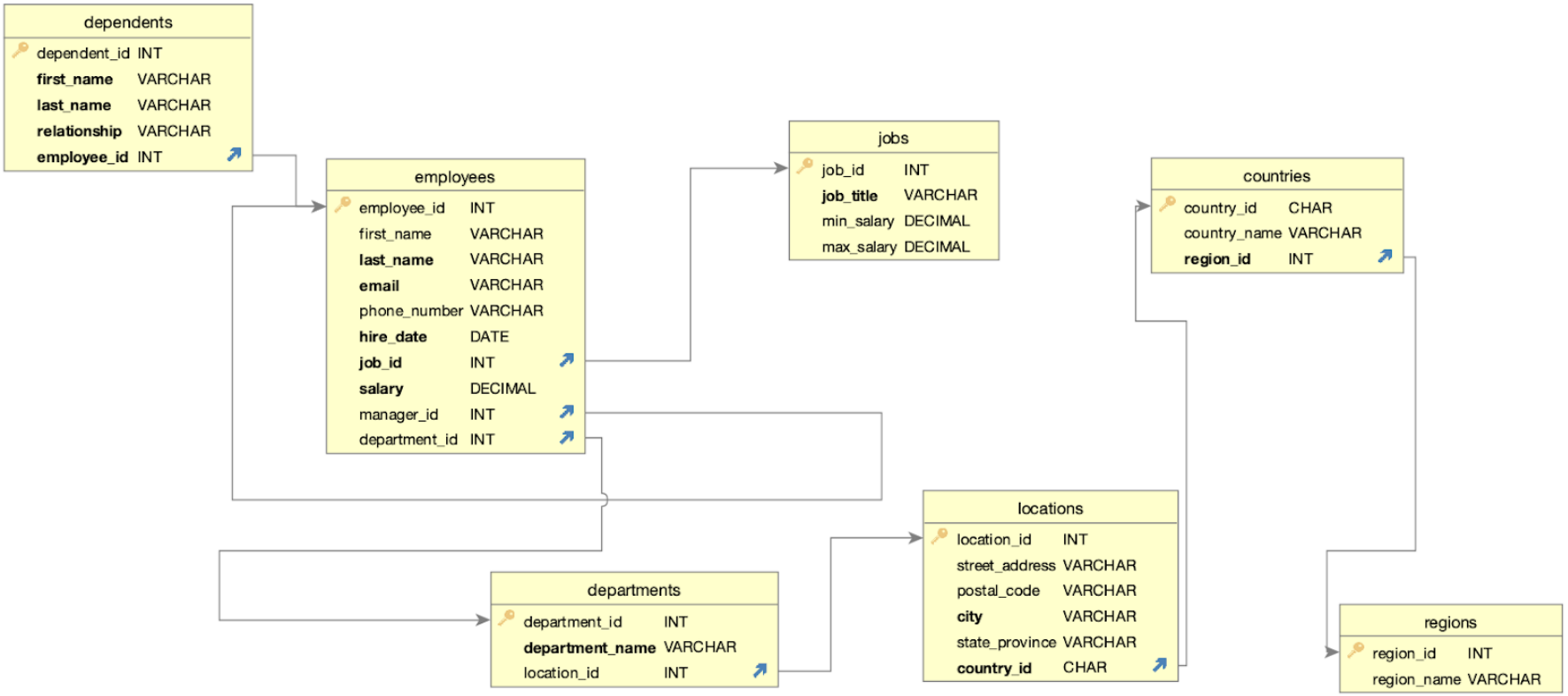
