00002	BO00005	93	Bob	Biology III
S00003	BO00005	96	Eva	Biology III
..	…		…	…

Normalization – 2NF

- Second Normal Form (2NF)
  - If a relation has a single-column primary key, then there is no possibility of partial functional dependencies (already in 2NF)
  - If a relation with a composite primary key has partial dependencies, then it is not in 2NF and needs normalization

Normalization – 2NF

- **Normalization to Second Normal Form (2NF)**
  - Creates an additional relation for each set of partial dependencies
    - The portion of the primary key in partial dependency => primary key of the new table (becomes a foreign key in original table)
    - The columns determined in partial dependency => columns of the new table (removed from original table)
  - The original table remains after the process of normalizing to 2NF, but it no longer contains the partially dependent columns

Normalization – 2NF (example)

DEMO:

Stu_ID	Course_ID	Score	Stu_Name	Course_Name
S00001	MA00003	90	Smith	Mathematics II
S00001	PH00002	87	Smith	Physics I
S00002	PH00002	81	Bob	Physics I
S00002	BO00005	93	Bob	Biology III
S00003	BO00005	96	Eva	Biology III

Stu_ID	Course_ID	Score
S00001	MA00003	90
S00001	PH00002	87
S00002	PH00002	81
S00002	BO00005	93
S00003	BO00005	96

Stu_ID	Stu_Name
S00001	Smith
S00002	Bob
S00003	Eva

Course_ID	Course_Name
MA00003	Mathematics II
PH00002	Physics I
BO00005	Biology III

Normalization to 2NF:

1) Find partial dependencies
2) For each partial dependency, move to a new table (remove duplicates)
3) Keep the keys in partial dependency in original table as foreign keys

Normalization – 2NF (example)

After normalization to 2NF
1. Moved redundant data to separate tables
2. Created relationship between those tables using foreign keys

Problems of data redundancy:
1. Disk space wastage
2. Data inconsistency in updates
3. SQL queries can be slow

Stu_ID	**Course_ID**	Score
S00001	MA00003	90
S00001	PH00002	87
S00002	PH00002	81
S00002	BO00005	93
S00003	BO00005	96

Stu_ID	Stu_Name
S00001	Smith
S00002	Bob
S00003	Eva

Course_ID	Course_Name
MA00003	Mathematics II
PH00002	Physics I
BO00005	Biology III

Normalization – 2NF (example)

AD CAMPAIGN MIX

AdCampaignID	AdCampaignName	StartDate	Duration	Campaign MgrID	Campaign MgrName	ModelD	Media	Range	BudgetPctg
111	SummerFun13	6.6.2013	12 days	CM100	Roberta	1	TV	Local	50%
111	SummerFun13	6.6.2013	12 days	CM100	Roberta	2	TV	National	50%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	1	TV	Local	60%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	3	Radio	Local	30%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	5	Print	Local	10%
333	FallBall13	6.9.2013	12 days	CM102	John	3	Radio	Local	80%
333	FallBall13	6.9.2013	12 days	CM102	John	4	Radio	National	20%
444	AutmnStyle13	6.9.2013	5 days	CM103	Nancy	6	Print	National	100%
555	AutmnColors13	6.9.2013	3 days	CM100	Roberta	3	Radio	Local	100%

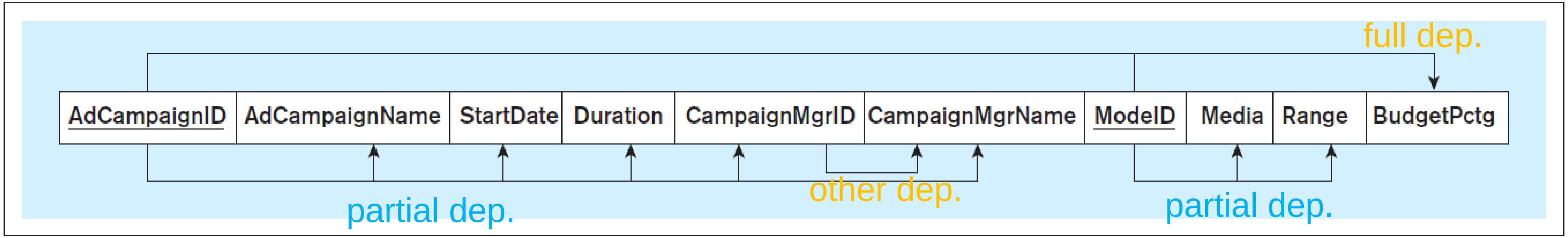
The Pressly Ad Agency manages ad campaigns through a variety of campaign modes.
- **Campain**: each ad campaign has a unique identifier, a unique name, a start date, a duration, and a campaign manager who has a name and a unique identifier.
- **Mode**: each ad campaign can use a number of different campaign modes, e.g. TV@local
- **Budget allocation**: if a campaign uses multiple campaign modes, a percentage of the total budget is allocated for each mode.

Normalization – 2NF (example)

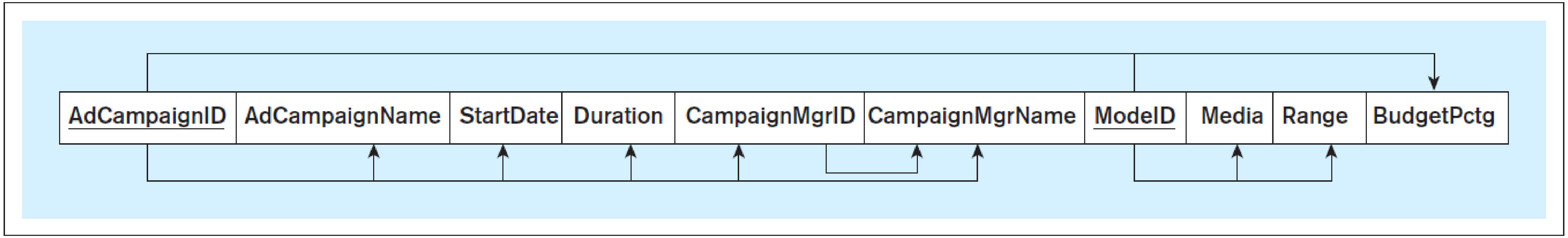
AD CAMPAIGN MIX

AdCampaignID	AdCampaignName	StartDate	Duration	Campaign MgrID	Campaign MgrName	ModelD	Media	Range	BudgetPctg
111	SummerFun13	6.6.2013	12 days	CM100	Roberta	1	TV	Local	50%
111	SummerFun13	6.6.2013	12 days	CM100	Roberta	2	TV	National	50%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	1	TV	Local	60%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	3	Radio	Local	30%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	5	Print	Local	10%
333	FallBall13	6.9.2013	12 days	CM102	John	3	Radio	Local	80%
333	FallBall13	6.9.2013	12 days	CM102	John	4	Radio	National	20%
444	AutmnStyle13	6.9.2013	5 days	CM103	Nancy	6	Print	National	100%
555	AutmnColors13	6.9.2013	3 days	CM100	Roberta	3	Radio	Local	100%

