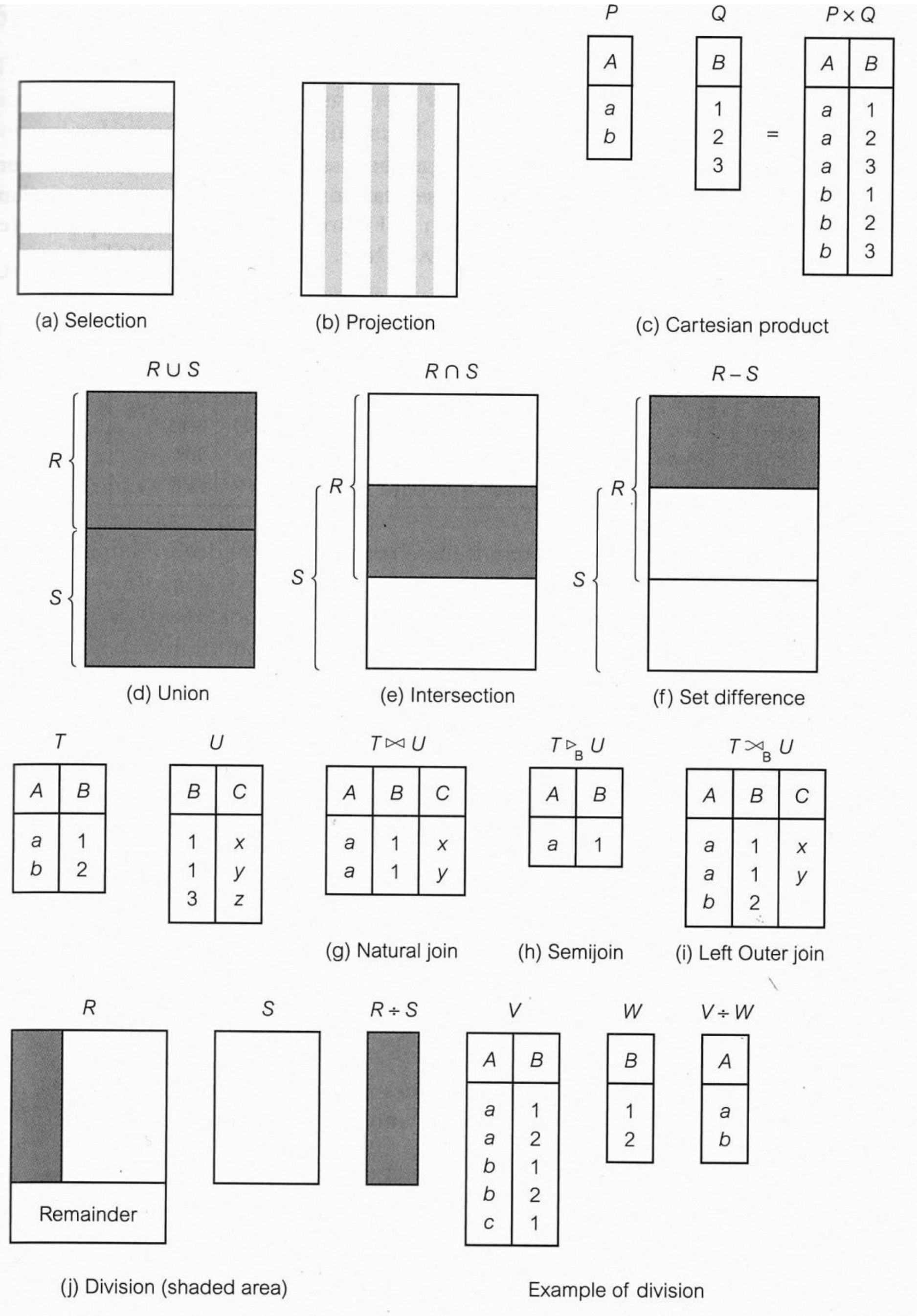
Relational Algebra, Chapter 5

Imperative Versus Declarative

Imperative	Declarative
Describes how to do something	Describes what is wanted, but not how to get it
Examples: Java, C, C#, FORTRAN, PHP, JavaScript, assembly, MATLAB	Prolog, Scheme, dBASE, Query-By-Example (Access)
Relational algebra	SQL

Relational Algebra



(a) Selection

(b) Projection

P

A
a
b

Q

B
1
2
3

$P \times Q$ =

A	B
a	1
a	2
a	3
b	1
b	2
b	3

(c) Cartesian product

(d) Union

(e) Intersection

(f) Set difference

T

A	B
a	1
b	2

U

B	C
1	x
1	y
3	z

$T \bowtie U$

A	B	C
a	1	x
a	1	y

(g) Natural join

$T \triangleright_B U$

A	B
a	1

(h) Semijoin

$T ⟕_B U$

A	B	C
a	1	x
a	1	y
b	2	

(i) Left Outer join

(j) Division (shaded area)

V

A	B
a	1
a	2
b	1
b	2
c	1

W

B
1
2

$V \div W$

A
a
b

Example of division

Figure 5.1 Illustration showing the function of the relational algebra operations.