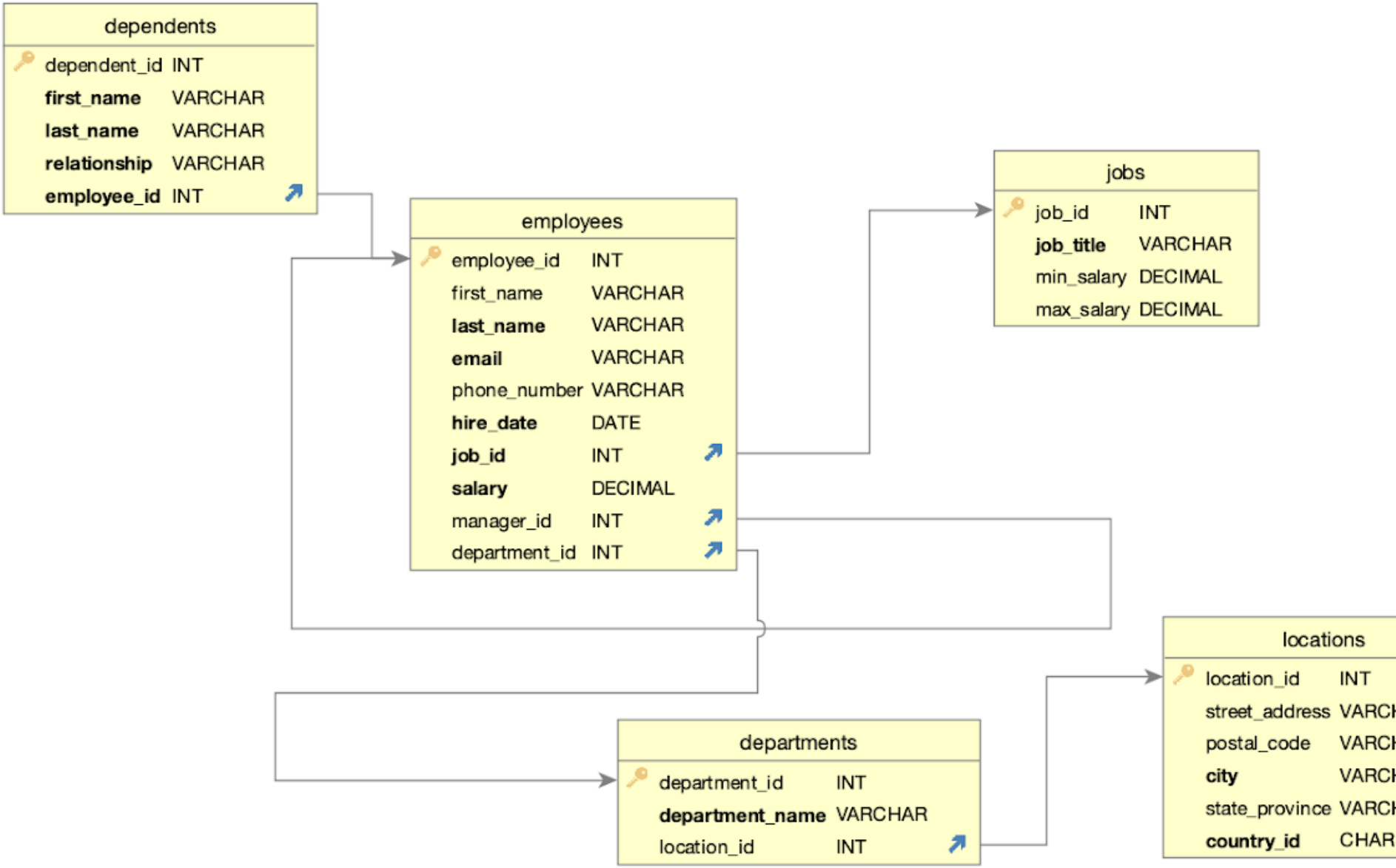
SQL