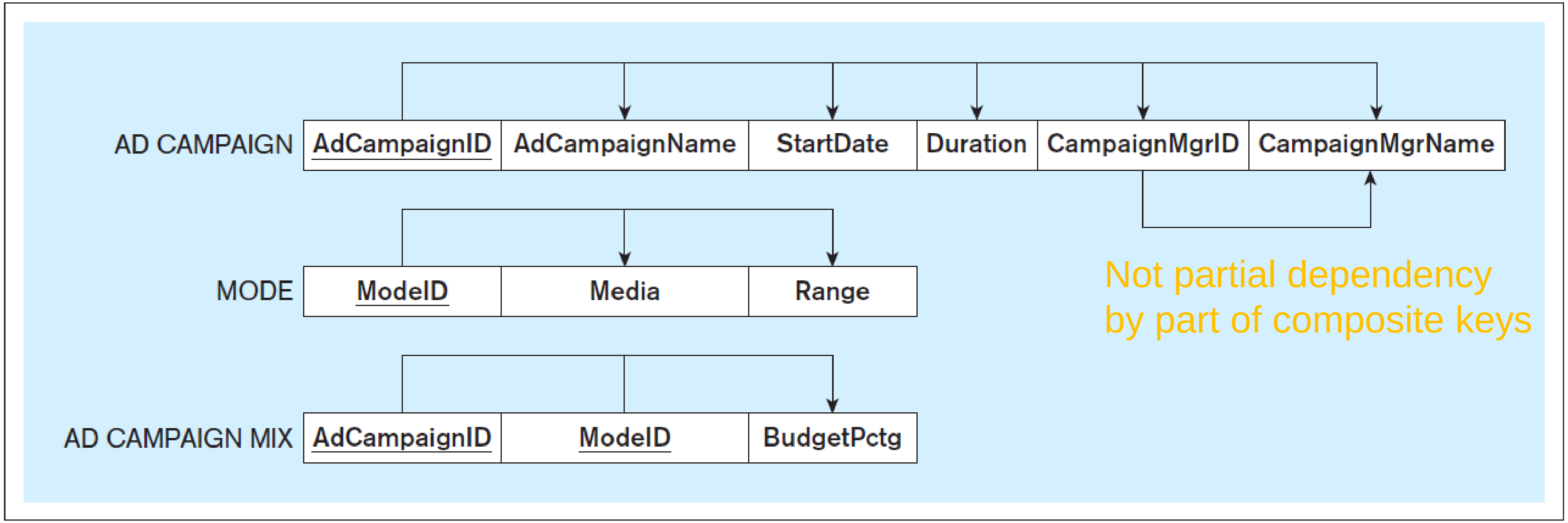
Composite primary keys, and all kinds of dependencies:



Normalization – 2NF (example)



DEMO: Pressly Ad Agency example - normalized to 2NF



Normalization – 2NF (task)

Task 3 : Normalize the following table based on 2NF rule.

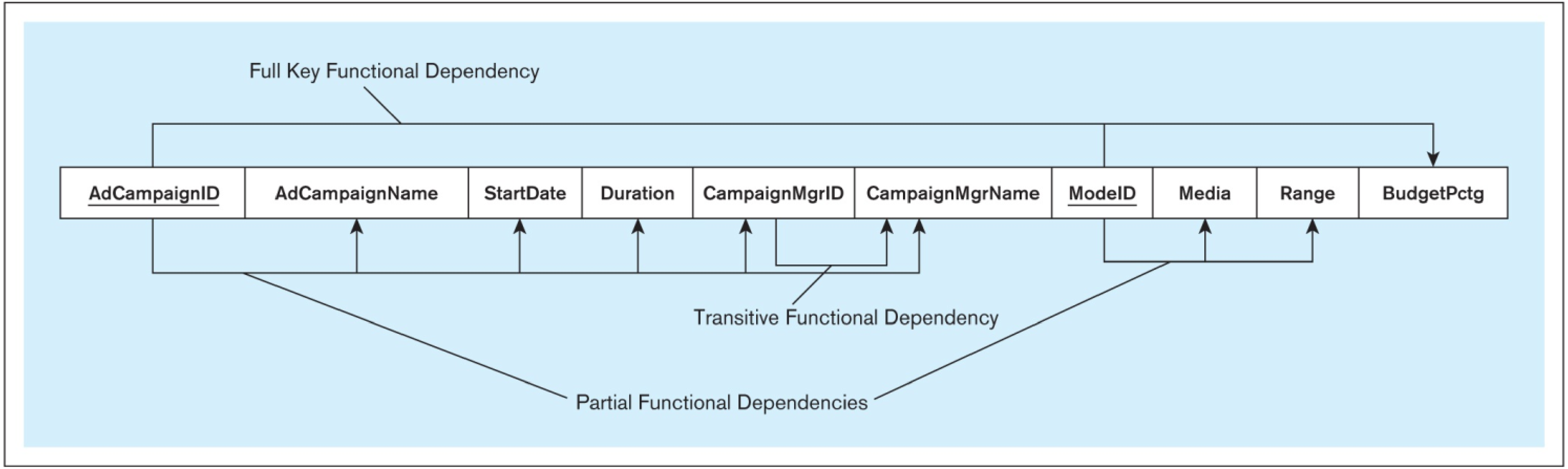
EmpID	EmpName	Gender	Salary	DeptID	DeptName	DeptHead	DeptLocation
1	Sam	M	4500	1	IT	John	London
2	Pam	F	2300	2	HR	Mike	Sydney
3	Simon	M	1345	1	IT	John	London
4	Mary	F	2567	2	HR	Mike	Sydney
5	Todd	M	6890	1	IT	John	London

DeptID	DeptName	DeptHead	DeptLocation
1	IT	John	London
2	HR	Mike	Sydney

EmpID	EmpName	Gender	Salary	DeptID
1	Sam	M	4500	1
2	Pam	F	2300	2
3	Simon	M	1345	1
4	Mary	F	2567	2
5	Todd	M	6890	1

Normalization – 3NF

- **Third Normal Form (3NF) -** *A table is in 3NF if it is in 2NF and if it* ***does not contain transitive functional dependencies***
- **Transitive functional dependency**
  - nonkey columns functionally determine other nonkey columns

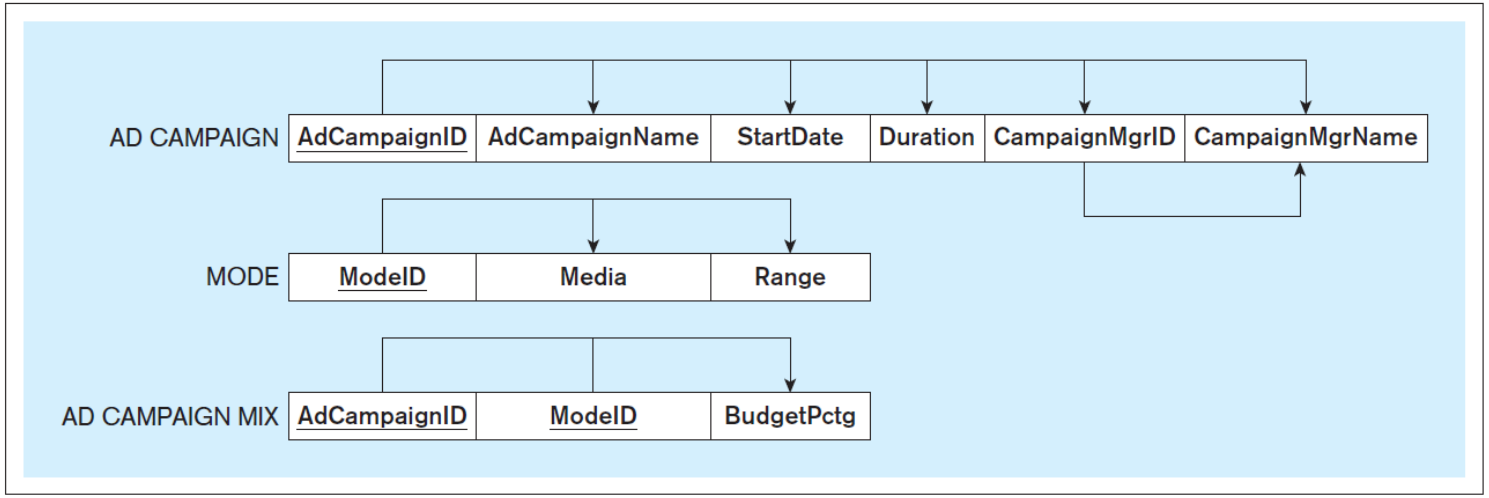


Normalization – 3NF

- Normalization of a relation to 3NF creates additional relations for each set of transitive dependencies in a relation
  - The transitively determinant nonkey column in the original table => the primary key of the new table
  - Move the determined nonkey columns to the new table
- The original table remains after normalizing to 3NF, but it no longer contains the transitively dependent columns