Relational Algebra

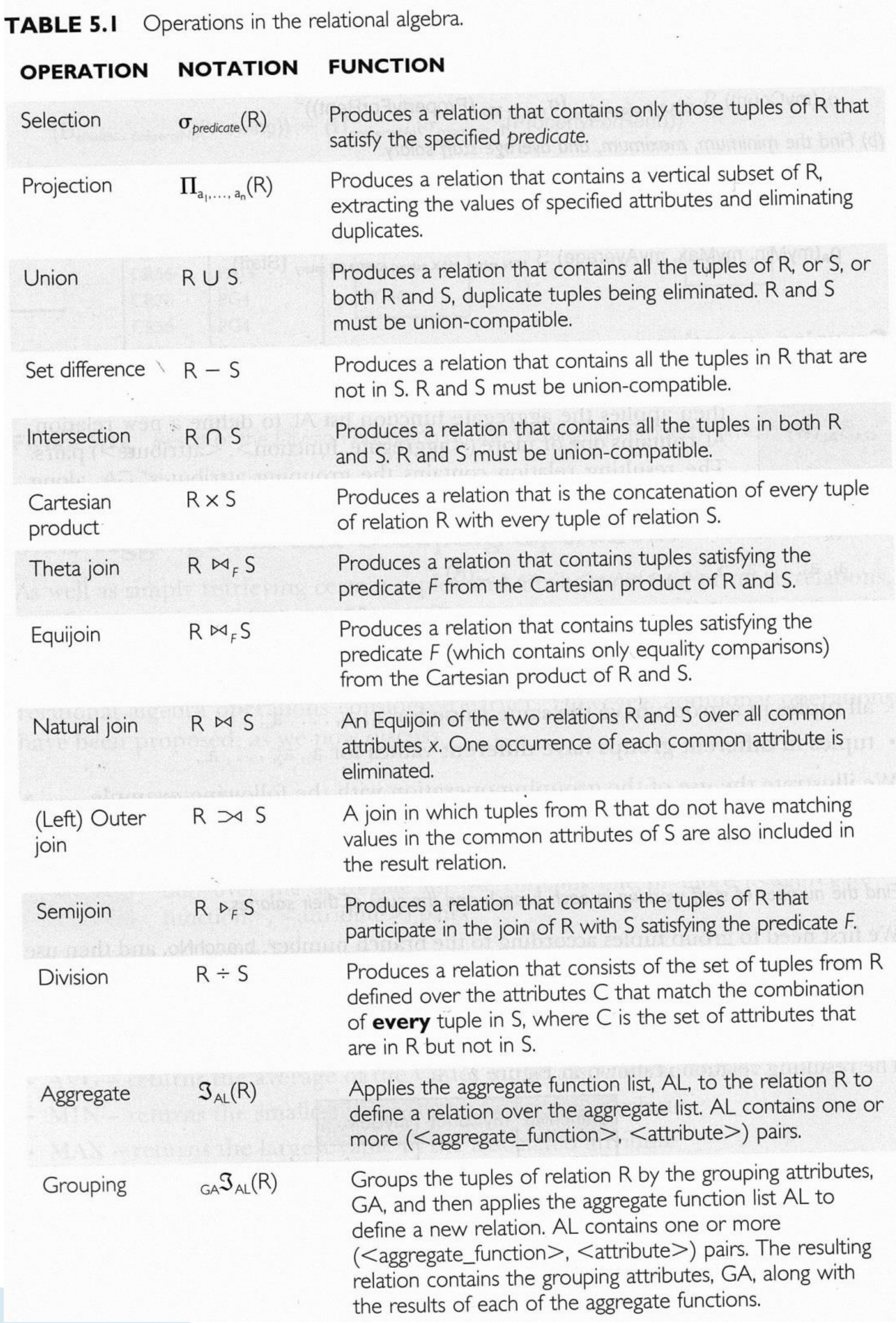
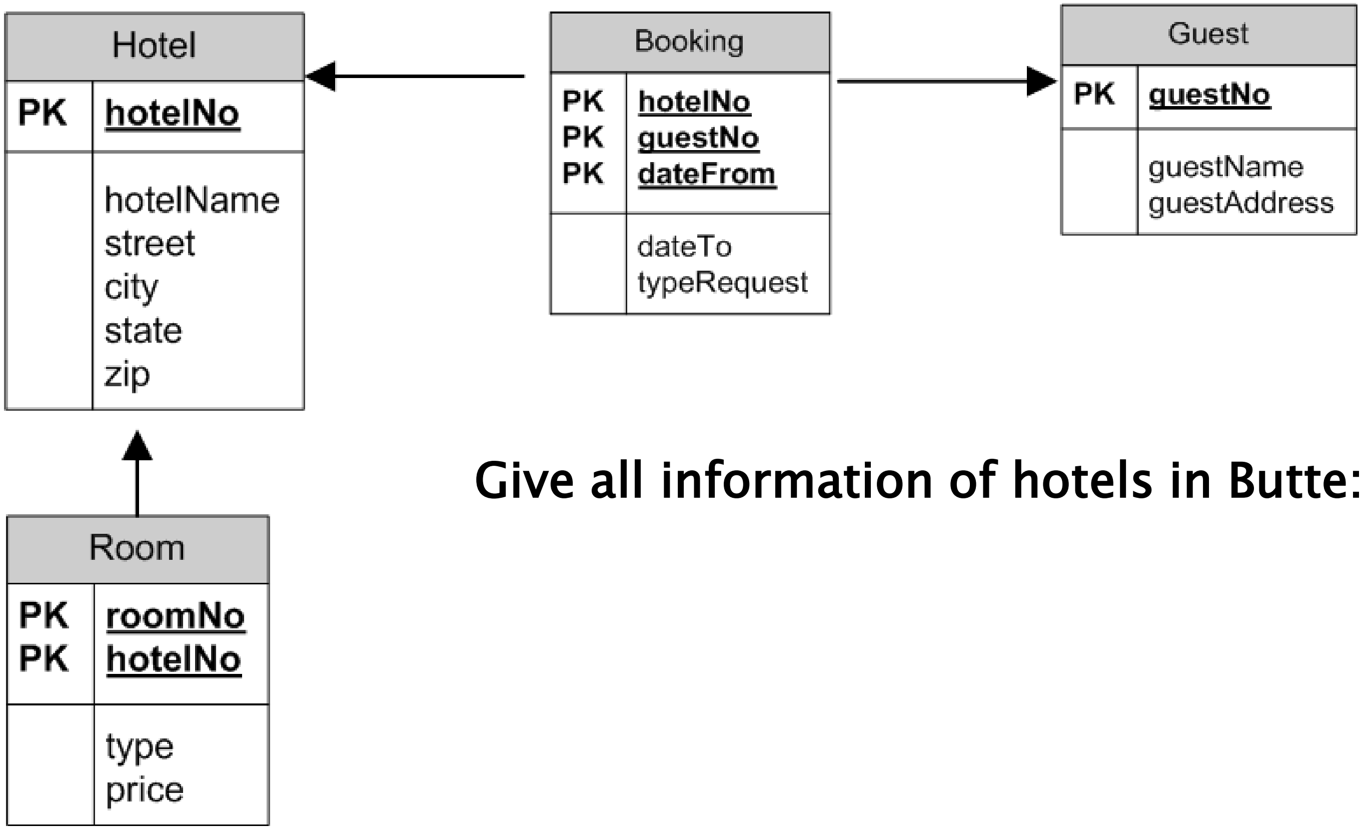


TABLE 5.1 Operations in the relational algebra.

OPERATION	NOTATION	FUNCTION
Selection	$\sigma_{predicate}(R)$	Produces a relation that contains only those tuples of R that satisfy the specified *predicate*.
Projection	$\Pi_{a_1,\ldots,a_n}(R)$	Produces a relation that contains a vertical subset of R, extracting the values of specified attributes and eliminating duplicates.
Union	$R \cup S$	Produces a relation that contains all the tuples of R, or S, or both R and S, duplicate tuples being eliminated. R and S must be union-compatible.
Set difference	$R - S$	Produces a relation that contains all the tuples in R that are not in S. R and S must be union-compatible.
Intersection	$R \cap S$	Produces a relation that contains all the tuples in both R and S. R and S must be union-compatible.
Cartesian product	$R \times S$	Produces a relation that is the concatenation of every tuple of relation R with every tuple of relation S.
Theta join	$R \bowtie_F S$	Produces a relation that contains tuples satisfying the predicate *F* from the Cartesian product of R and S.
Equijoin	$R \bowtie_F S$	Produces a relation that contains tuples satisfying the predicate *F* (which contains only equality comparisons) from the Cartesian product of R and S.
Natural join	$R \bowtie S$	An Equijoin of the two relations R and S over all common attributes *x*. One occurrence of each common attribute is eliminated.
(Left) Outer join	$R \sqsupset\!\!\bowtie S$	A join in which tuples from R that do not have matching values in the common attributes of S are also included in the result relation.
Semijoin	$R \triangleright_F S$	Produces a relation that contains the tuples of R that participate in the join of R with S satisfying the predicate *F*.
Division	$R \div S$	Produces a relation that consists of the set of tuples from R defined over the attributes C that match the combination of **every** tuple in S, where C is the set of attributes that are in R but not in S.
Aggregate	$\Im_{AL}(R)$	Applies the aggregate function list, AL, to the relation R to define a relation over the aggregate list. AL contains one or more (<aggregate_function>, <attribute>) pairs.
Grouping	${}_{GA}\Im_{AL}(R)$	Groups the tuples of relation R by the grouping attributes, GA, and then applies the aggregate function list AL to define a new relation. AL contains one or more (<aggregate_function>, <attribute>) pairs. The resulting relation contains the grouping attributes, GA, along with the results of each of the aggregate functions.

Query 1



Hotel	
PK	hotelNo
	hotelName street city state zip

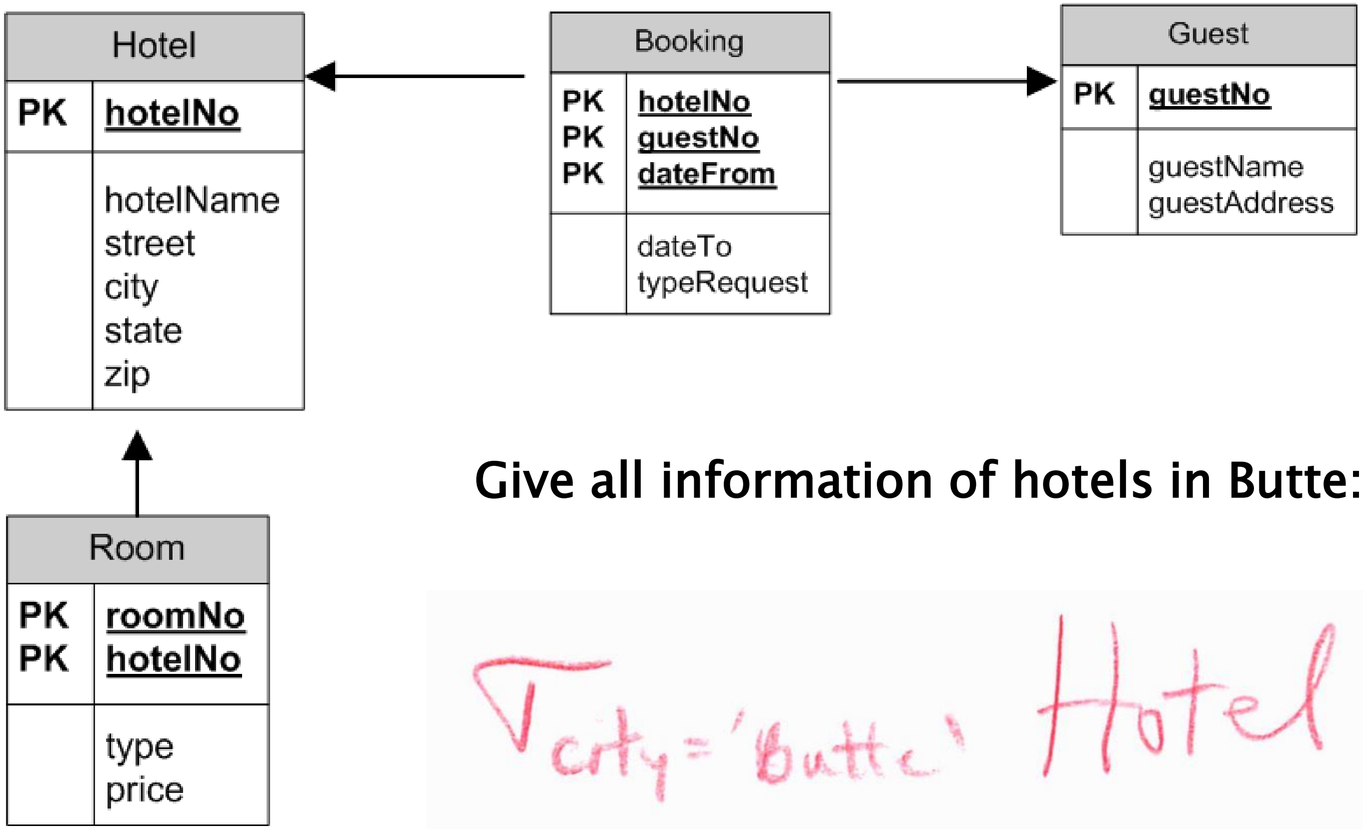
Booking	
PK PK PK	hotelNo guestNo dateFrom
	dateTo typeRequest