Lecture #4

Setting the Context
Using Project Stage 2


dependents
dependent_id INT
first_name VARCHAR
last_name VARCHAR
relationship VARCHAR
employee_id INT
employees
employee_id INT
first_name VARCHAR
last_name VARCHAR
email VARCHAR
phone_number VARCHAR
hire_date DATE
job_id INT
salary DECIMAL
manager_id INT
department_id INT
jobs
job_id INT
job_title VARCHAR
min_salary DECIMAL
max_salary DECIMAL
countries
country_id CHAR
country_name VARCHAR
region_id INT
locations
location_id INT
street_address VARCHAR
postal_code VARCHAR
city VARCHAR
state_province VARCHAR
country_id CHAR
departments
department_id INT
department_name VARCHAR
location_id INT
regions
region_id INT
region_name VARCHAR


dependents
dependent_id INT
first_name VARCHAR
last_name VARCHAR
relationship VARCHAR
employee_id INT
employees
employee_id INT
first_name VARCHAR
last_name VARCHAR
email VARCHAR
phone_number VARCHAR
hire_date DATE
job_id INT
salary DECIMAL
manager_id INT
department_id INT
jobs
job_id INT
job_title VARCHAR
min_salary DECIMAL
max_salary DECIMAL
departments
department_id INT
department_name VARCHAR
location_id INT
locations
location_id INT
street_address VARCH
postal_code VARCH
city VARCH
state_province VARCH
country_id CHAR

$sqlite3
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>

```
CREATE TABLE regions (
region_id INTEGER PRIMARY KEY AUTOINCREMENT NOT
NULL,
region_name text NOT NULL
);


CREATE TABLE countries (
country_id text NOT NULL,
country_name text NOT NULL,
region_id INTEGER NOT NULL,
PRIMARY KEY (country_id ASC),
FOREIGN KEY (region_id) REFERENCES regions
(region_id) ON DELETE CASCADE ON UPDATE CASCADE
);


CREATE TABLE locations (
location_id INTEGER PRIMARY KEY AUTOINCREMENT NOT
NULL,
street_address text,
postal_code text,
city text NOT NULL,
state_province text,
country_id INTEGER NOT NULL,
FOREIGN KEY (country_id) REFERENCES countries
(country_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Now load the 7 csv files into the above empty tables. To do so, execute the following steps:

1. Run the following command in SQLite
   sqlite> .mode csv
2. Now run the following command 7 times to load data into the 7 tables. Here 'filename' is the name of the csv file.
   sqlite> .import '| tail -n +2 {filepath}/{filename}.csv' {filename}

For example, suppose the 7 csv files are stored in directory /users/user1/cs564 on the machine that you are using,
then *sqlite> .import '| tail -n +2 /Users/user1/CS564_stage2/countries.csv' countries* will load data from countries.csv into table 'countries' in SQLite.

Table	Rows
employees	40
dependents	30
departments	11
jobs	19
locations	7
countries	25
regions	4

SQL Introduction

Standard language for querying and manipulating data

Structured **Q**uery **L**anguage

Many standards out there: SQL92, SQL2, SQL3, SQL99
Vendors support various subsets of these, but all of what we'll be talking about.

The Main Thing to Remember

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

- Later: the devil is in the details

Select-From-Where Statements

SELECT desired attributes

FROM one or more tables

WHERE condition about tuples of the tables

Single-Table Queries

Our Running Example

- Most of our SQL queries will be based on the following database schema.
  - Underline indicates key attributes.

Beers(<u>name</u>, manf)

Bars(<u>name</u>, addr, license)

Drinkers(<u>name</u>, addr, phone)

Likes(<u>drinker</u>, <u>beer</u>)

Sells(<u>bar</u>, <u>beer</u>, price)

Frequents(<u>drinker</u>, <u>bar</u>)

Example

- Using Beers(name, manf), what beers are made by Anheuser-Busch?

```
SELECT name
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

Beers(name manf)

Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch
Spotted Cow	New Glarus

Result of Query

name
‘Bud’
‘Bud Lite’
‘Michelob’

Beers(name manf)

Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch
Spotted Cow	New Glarus

The answer is a relation with a single attribute, name, and tuples with the name of each beer by Anheuser-Busch, such as Bud.

Meaning of Single-Relation Query

- Begin with the relation in the FROM clause.
- Apply the selection indicated by the WHERE clause.
- Apply the extended projection indicated by the SELECT clause.