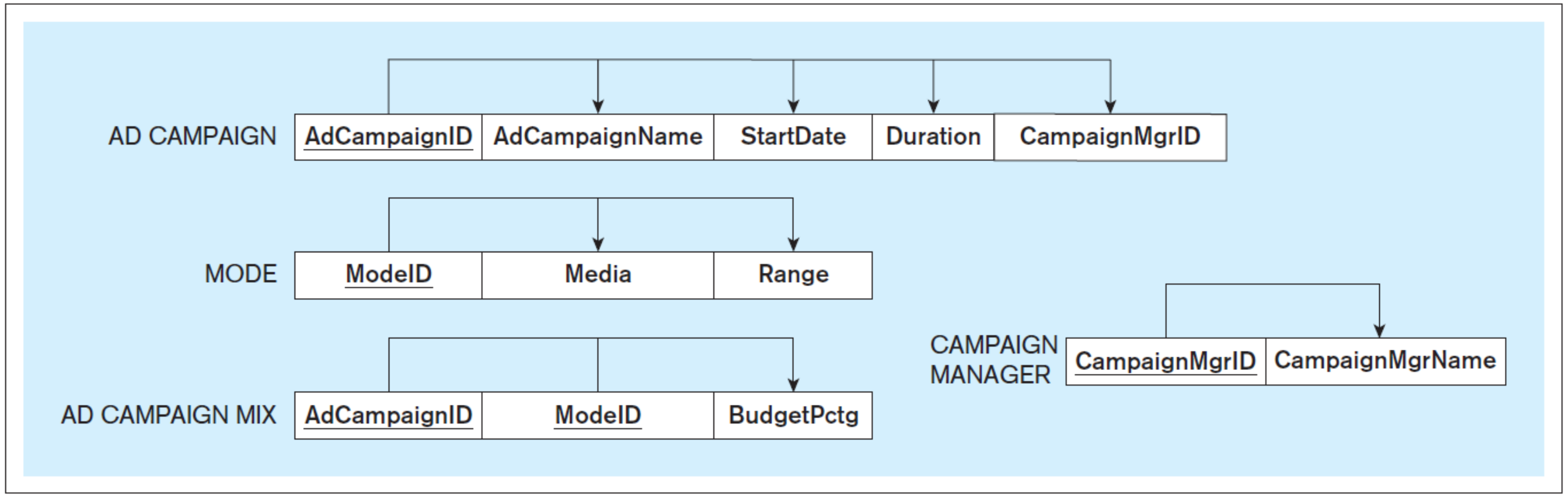
Normalization – 3NF (example)

Pressly Ad Agency example - normalized to **2NF**



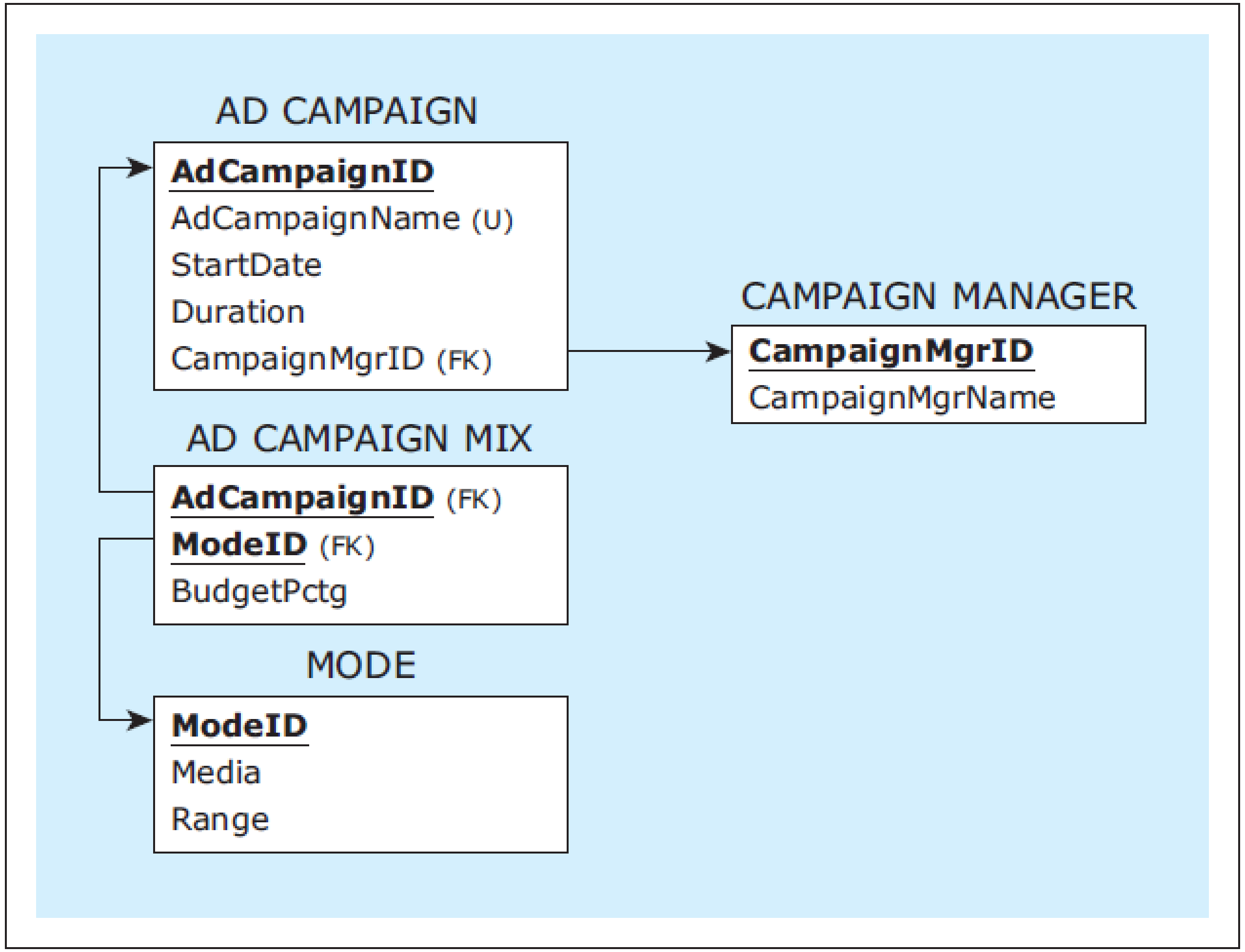
DEMO:

Pressly Ad Agency example - normalized to **3NF**



Normalization – 3NF (example)

Pressly Ad Agency example – relational schema of 3NF relations



Normalization – 3NF (example)