Guest	
PK	guestNo
	guestName guestAddress

Room	
PK PK	roomNo hotelNo
	type price

Give all information of hotels in Butte:

Query 1 – answer



Hotel	
PK	**hotelNo**
	hotelName street city state zip

Booking	
PK PK PK	**hotelNo** **guestNo** **dateFrom**
	dateTo typeRequest

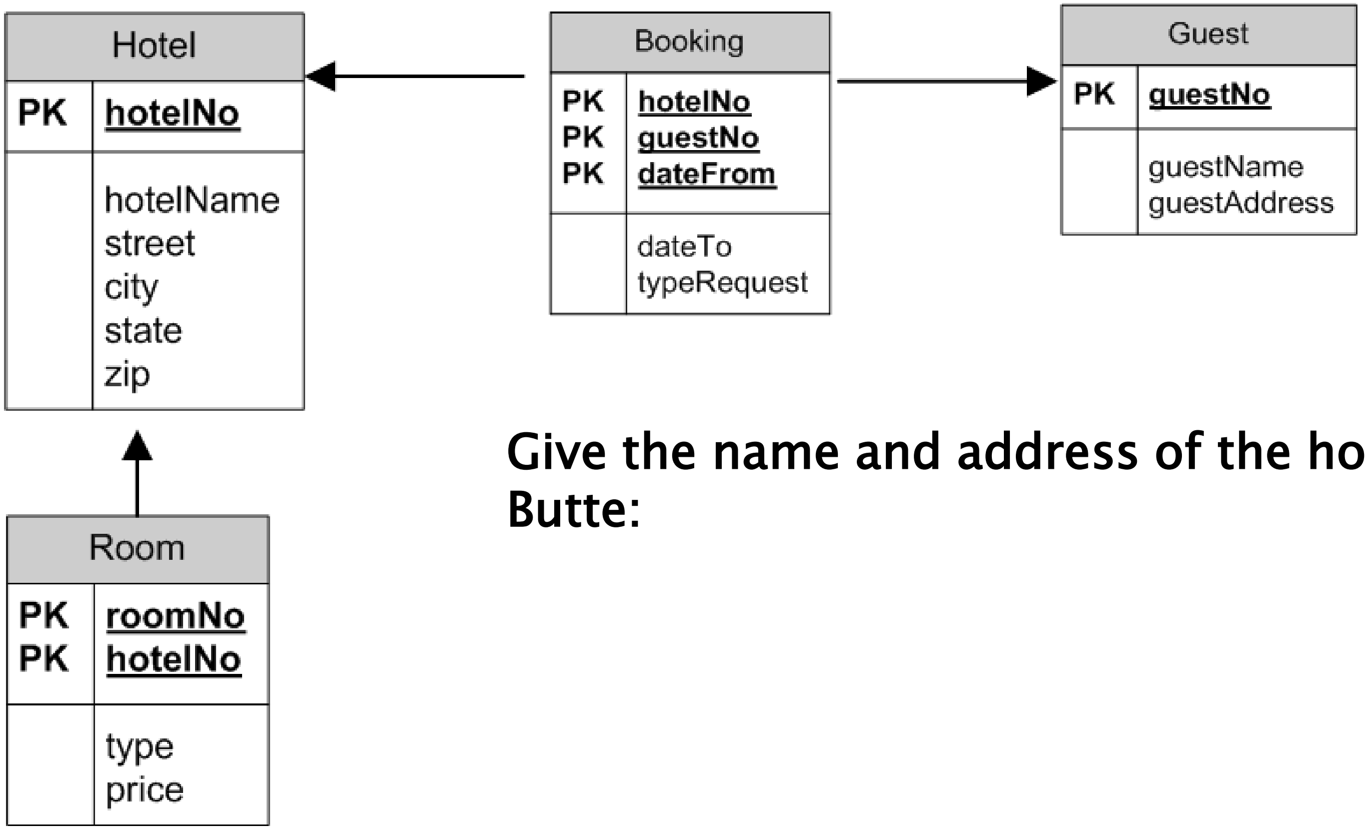
Guest	
PK	**guestNo**
	guestName guestAddress

Room	
PK PK	**roomNo** **hotelNo**
	type price

Give all information of hotels in Butte:

$$\sigma_{city='Butte'}\ Hotel$$

Query 2



Hotel

PK **hotelNo**

hotelName
street
city
state
zip

Booking

PK **hotelNo**
PK **guestNo**
PK **dateFrom**

dateTo
typeRequest

Guest

PK **guestNo**

guestName
guestAddress

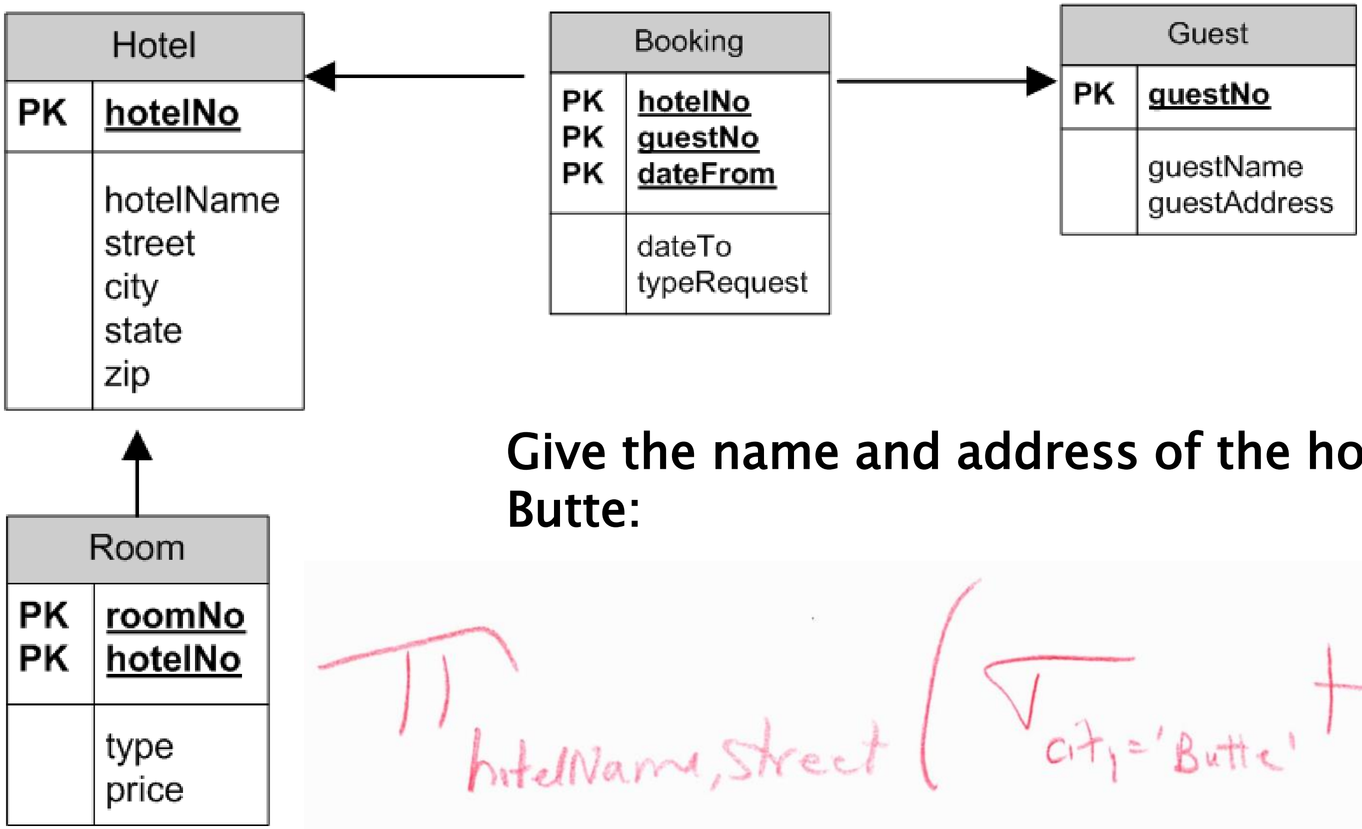
Room

PK **roomNo**
PK **hotelNo**

type
price

Give the name and address of the hotels in Butte:

Query 2 – Answer



Hotel
PK hotelNo
hotelName
street
city
state
zip

Booking
PK hotelNo
PK guestNo
PK dateFrom
dateTo
typeRequest

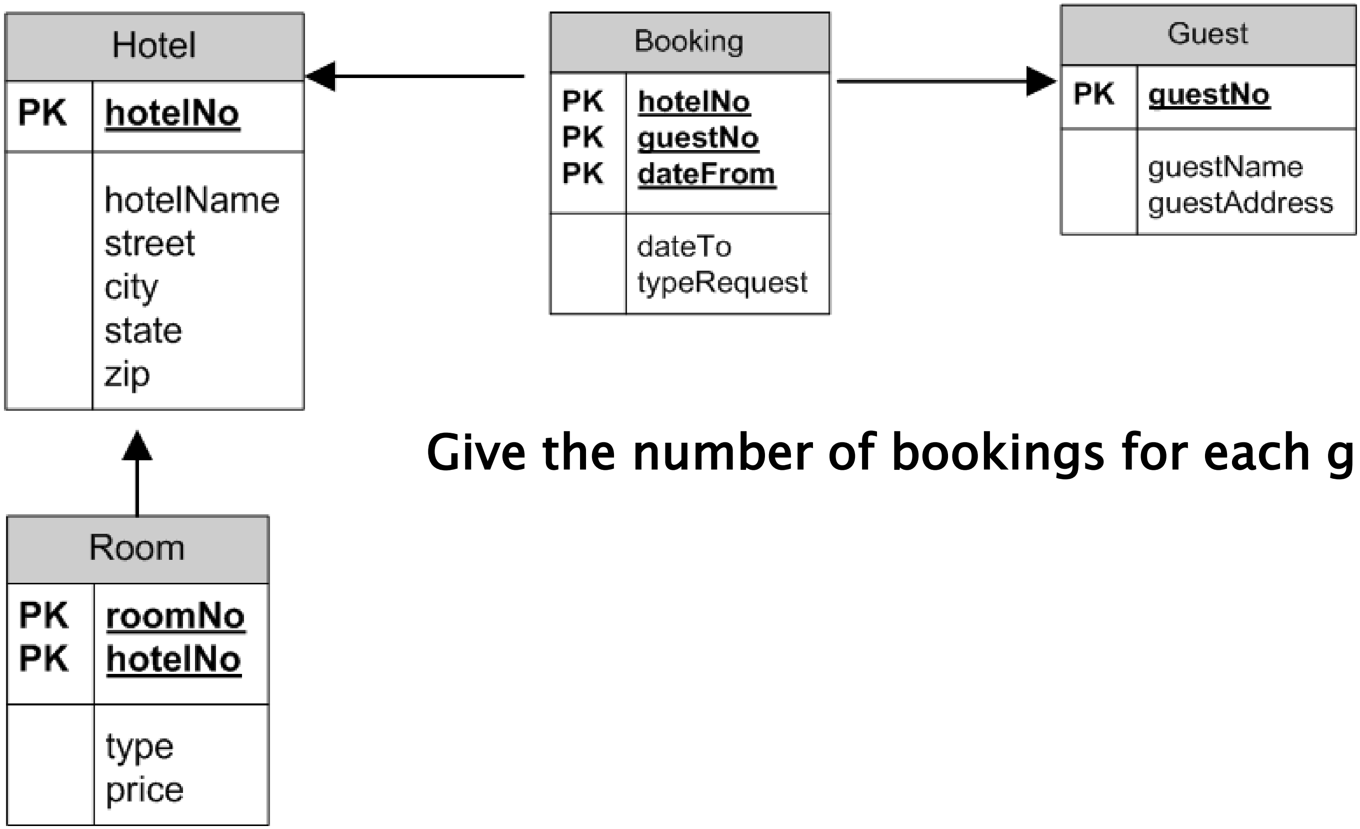
Guest
PK guestNo
guestName
guestAddress

Room
PK roomNo
PK hotelNo
type
price

Give the name and address of the hotels in Butte:

$$\pi_{hotelName, street}\left(\sigma_{city='Butte'} Hotel\right)$$

Aggregate Query



Hotel	
PK	hotelNo
	hotelName street city state zip

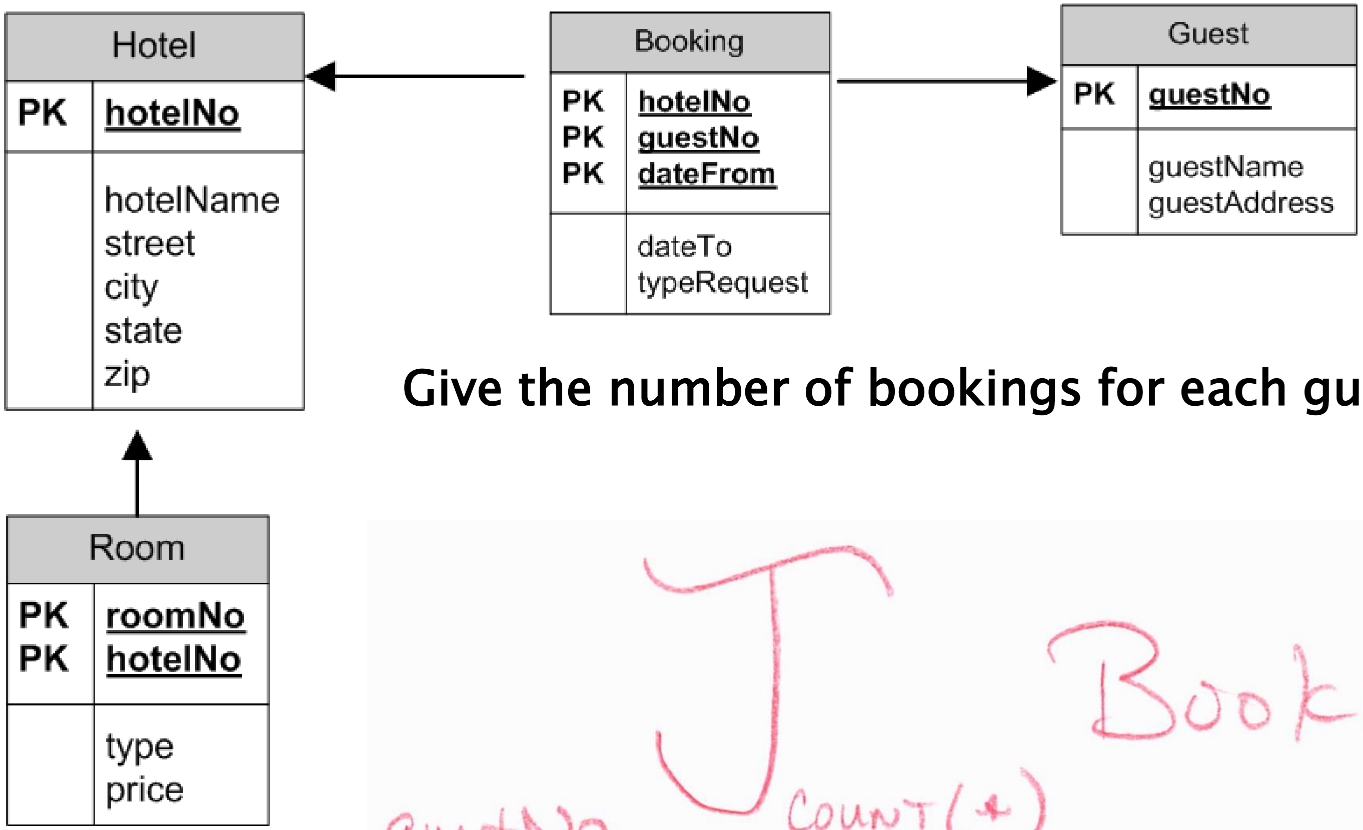
Booking	
PK PK PK	hotelNo guestNo dateFrom
	dateTo typeRequest