Operational Semantics

- To implement this algorithm think of a *tuple variable* ranging over each tuple of the relation mentioned in FROM.
- Check if the "current" tuple satisfies the WHERE clause.
- If so, compute the attributes or expressions of the SELECT clause using the components of this tuple.

* In SELECT clauses

- When there is one relation in the FROM clause, * in the SELECT clause stands for "all attributes of this relation."
- Example using Beers(name, manf):

```
SELECT *
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

Result of Query:

name	manf
‘Bud’	‘Anheuser-Busch’
‘Bud Lite’	‘Anheuser-Busch’
‘Michelob’	‘Anheuser-Busch’

Now, the result has each of the attributes of Beers.

Another Example

Company(sticker, name, country, stockPrice)

Find all US companies whose stock is > 50:

```
SELECT   *
FROM     Company
WHERE    country="USA" AND stockPrice > 50
```

Output schema: R(sticker, name, country, stockPrice)

Renaming Attributes

- If you want the result to have different attribute names, use "AS <new name>" to rename an attribute.
- Example based on Beers(name, manf):

```
SELECT name AS beer, manf
FROM Beers
WHERE manf = 'Anheuser-Busch'
```

Result of Query:

beer	manf
‘Bud’	‘Anheuser-Busch’
‘Bud Lite’	‘Anheuser-Busch’
‘Michelob’	‘Anheuser-Busch’

Expressions in SELECT Clauses

- Any expression that makes sense can appear as an element of a SELECT clause.
- Example: from Sells(bar, beer, price):

```
SELECT bar, beer,
       price * 120 AS priceInYen
FROM Sells;
```

Result of Query

bar	beer	priceInYen
Joe’s	Bud	300
Sue’s	Miller	360
…	…	…

Another Example: Constant Expressions

- From Likes(drinker, beer):

```
SELECT drinker,
       'likes Bud' AS whoLikesBud
FROM Likes
WHERE beer = 'Bud';
```

Result of Query

drinker	whoLikesBud
‘Sally’	‘likes Bud’
‘Fred’	‘likes Bud’
…	…

Complex Conditions in WHERE Clause

- From Sells(bar, beer, price), find the price Joe's Bar charges for Bud:

```
SELECT price
FROM Sells
WHERE bar = 'Joe''s Bar' AND
      beer = 'Bud';
```

Selections

What you can use in WHERE:

attribute names of the relation(s) used in the FROM.
comparison operators: =, <>, <, >, <=, >=
apply arithmetic operations: stockprice*2
operations on strings (e.g., "||" for concatenation).
Lexicographic order on strings.
Pattern matching: s LIKE p
Special stuff for comparing dates and times.

Important Points

- Two single quotes inside a string represent the single-quote (apostrophe).
- Conditions in the WHERE clause can use AND, OR, NOT, and parentheses in the usual way boolean conditions are built.
- SQL is *case-insensitive*. In general, upper and lower case characters are the same, except inside quoted strings.

Patterns

- WHERE clauses can have conditions in which a string is compared with a pattern, to see if it matches.
- General form: <Attribute> LIKE <pattern>
  or <Attribute> NOT LIKE <pattern>

- Pattern is a quoted string with % = “any string”; _ = “any character.”

Example

- From Drinkers(name, addr, phone) find the drinkers with exchange 555:

```
SELECT name
FROM Drinkers
WHERE phone LIKE '%555-_ _ _ _';
```

The **LIKE** operator

- s **LIKE** p: pattern matching on strings
- p may contain two special symbols:
  - % = any sequence of characters
  - _ = any single character

Company(sticker, name, address, country, stockPrice)

Find all US companies whose address contains "Mountain":

```
SELECT  *
FROM    Company
WHERE   country="USA" AND
        address LIKE "%Mountain%"
```

Multi-Table Queries

Multirelation Queries

- Interesting queries often combine data from more than one relation.
- We can address several relations in one query by listing them all in the FROM clause.
- Distinguish attributes of the same name by "<relation>.<attribute>"

Example

- Using relations Likes(drinker, beer) and Frequents(drinker, bar), find the beers liked by at least one person who frequents Joe's Bar.