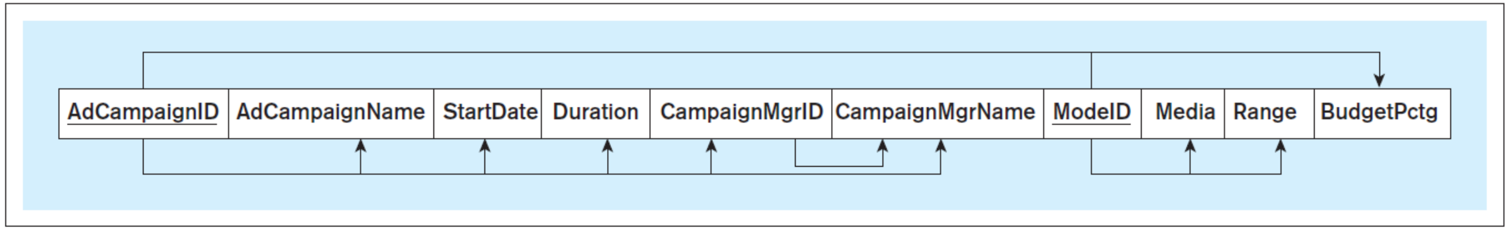
Before normalization: when update, need to change a lot, prone to **errors**

AD CAMPAIGN MIX

AdCampaignID	AdCampaignName	StartDate	Duration	Campaign MgrID	Campaign MgrName	ModelD	Media	Range	BudgetPctg
111	SummerFun13	6.6.2013	12 days	CM100	Roberta	1	TV	Local	50%
111	SummerFun13	6.6.2013	12 days	CM100	Roberta	2	TV	National	50%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	1	TV	Local	60%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	3	Radio	Local	30%
222	SummerZing13	6.8.2013	30 days	CM101	Sue	5	Print	Local	10%
333	FallBall13	6.9.2013	12 days	CM102	John	3	Radio	Local	80%
333	FallBall13	6.9.2013	12 days	CM102	John	4	Radio	National	20%
444	AutmnStyle13	6.9.2013	5 days	CM103	Nancy	6	Print	National	100%
555	AutmnColors13	6.9.2013	3 days	CM100	Roberta	3	Radio	Local	100%

DEMO:

Change Campaign 222's name to "SummerZing15"



Normalization – 3NF (example)

After normalization: easier to maintain and less error-prone

AD CAMPAIGN

AdCampaignID	AdCampaignName	StartDate	Duration	CampaignMgrID
111	SummerFun13	6.6.2013	12 days	CM100
222	SummerZing13	6.8.2013	30 days	CM101
333	FallBall13	6.9.2013	12 days	CM102
444	AutmnStyle13	6.9.2013	5 days	CM103
555	AutmnColors13	6.9.2013	3 days	CM100

CAMPAIGN MANAGER

CampaignMgrID	CampaignMgrName
CM100	Roberta
CM101	Sue
CM102	John
CM103	Nancy

MODE

ModeID	Media	Range
1	TV	Local
2	TV	National
3	Radio	Local
4	Radio	National
5	Print	Local
6	Print	National

AD CAMPAIGN MIX

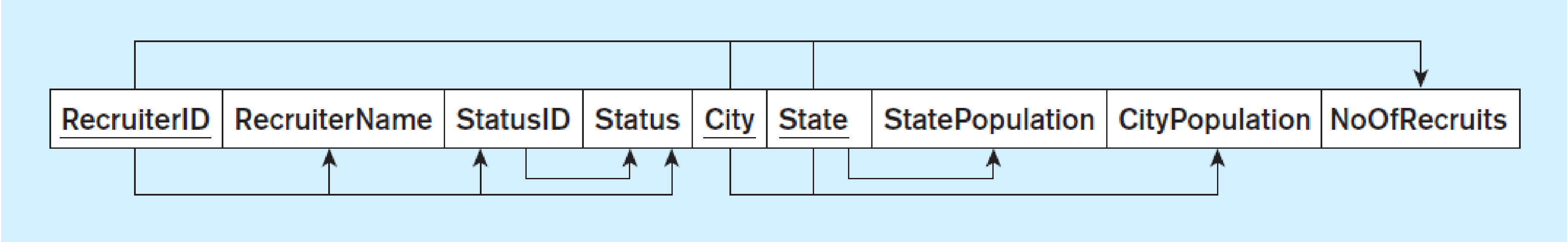
AdCampaignID	ModeID	BudgetPctg
111	1	50%
111	2	50%
222	1	60%
222	3	30%
222	5	10%
333	3	80%
333	4	20%
444	6	100%
555	3	100%

DEMO:

Change Campaign 222's name to "SummerZing15"

Normalization – 3NF (task)

Task 4 : Normalize the following table to 1NF, 2NF, and 3NF (15min)