Guest	
PK	guestNo
	guestName guestAddress

Room	
PK PK	roomNo hotelNo
	type price

Give the number of bookings for each guest:

Aggregate Query – GROUP BY and COUNT



Hotel
PK hotelNo
hotelName
street
city
state
zip

Booking
PK hotelNo
PK guestNo
PK dateFrom
dateTo
typeRequest

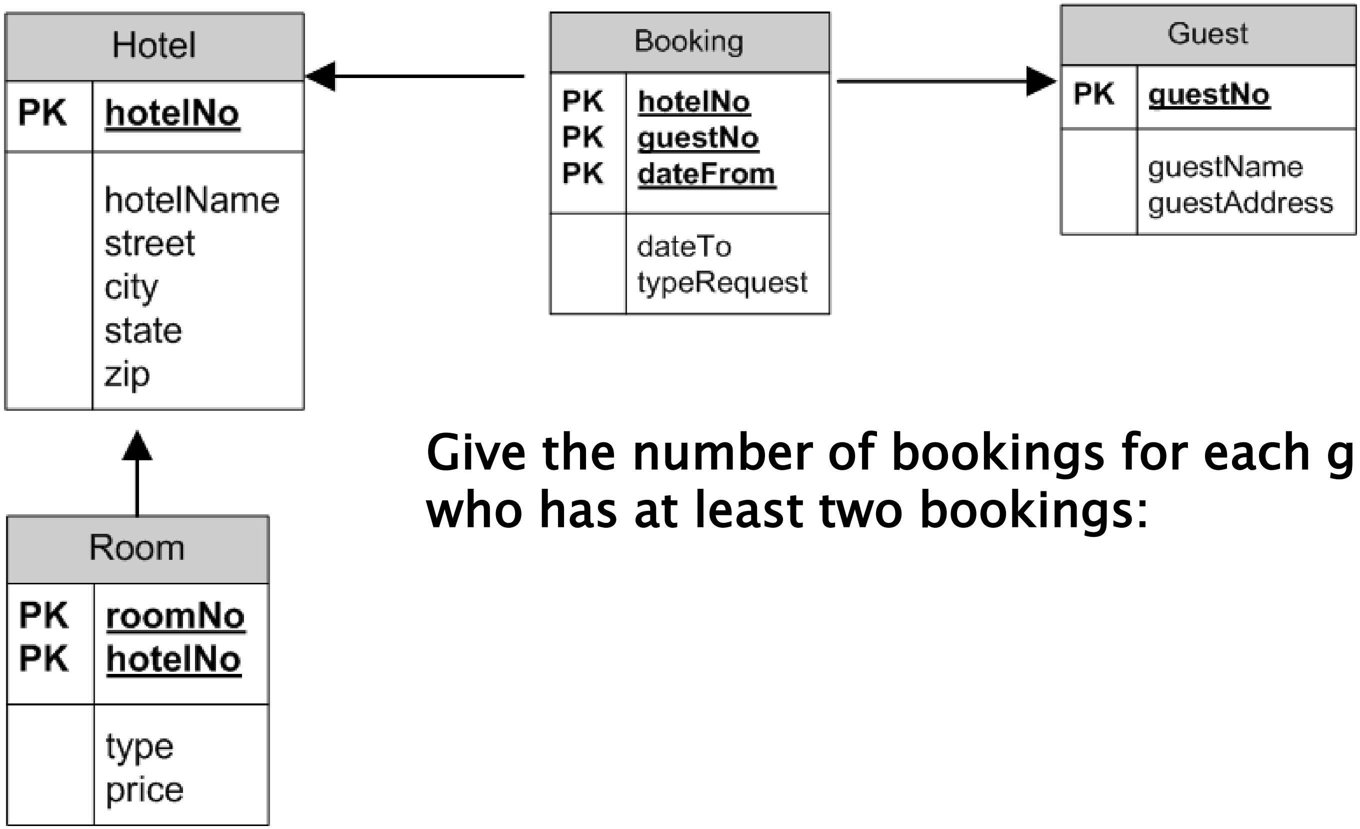
Guest
PK guestNo
guestName
guestAddress

Room
PK roomNo
PK hotelNo
type
price

Give the number of bookings for each guest:

$${}_{guestNo}\mathfrak{I}_{COUNT(*)}\ Booking$$

Aggregate Query Filtered



Hotel
PK hotelNo
hotelName
street
city
state
zip

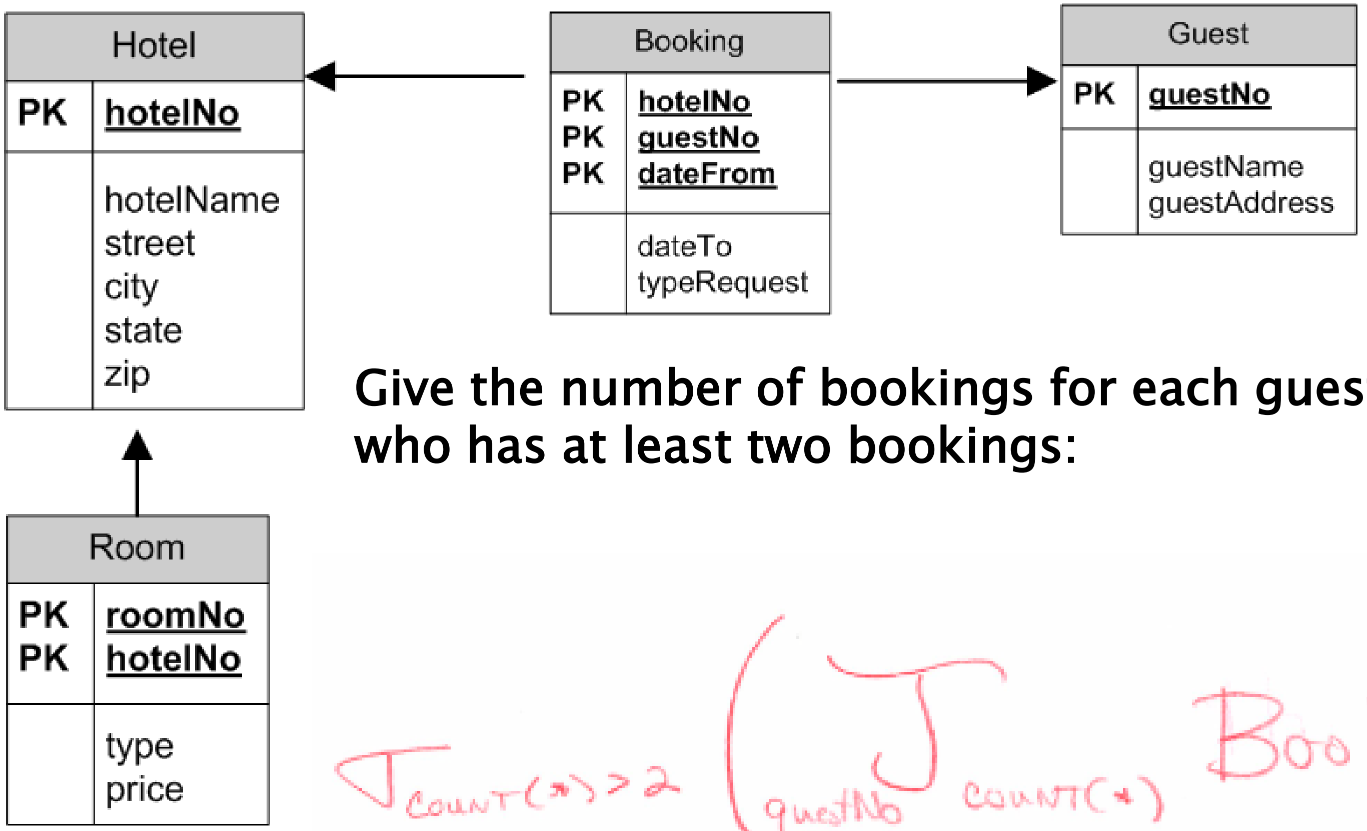
Booking
PK hotelNo
PK guestNo
PK dateFrom
dateTo
typeRequest