```
SELECT beer
FROM Likes, Frequents
WHERE bar = 'Joe''s Bar' AND
   Frequents.drinker = Likes.drinker;
```

Likes(drinker, beer)

a	x
b	y
c	z

Frequents(drinker, bar)

a	t
b	u
c	t

Find beers liked by at least one person who frequents bar t.

Select beer
From Likes, Frequents
Where bar = ‘t’ and
Likes.drinker = Frequents.drinker

Likes(drinker, beer)

a	x
b	y
c	z

```
Select beer
From Likes, Frequents
Where bar = 't' and
      Likes.drinker = Frequents.drinker
```

Frequents(drinker, bar)

a	t
b	u
c	t

Solution:

1) Enumerate all combinations of tuples from Likes and Frequents
2) Keep only combinations that satisfy the condition in Where clause
3) Return beer from those combinations

Likes(drinker, beer)

a	x
b	y
c	z

Frequents(drinker, bar)

a	t
b	u
c	t

Select beer
From Likes, Frequents
Where bar = ‘t’ and
Likes.drinker = Frequents.drinker

Solution:
1) Enumerate all combinations of tuples from Likes and Frequents
2) Keep only combinations that satisfy the condition in Where clause
3) Return beer from those combinations

(a,x) (a,t)
(a,x) (b,u)
(a,x) (c,t)
(b,y) (a,t)
(b,y) (b,u)
(b,y) (c,t)
(c,z) (a,t)
(c,z) (b,u)
(c,z) (c,t)

(a,x) (a,t)
(a,x) (b,u)
(a,x) (c,t)
(b,y) (a,t)
(b,y) (b,u)
(b,y) (c,t)
(c,z) (a,t)
(c,z) (b,u)
(c,z) (c,t)

Output(beer)

x

z

Likes(drinker, beer)

a	x
b	y
c	z

Frequents(drinker, bar)

a	t
b	u
c	t

Select beer
From Likes, Frequents
Where bar = ‘t’ and
Likes.drinker = Frequents.drinker

Solution:
1) Enumerate all combinations of tuples from Likes and Frequents
2) Keep only combinations that satisfy the condition in Where clause
3) Return beer from those combinations

How to do this fast? (Will cover this later in the course)

1. Have an index on bar for table Frequents
2. Use the index to quickly find tuples in Frequents with bar = t
3. Find drinkers in these tuples (a and c in this case)
4. Have an index on drinker for table Likes
5. Use the index to quickly find tuples in Likes with drinker a, c
6. Find bars in these tuples and return (x, z)

Many different ways to do this, some may be much faster than others

Another Example

Product (pname, price, category, maker)
Purchase (buyer, seller, store, product)
Company (cname, stockPrice, country)
Person(pname, phoneNumber, city)

Find names of people living in Champaign that bought gizmo products, and the names of the stores they bought from

```
SELECT   pname, store
FROM     Person, Purchase
WHERE    pname=buyer AND city="Champaign"
         AND product="gizmo"
```

Disambiguating Attributes

Find names of people buying telephony products:

Product (name, price, category, maker)
Purchase (buyer, seller, store, product)
Person(name, phoneNumber, city)

```
SELECT   Person.name
FROM     Person, Purchase, Product
WHERE    Person.name=Purchase.buyer
     AND product=Product.name
     AND Product.category="telephony"
```

Disambiguating Attributes

Find names of people buying telephony products:

Product (name, price, category, maker)
Purchase (buyer, seller, store, product)
Person(name, phoneNumber, city)