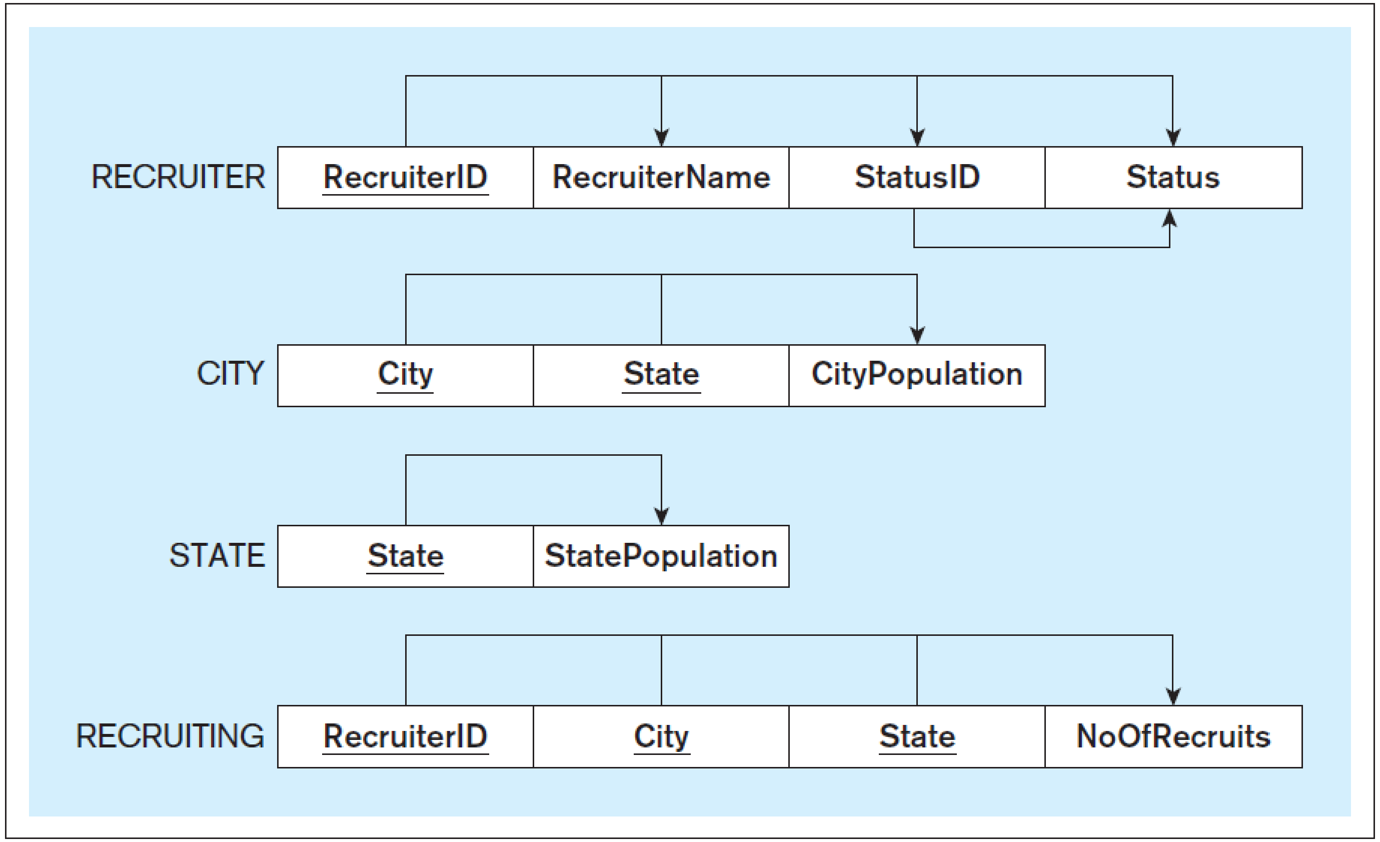


RECRUITING

RecruiterID	RecruiterName	StatusID	Status	City	State	StatePopulation	CityPopulation	NoOfRecruits
R1	Katy	IF	Internal Full Time	Portland	ME	1,350,000	70,000	11
R1	Katy	IF	Internal Full Time	Grand Rapids	MI	9,900,000	190,000	20
R2	Abra	IP	Internal Part Time	Rockford	IL	12,900,000	340,000	17
R3	Jana	CN	Contractor	Spokane	WA	6,800,000	210,000	8
R3	Jana	CN	Contractor	Portland	OR	3,900,000	600,000	30
R3	Jana	CN	Contractor	Eugene	OR	3,900,000	360,000	20
R4	Maria	IF	Internal Full Time	Rockford	IL	12,900,000	340,000	14
R4	Maria	IF	Internal Full Time	Grand Rapids	MN	5,400,000	11,000	9
R5	Dan	CN	Contractor	Grand Rapids	MI	9,900,000	190,000	33

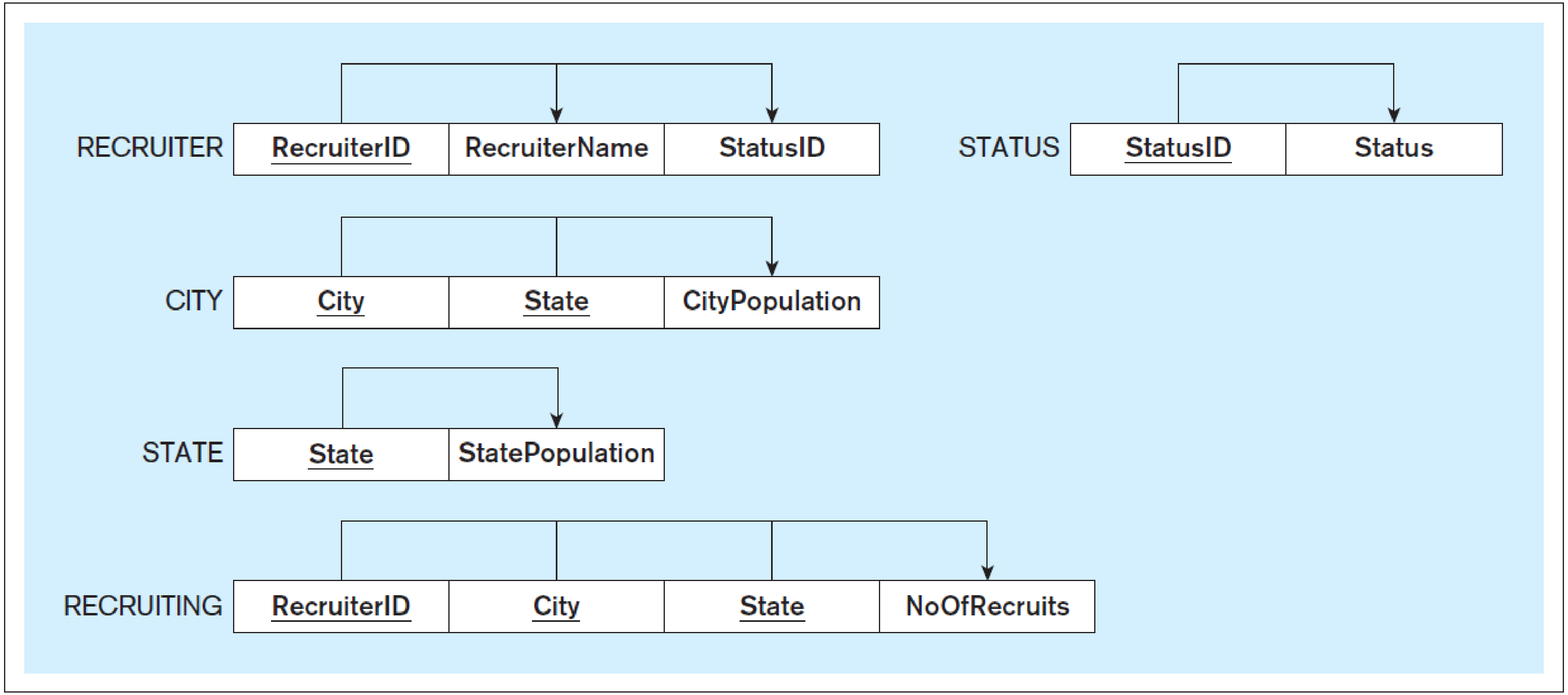
Normalization – 3NF (task solution)

2NF



Normalization – 3NF (task solution)

3NF



Normalization – 3NF (task solution)

3NF completed

RECRUITER

RecruiterID	RecruiterName	StatusID
R1	Katy	IF
R2	Abra	IP
R3	Jana	CN
R4	Maria	IF
R5	Dan	CN

STATUS

StatusID	Status
CN	Contractor
IF	Internal Full Time
IP	Internal Part Time

CITY

City	State	CityPopulation
Portland	ME	70,000
Grand Rapids	MI	190,000
Rockford	IL	340,000
Spokane	WA	210,000
Portland	OR	600,000
Eugene	OR	360,000
Grand Rapids	MN	11,000

STATE

State	StatePopulation
ME	1,350,000
MI	9,900,000
IL	12,900,000
WA	6,800,000
OR	3,900,000
MN	5,400,000

RECRUITING

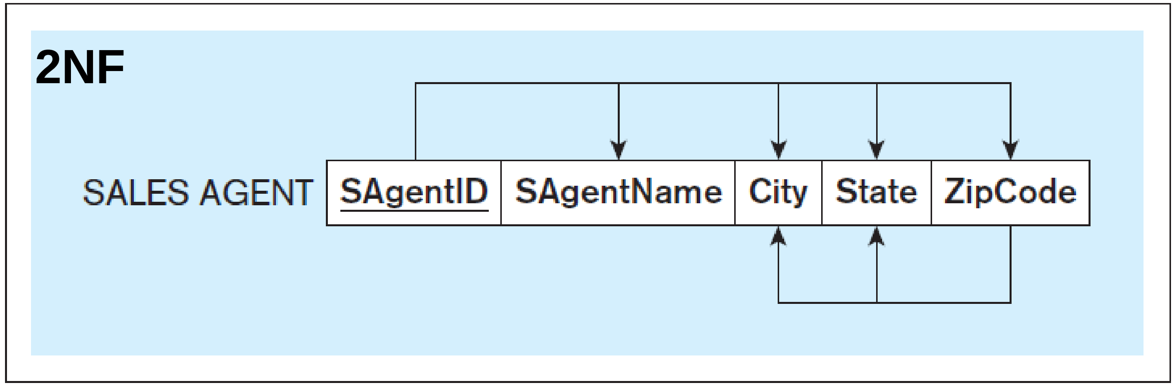
RecruiterID	City	State	NoOfRecruits
R1	Portland	ME	11
R1	Grand Rapids	MI	20
R2	Rockford	IL	17
R3	Spokane	WA	8
R3	Portland	OR	30
R3	Eugene	OR	20
R4	Rockford	IL	14
R4	Grand Rapids	MN	9
R5	Grand Rapids	MI	33