Guest
PK guestNo
guestName
guestAddress

Room
PK roomNo
PK hotelNo
type
price

Give the number of bookings for each guest who has at least two bookings:

Aggregate Query – GROUP BY and HAVING



Hotel

PK hotelNo

hotelName
street
city
state
zip

Booking

PK hotelNo
PK guestNo
PK dateFrom

dateTo
typeRequest

Guest

PK guestNo

guestName
guestAddress

Room

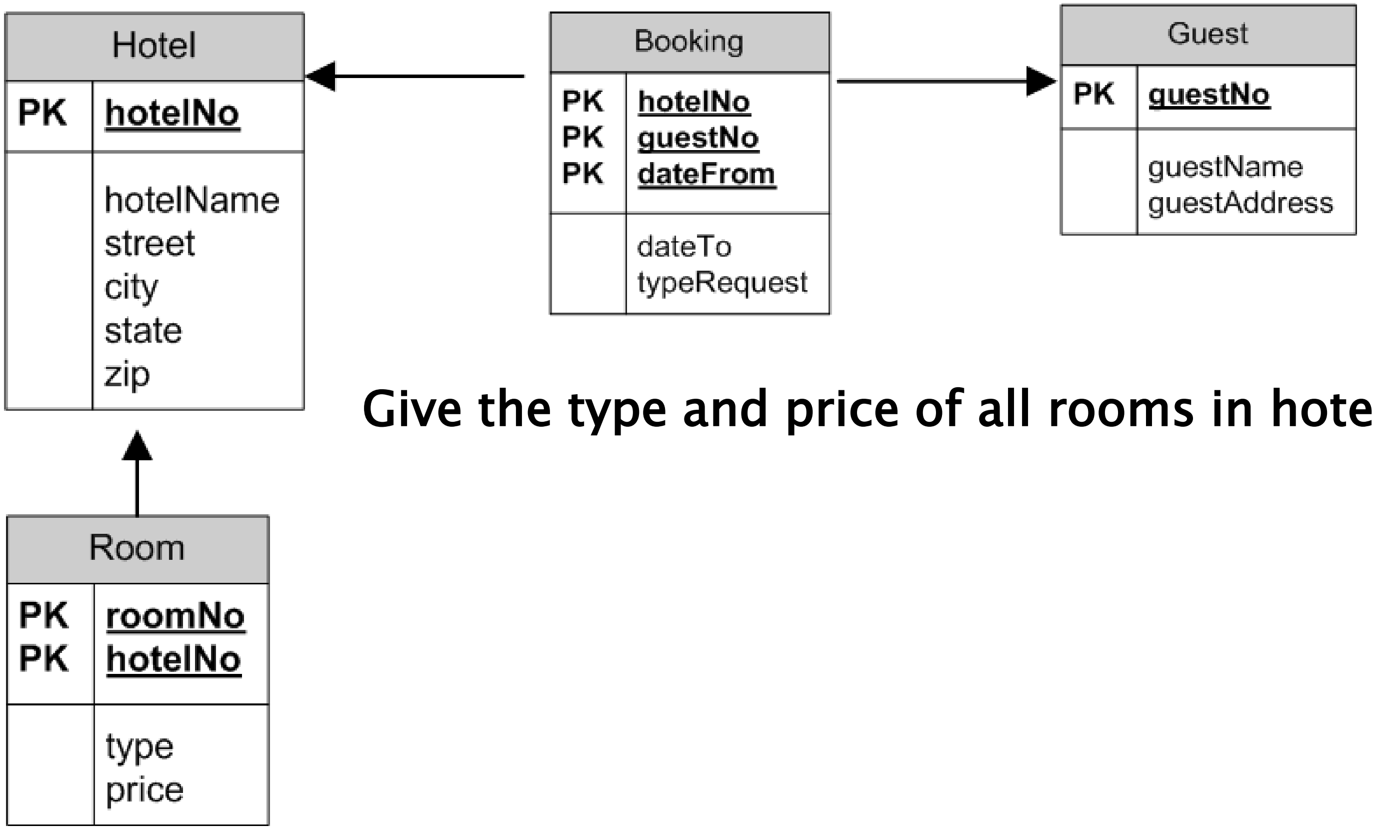
PK roomNo
PK hotelNo

type
price

Give the number of bookings for each guest who has at least two bookings:

$$\sigma_{COUNT(*)>2}\left({}_{guestNo}\Im_{COUNT(*)}\ Booking\right)$$

Query on Multiple Tables



Hotel

PK **hotelNo**

hotelName
street
city
state
zip

Booking

PK **hotelNo**
PK **guestNo**
PK **dateFrom**

dateTo
typeRequest

Guest

PK **guestNo**

guestName
guestAddress

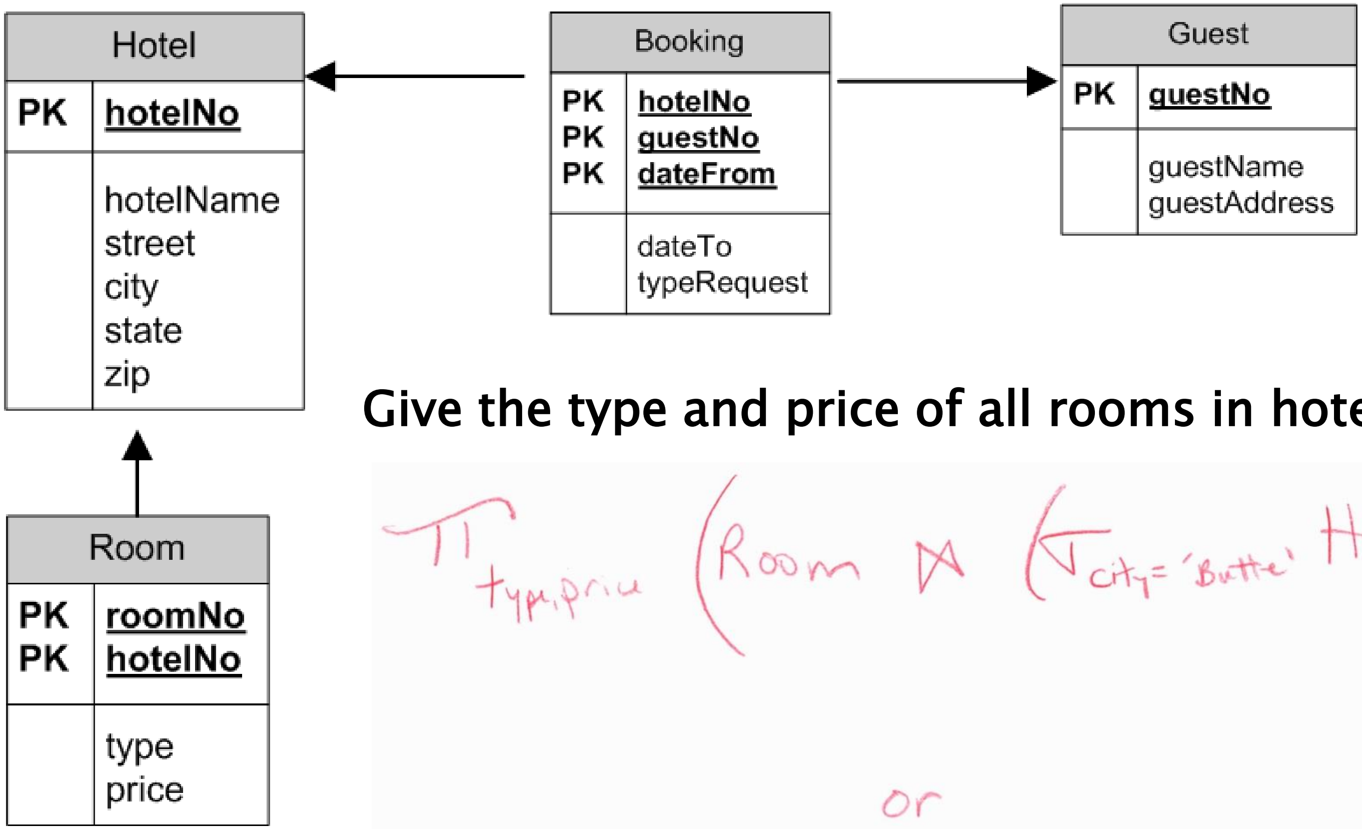
Room

PK **roomNo**
PK **hotelNo**

type
price

Give the type and price of all rooms in hotels in Butte:

Query on Multiple Tables



Hotel

PK hotelNo

hotelName
street
city
state
zip

Booking

PK hotelNo
PK guestNo
PK dateFrom

dateTo
typeRequest

Guest

PK guestNo

guestName
guestAddress

Room

PK roomNo
PK hotelNo

type
price

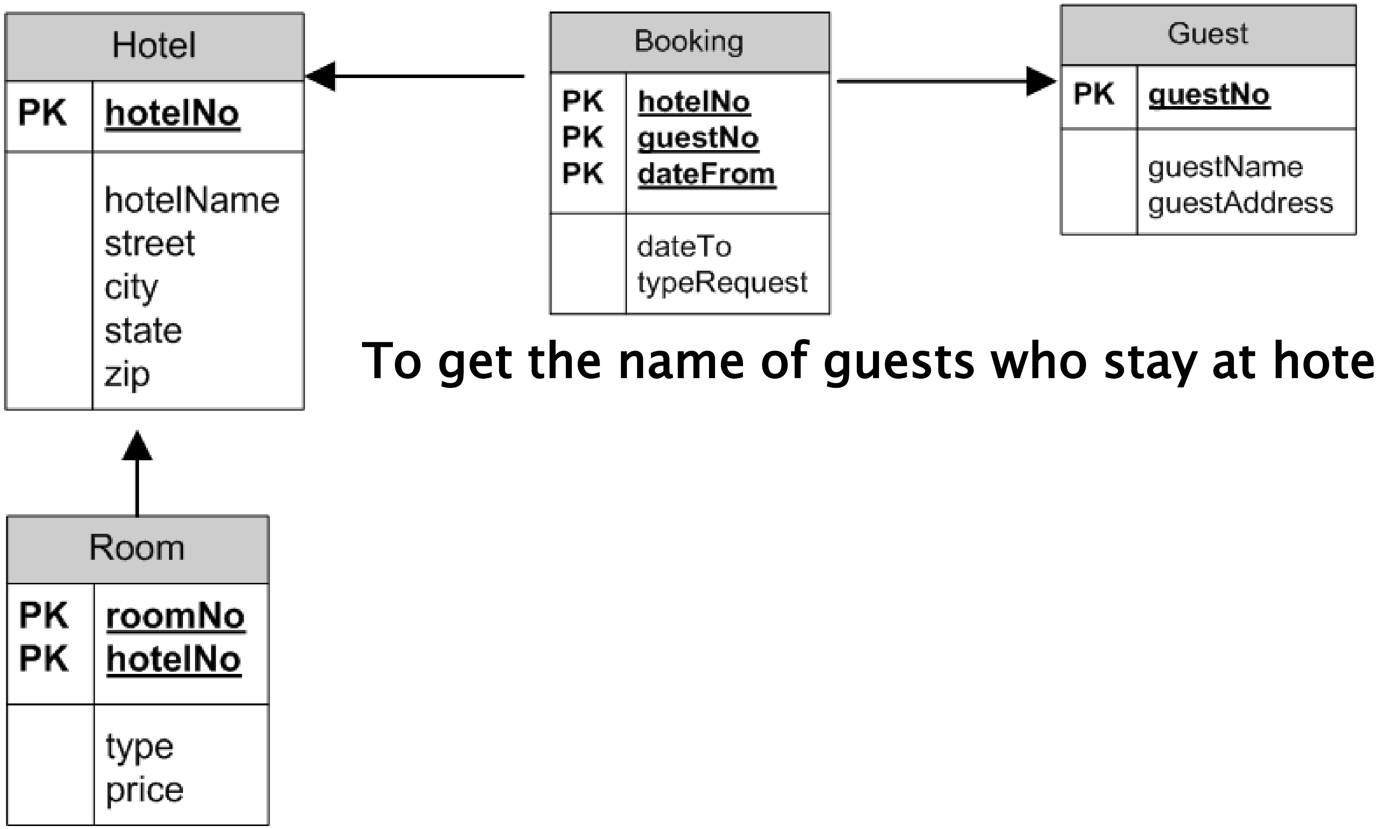
Give the type and price of all rooms in hotels in Butte:

$$\pi_{type,price}(Room \bowtie (\sigma_{city='Butte'}\ Hotel))$$

or

$$\pi_{type,price}(\sigma_{city='Butte'}(Room \bowtie Hotel))$$

Query on Multiple Tables



Hotel

PK **hotelNo**

hotelName
street
city
state
zip

Booking

PK **hotelNo**
PK **guestNo**
PK **dateFrom**

dateTo
typeRequest

Guest

PK **guestNo**

guestName
guestAddress

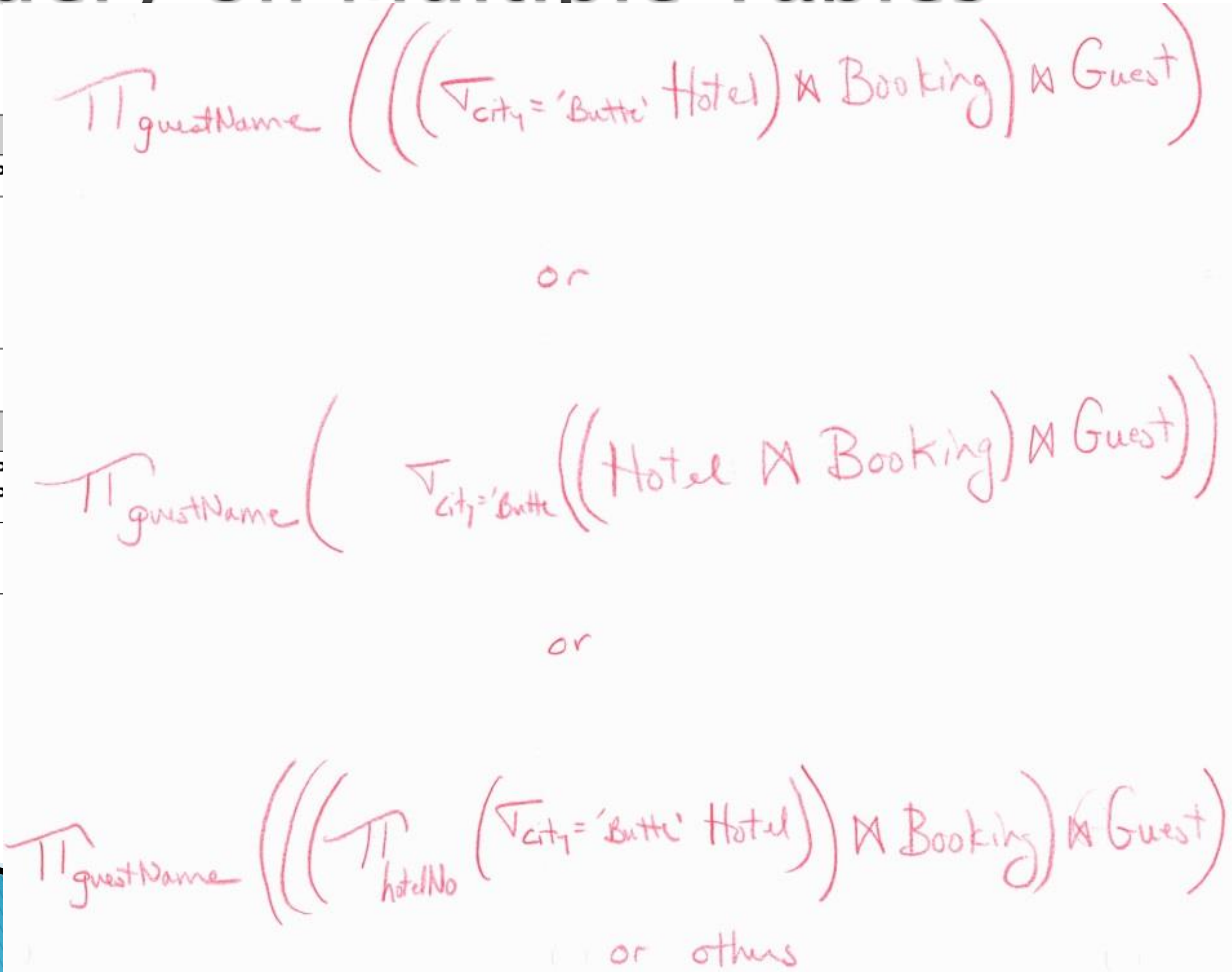
Room

PK **roomNo**
PK **hotelNo**

type
price

To get the name of guests who stay at hotels in Butte:

Query on Multiple Tables



$$\Pi_{guestName}(((\sigma_{city='Butte'} Hotel) \bowtie Booking) \bowtie Guest)$$

or

$$\Pi_{guestName}(\sigma_{city='Butte}((Hotel \bowtie Booking) \bowtie Guest))$$

or

$$\Pi_{guestName}(((\Pi_{hotelNo}(\sigma_{city='Butte'} Hotel)) \bowtie Booking) \bowtie Guest)$$

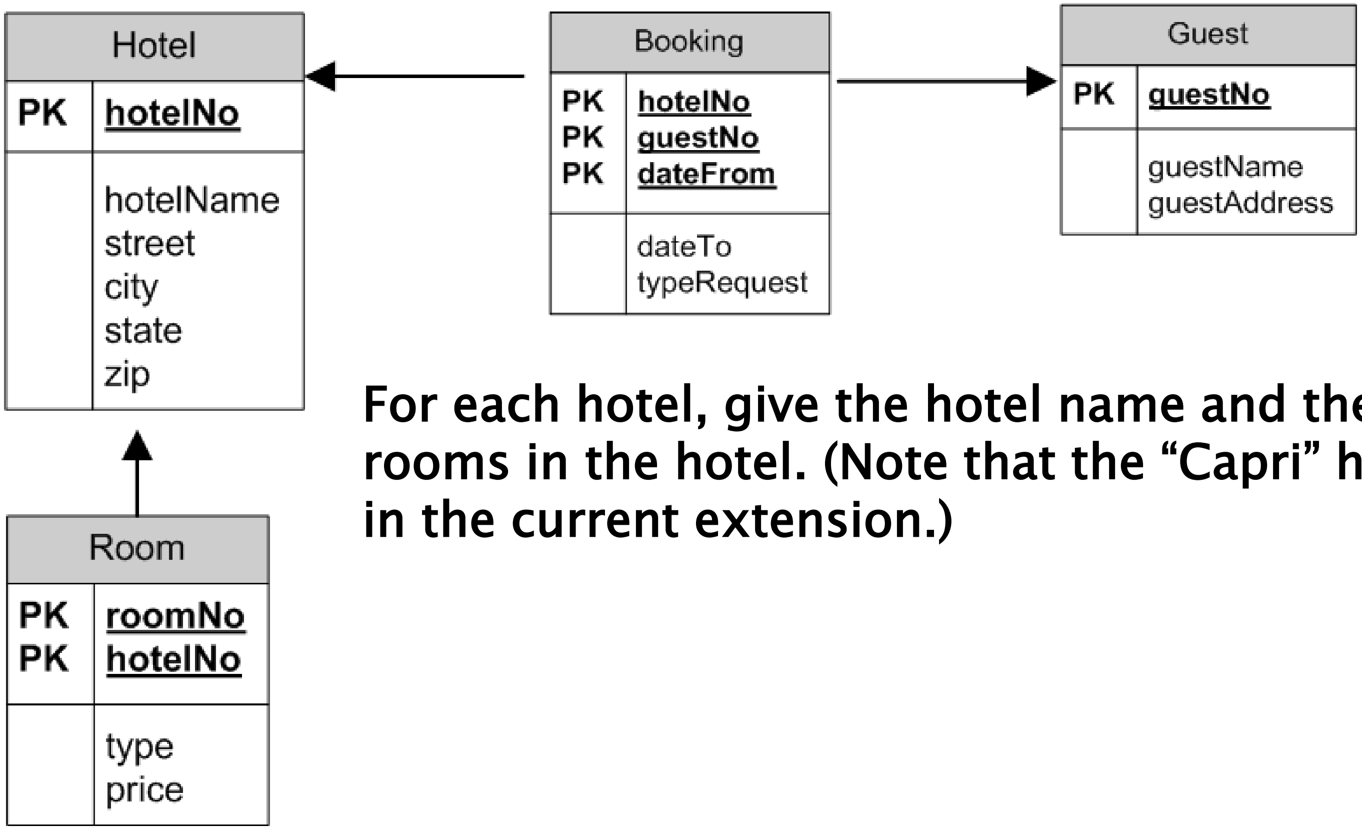
or others

P

P
P

tte:

Query Getting All



Hotel	
PK	hotelNo
	hotelName street city state zip

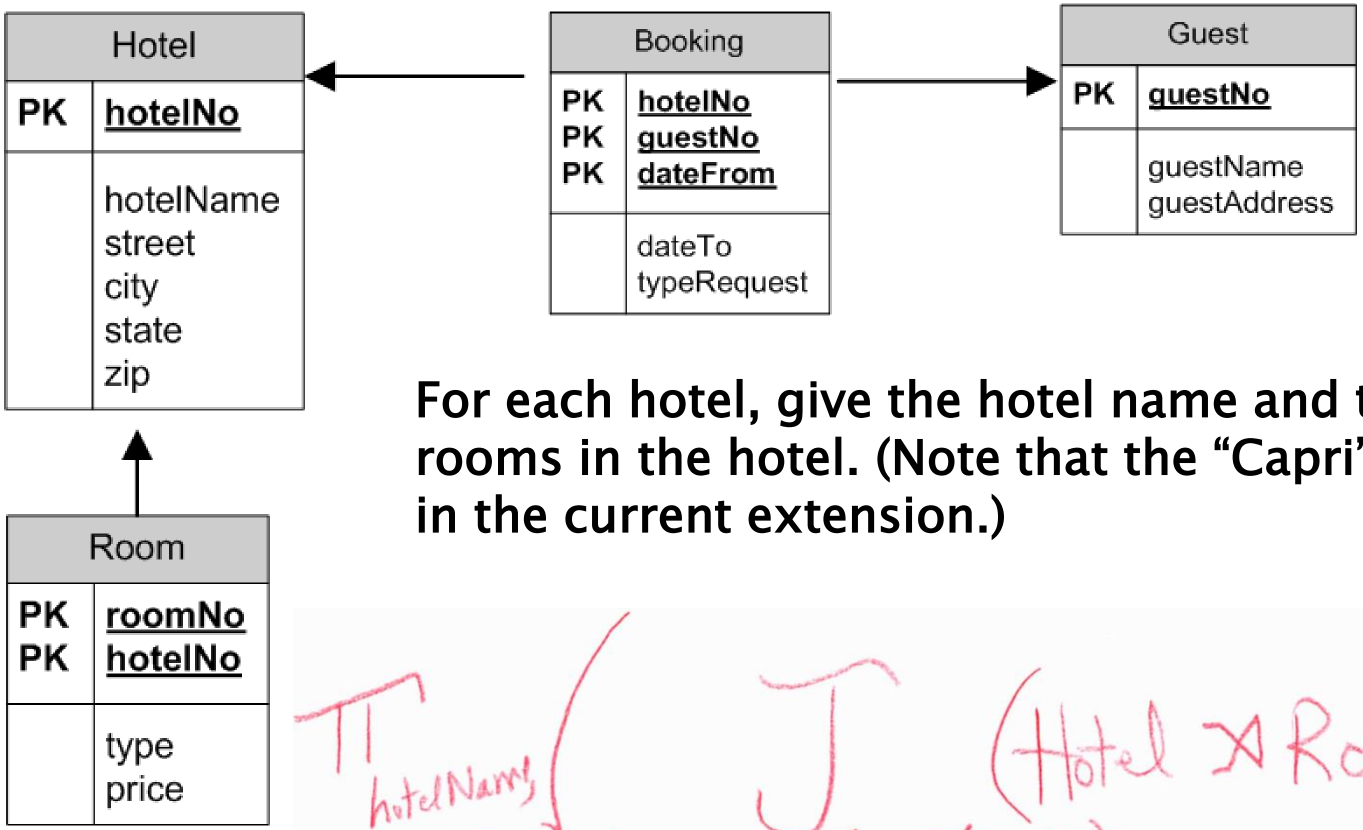
Booking	
PK PK PK	hotelNo guestNo dateFrom
	dateTo typeRequest

Guest	
PK	guestNo
	guestName guestAddress

Room	
PK PK	roomNo hotelNo
	type price

For each hotel, give the hotel name and the number of rooms in the hotel. (Note that the “Capri” has no rooms in the current extension.)

Query Getting All use Outer Join



For each hotel, give the hotel name and the number o rooms in the hotel. (Note that the “Capri” has no roon in the current extension.)

$$\pi_{hotelName,\ COUNT(roomNo)}\left({}_{hotelNo}\mathfrak{I}_{COUNT(roomNo)}\left(Hotel \mathbin{⟕} Room\right)\right)$$