```
SELECT   Person.name
FROM     Person x, Purchase y, Product z
WHERE    x.name=y.buyer
     AND  y.product=z.name
     AND  z.category="telephony"
```

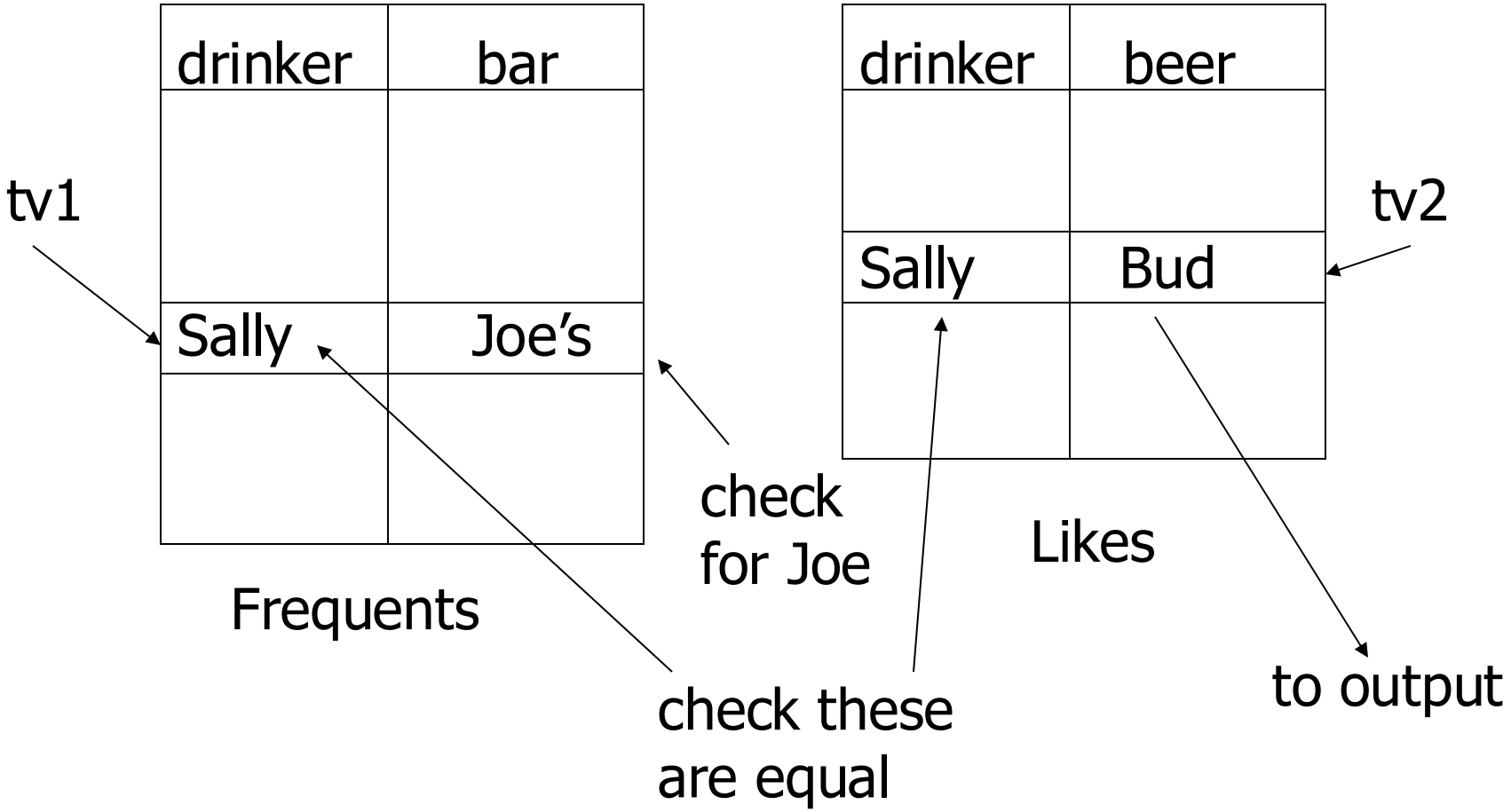
Formal Semantics

- Almost the same as for single-relation queries:
  1. Start with the product of all the relations in the FROM clause.
  2. Apply the selection condition from the WHERE clause.
  3. Project onto the list of attributes and expressions in the SELECT clause.

Operational Semantics

- Imagine one tuple-variable for each relation in the FROM clause.
  - These tuple-variables visit each combination of tuples, one from each relation.
- If the tuple-variables are pointing to tuples that satisfy the WHERE clause, send these tuples to the SELECT clause.

Example


drinker
bar
Sally
Joe's
Frequents
drinker
beer
Sally
Bud
Likes
tv1
tv2
check
for Joe
check these
are equal
to output

Explicit Tuple-Variables

- Sometimes, a query needs to use two copies of the same relation.
- Distinguish copies by following the relation name by the name of a tuple-variable, in the FROM clause.
- It's always an option to rename relations this way, even when not essential.

Example

- From Beers(name, manf), find all pairs of beers by the same manufacturer.
  - Do not produce pairs like (Bud, Bud).
  - Produce pairs in alphabetic order, e.g. (Bud, Miller), not (Miller, Bud).