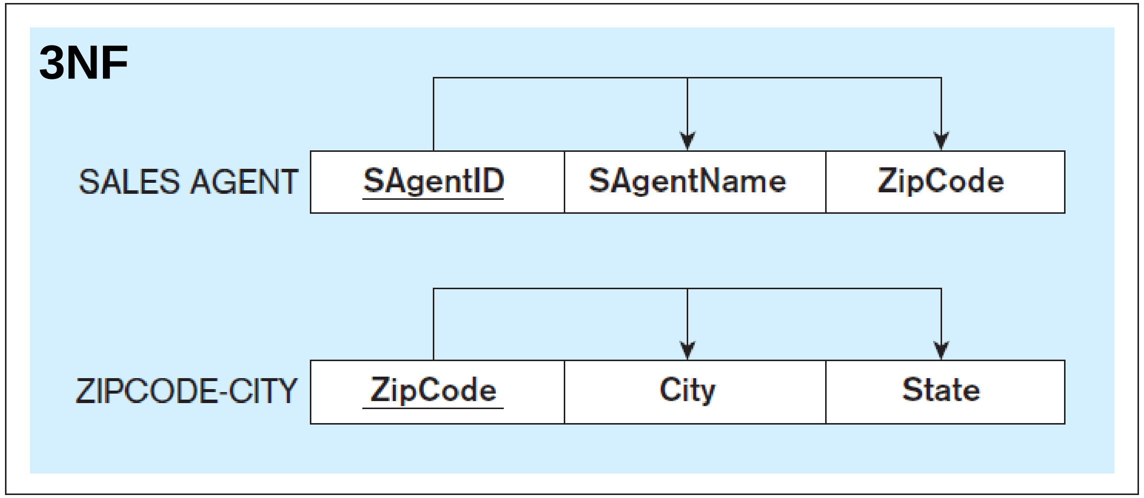
Normalization – Yes or no?

Task 5 : We have this SALES AGENT table, and we normalized it to 2NF. Should we go on normalizing it to 3NF? Yes or no? Give an explanation?

SALES AGENT

SAgentID	SAgentName	City	State	ZipCode
SA1	Rose	Glen Ellyn	IL	60137
SA2	Sidney	Chicago	IL	60611
SA3	James	Chicago	IL	60610
SA4	Violet	Wheaton	IL	60187
SA5	Nicole	Kenosha	WI	53140
SA6	Justin	Milwaukee	WI	53201





Normalization – Yes or no?

Should we go on normalizing it to 3NF? Yes or no? Give an explanation?

We may need decide by evaluating the tradeoffs
- The **pros**: reduce some redundancy
- The **cons**: increasing the complexity of the relational schema by adding another table and referential integrity constraint to it.

e.g. If sales agents within the same zip code are rare, the benefits of normalization to 3NF would be marginal or negligible. (No need)
e.g. If the company has a limit of two sales agents per state, then the possible redundancy in the original relation is even smaller. (No need)

Normalization Exceptions

- **Normalization Exceptions**
  - In general, database relations are normalized to 3NF in order to eliminate unnecessary data redundancy and avoid update anomalies
  - However, normalization to 3NF should be done carefully and pragmatically, which may in some cases call for deliberately not normalizing certain relations to 3NF

Normalization Exceptions

- **Denormalization -** reversing the effect of normalization by **joining** normalized relations into a relation that is not normalized, in order to improve query performance
  - The data that resided in fewer relations prior to normalization is spread out across more relations after normalization
  - This has an effect on the performance of data retrievals
  - Denormalization can be used in dealing with the normalization vs. performance issue
- Denormalization is not a default process that is to be undertaken in all circumstances
  - Instead, denormalization should be used carefully, after analyzing its costs and benefits

Indexing

- **INDEX** - Mechanism for **increasing the speed** of data **search** and data retrieval on relations with a large number of records
  - Most relational DBMS software tools enable definition of indexes
- Data search speed is an important performance issue for big databases
  - Imagine 11-11/12-12 online shopping in TaoBao/JingDong - millions of products, but your searching remains faster enough
- How to improve searching speed?
  - Linear searching vs. non-linear searching

Linear searching - unsorted column

- We want to look for Customer with Name of 'steve'

CUSTOMER

CustID	CustName	Zip
1000	Zach	60111
1001	Ana	60333
1002	Matt	60222
1003	Lara	60555
1004	Pam	60444
1005	Sally	60555
1006	Bob	60333
1007	Adam	60555
1008	Steve	60222
1009	Pam	60333
1010	Ema	60111
1011	Peter	60666
1012	Fiona	60444

Task 6: What is the complexity of linear search (worst-case)?

Linear (Sequential) search

O(n)

CUSTOMER

CustID	CustName	Zip	
1000	Zach	60111	→ Step 1 (not Steve)
1001	Ana	60333	→ Step 2 (not Steve)
1002	Matt	60222	→ Step 3 (not Steve)
1003	Lara	60555	→ Step 4 (not Steve)
1004	Pam	60444	→ Step 5 (not Steve)
1005	Sally	60555	→ Step 6 (not Steve)
1006	Bob	60333	→ Step 7 (not Steve)
1007	Adam	60555	→ Step 8 (not Steve)
1008	Steve	60222	→ Step 9 Customer Steve found
1009	Pam	60333	
1010	Ema	60111	
1011	Peter	60666	
1012	Fiona	60444	

Can be very very slow for huge data

Non-linear (Binary) searching - sorted column

Look for customer ID = 1008

Task 7 What is the complexity of binary search?

O(log(n))

CUSTOMER

CustID	CustName	Zip
1000	Zach	60111
1001	Ana	60333
1002	Matt	60222
1003	Lara	60555
1004	Pam	60444
1005	Sally	60555
1006	Bob	60333
1007	Adam	60555
1008	Steve	60222
1009	Pam	60333
1010	Ema	60111
1011	Peter	60666
1012	Fiona	60444

Search on sorted column is faster than on unsorted column!
Because during binary partitions:
n -> n/2 -> n/4 -> n/8 -> … -> 1
Let partition number be k,
then n/(2^k) ~ 1. Get k ~ log(n).

Step 1 Eliminate records from here, above (since CustID value is lower than 1008)

Step 3 Customer 1008 found

Step 2 Eliminate records from here, below (since CustID value is higher than 1008)

Indexing - concept

Indexing creates an additional index table (sorted + pointer), and allows binary search on it and then points back to the original column (unsorted)

Additional index table

CUSTNAME_INDEX

CustName	CustID
Adam	1007
Ana	1001
Bob	1006
Ema	1010
Fiona	1012
Lara	1003
Matt	1002
Pam	1004
Pam	1009
Peter	1011
Sally	1005
Steve	1008
Zach	1000

sorted pointer

CUSTOMER

CustID	CustName	Zip
1000	Zach	60111
1001	Ana	60333
1002	Matt	60222
1003	Lara	60555
1004	Pam	60444
1005	Sally	60555
1006	Bob	60333
1007	Adam	60555
1008	Steve	60222
1009	Pam	60333
1010	Ema	60111
1011	Peter	60666
1012	Fiona	60444

unsorted

Indexing - example

Conceptual simplified illustration of the principles on which an index is based **Increased search speed using the index – example (custName='Steve')**

CUSTNAME_INDEX

CustName	CustID
Adam	1007
Ana	1001
Bob	1006
Ema	1010
Fiona	1012
Lara	1003
Matt	1002
Pam	1004
Pam	1009
Peter	1011
Sally	1005
Steve	1008
Zach	1000

Step 1 Eliminate records from here, above (since Matt occurs alphabetically before Steve)

Step 2 Eliminate records from here, above (since Peter occurs alphabetically before Steve)

Step 3 (Name Steve found in index record)

Step 4 Following the pointer from the index record, the searched-for record (customer Steve) is found in the relation CUSTOMER

CUSTOMER

CustID	CustName	Zip
1000	Zach	60111
1001	Ana	60333
1002	Matt	60222
1003	Lara	60555
1004	Pam	60444
1005	Sally	60555
1006	Bob	60333
1007	Adam	60555
1008	Steve	60222
1009	Pam	60333
1010	Ema	60111
1011	Peter	60666
1012	Fiona	60444

Indexing - task

Task 8 :

1. In the following table, we want to find out the record of custName="Pam". If we do the linear search, how many steps are there? Hint: you need find all matching records.
2. If we create the index, how many steps are there? Hint: same as above.

CUSTOMER

CustID	CustName	Zip
1000	Zach	60111
1001	Ana	60333
1002	Matt	60222
1003	Lara	60555
1004	Pam	60444
1005	Sally	60555
1006	Bob	60333
1007	Adam	60555
1008	Steve	60222
1009	Pam	60333
1010	Ema	60111
1011	Peter	60666
1012	Fiona	60444

Indexing - task solution

Task 8 :

1. In the following table, we want to find out the record(s) of custName="Pam". If we do the linear search, how many steps are there? Hint: you need find all matching records.
2. If we create the index, how many steps are there? Hint: same as above.

13, 6

CUSTOMER

CustID	CustName	Zip	
1000	Zach	60111	Step 1 (not Pam, counter = 0)
1001	Ana	60333	Step 2 (not Pam, counter = 0)
1002	Matt	60222	Step 3 (not Pam, counter = 0)
1003	Lara	60555	Step 4 (not Pam, counter = 0)
1004	Pam	60444	Step 5 (Pam, counter = 1)
1005	Sally	60555	Step 6 (not Pam, counter = 1)
1006	Bob	60333	Step 7 (not Pam, counter = 1)
1007	Adam	60555	Step 8 (not Pam, counter = 1)
1008	Steve	60222	Step 9 (not Pam, counter = 1)
1009	Pam	60333	Step 10 (Pam, counter = 2)
1010	Ema	60111	Step 11 (not Pam, counter = 2)
1011	Peter	60666	Step 12 (not Pam, counter = 2)
1012	Fiona	60444	Step 13 (not Pam, final count = 2)

CUSTNAME_INDEX

CustName	CustID
Adam	1007
Ana	1001
Bob	1006
Ema	1010
Fiona	1012
Lara	1003
Matt	1002
Pam	1004
Pam	1010
Peter	1011
Sally	1005
Steve	1008
Zach	1000

Step 1 Eliminate records from here, above (since Matt occurs alphabetically before Pam)

Step 6 Check the name of the record above (not Pam, final count = 2)

Step 3 An instance of the name Pam found (counter = 1)

Step 4 Check the name in the record below (not Pam, counter = 1)

Step 5 Check the name in the record above (Pam, counter = 2)

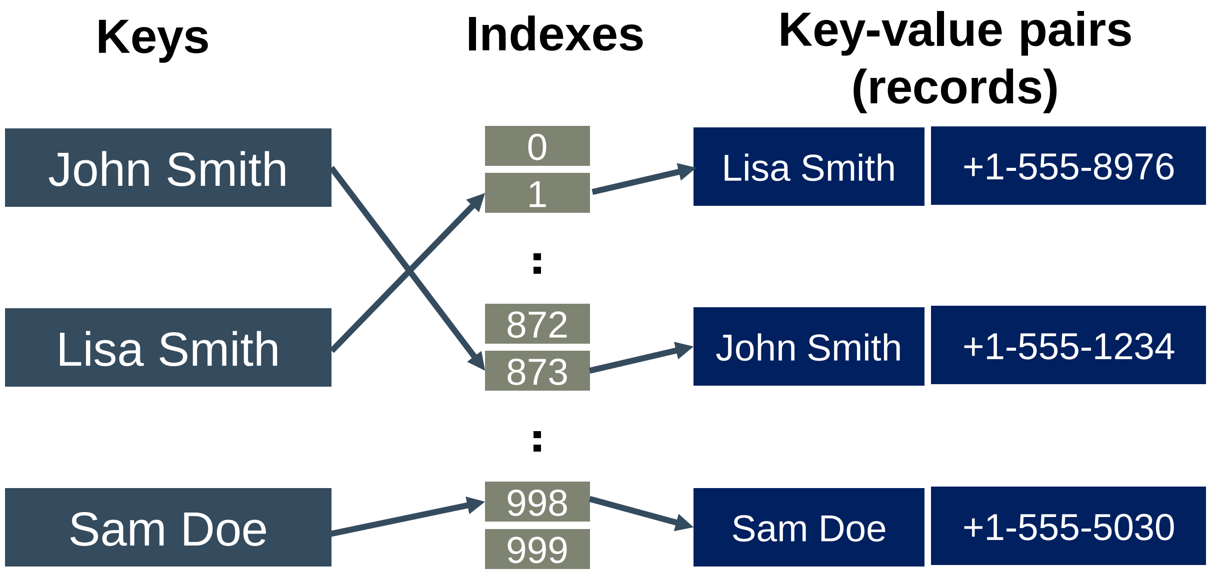
Step 2 Eliminate records from here, below (since Peter occurs alphabetically after Pam)

Indexing - other types

- The preceding examples provided simplified conceptual illustration of the principles on which an index is based
- Instead of simply sorting on the indexed column and applying binary search, different contemporary DBMS tools implement indexes using different logical and technical approaches, such as:
  - Clustering indexes
  - Hash indexes
  - B-trees, etc.
- Each of the available approaches has the same goal – increase the speed of search and retrieval on the columns that are being indexed

Indexing – Hash Index

- Hash index structuring system is very useful when files are **not sequential**.
- System can be described as data storage spaces are divided into compartments called **buckets**. Data then distributed to these buckets depending on the **key value** calculated by **hash function**. So very fast.

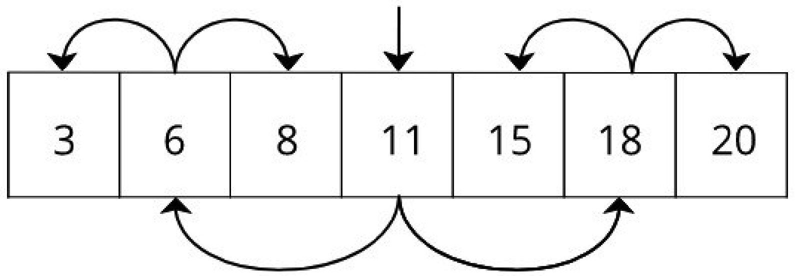


Task 9: What is the complexity of hash index search?

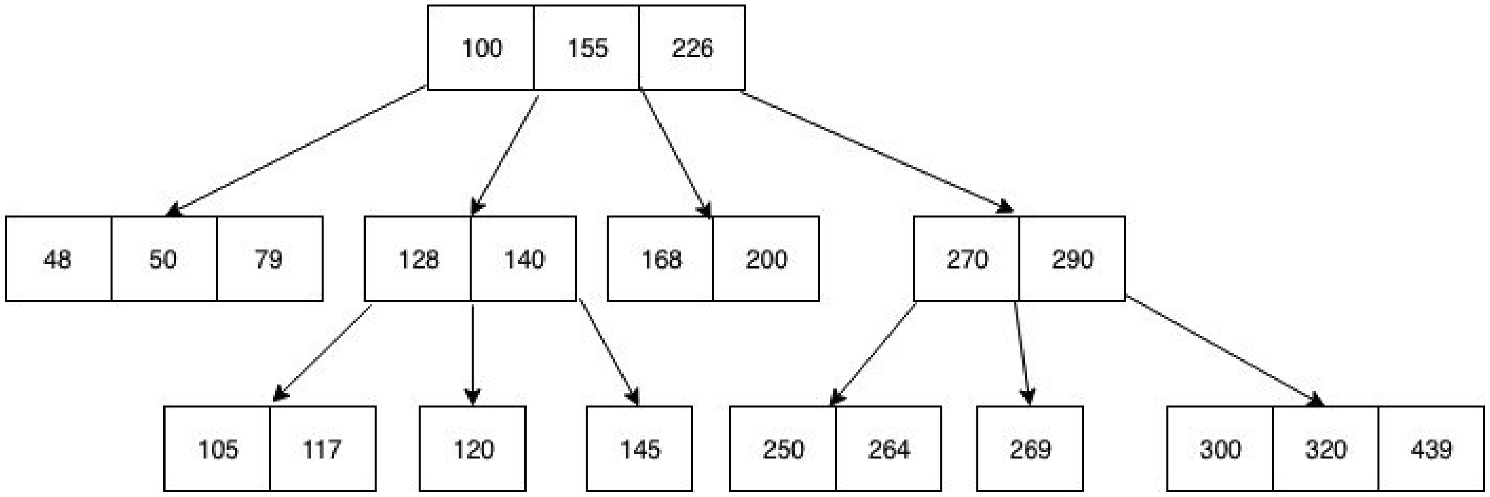
O(1)

Indexing – B-Tree indexing

- B-tree generalizes the binary search tree to allow more than 2 branches in the nodes



- The index tree is stored separately from the data
- The lower-level leaves contain the pointers to the actual data rows

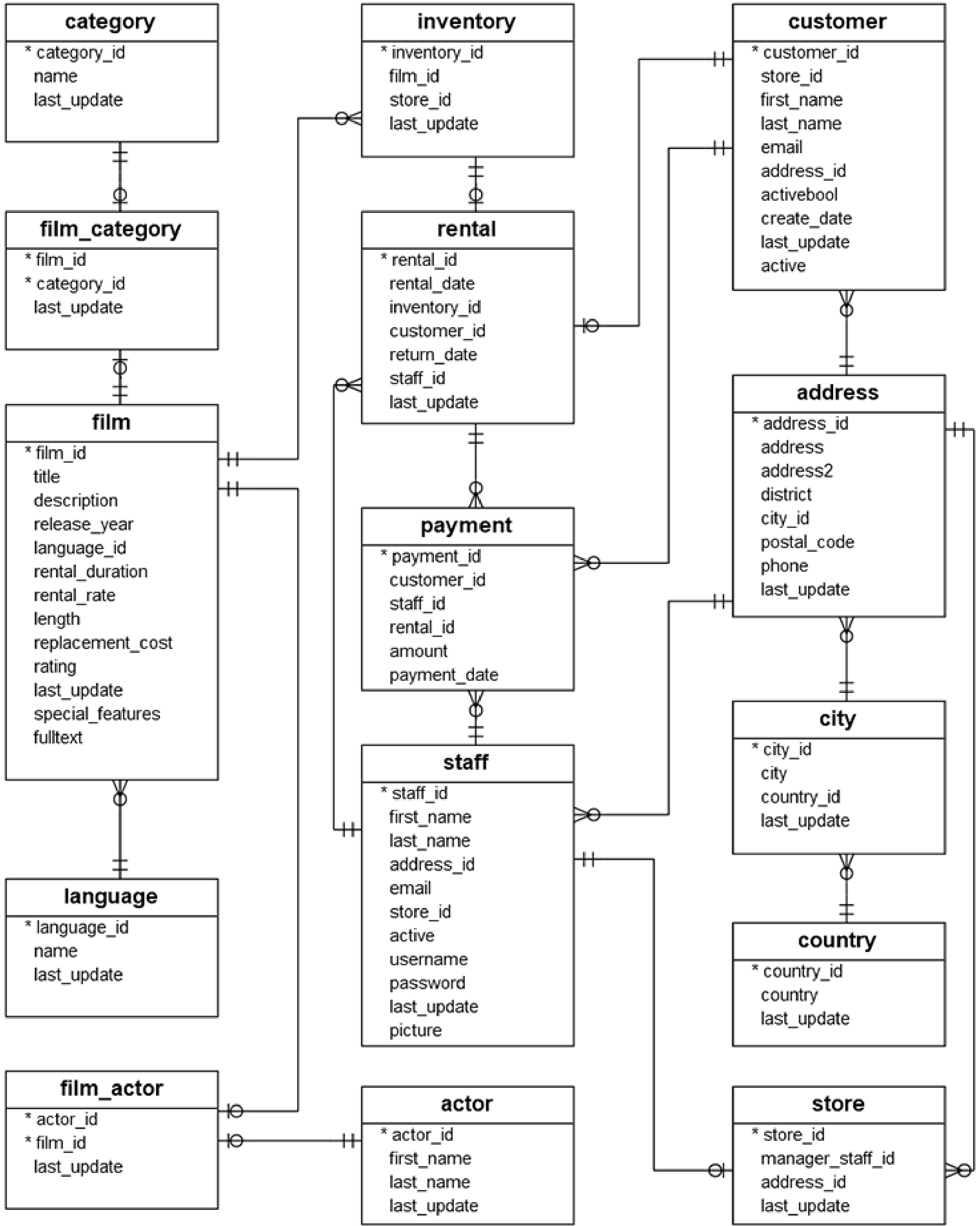


Indexing – pros and cons

	Pros	Cons
Hash O(1)	• Very efficient at equality queries (column==value). • Take constant time, independent of the number of rows in a table	• Not efficient at inequality queries (eg. <, <=, >, <=, between, in, not in, like). • Require more space than range indexes.
B-Tree O(log(n))	• Range indexes are efficient at processing inequality queries (eg. <, <=, >, <=, between, in, not in, like)	• Not fast for equality queries.

Indexing – MySQL syntax

- CREATE INDEX
  - Syntax:
    CREATE INDEX index_name ON tbl_name (col_name);
  - This statement creates an index on the column *col_name*
- DROP INDEX
  - Syntax:
    DROP INDEX index_name ON tbl_name;
  - This statement drops the index, and the index is no longer used



Let’s get back to our DVD rental db example

Still remember how to launch MySQL and choose the database?

Indexing in MySQL

A simple version of CREATE INDEX statement is as follows:

```
CREATE INDEX  index_name
    [ USING {BTREE | HASH} ]
    ON table_name (column_name [ASC | DESC],    ...);
```

DEMO: ***What will happen?** See if the message tells the indexing key (idx_address_phone) is used in searching*

```
EXPLAIN
SELECT
    *
FROM
    address
WHERE
    phone = '223664661973';
```

```
CREATE INDEX idx_address_phone
ON address (phone);

EXPLAIN
SELECT
    *
FROM
    address
WHERE
    phone = '223664661973';
```

Indexing – last word

- Indexes are effective tools to enhance database performance. Indexes help the database server find specific rows much faster than it could do without indexes.

- However, indexes add write and storage overheads to the database system

- Therefore, using them appropriately are very important.

Summary

- Understand what is database normalization?
  - Why? And how?
  - What is 1NF, 2NF, 3NF?
  - How to normalize to 1NF, 2NF, 3NF?
- What is database index? Why we need indexing?
  - Linear search, Binary search
  - Simple index, Hash index, etc.
  - MySQL index syntax
- Both normalization & indexing are double-edged swords