```
SELECT b1.name, b2.name
FROM Beers b1, Beers b2
WHERE b1.manf = b2.manf AND
   b1.name < b2.name;
```

Tuple Variables

Find pairs of companies making products in the same category

```
SELECT  product1.maker, product2.maker
FROM    Product AS product1, Product AS product2
WHERE   product1.category=product2.category
    AND  product1.maker <> product2.maker
```

Product (name, price, category, maker)

Tuple Variables

Tuple variables introduced automatically by the system:

Product (name, price, category, maker)

Becomes:

```
SELECT  name
FROM    Product
WHERE   price > 100
```

```
SELECT Product.name
FROM   Product AS Product
WHERE Product.price > 100
```

Doesn't work when Product occurs more than once:
In that case the user needs to define variables explicitly.

Meaning (Semantics) of SQL Queries

SELECT a1, a2, …, ak
FROM R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE Conditions

1. Nested loops:

```
Answer = {}
for x1 in R1 do
    for x2 in R2 do
        …..
            for xn in Rn do
                if Conditions
                    then Answer = Answer U {(a1,…,ak)
return Answer
```

Meaning (Semantics) of SQL Queries

SELECT a1, a2, …, ak
FROM R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE Conditions

2. Parallel assignment

```
Answer = {}
for all assignments x1 in R1, …, xn in Rn do
        if Conditions then Answer = Answer U {(a1,…,ak)}
return Answer
```

Doesn't impose any order !

Tip for Writing SQL Queries

Product (pname, price, category, maker)
Purchase (buyer, seller, store, product)
Company (cname, stockPrice, country)
Person(pname, phoneNumber, city)

Find names of people living in Champaign that bought gizmo products, and the names of the stores they bought from

```
SELECT   pname, store
FROM     Person, Purchase
WHERE    pname=buyer AND city="Champaign"
         AND product="gizmo"
```

Exercises

Product (pname, price, category, maker)
Purchase (buyer, seller, store, product)
Company (cname, stock price, country)
Person(per-name, phone number, city)

Ex #1: Find people who bought telephony products.
Ex #2: Find names of people who bought American products
Ex #3: Find names of people who bought American products and did not buy French products
Ex #4: Find names of people who bought American products and they live in Champaign.
Ex #5: Find people who bought stuff from Joe or bought products from a company whose stock prices is more than $50.

Aggregation

Aggregations

- SUM, AVG, COUNT, MIN, and MAX can be applied to a column in a SELECT clause to produce that aggregation on the column.
- Also, COUNT(*) counts the number of tuples.

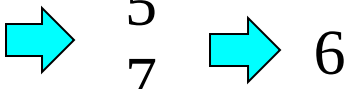
Example: Aggregation

- From Sells(bar, beer, price), find the average price of beer x:

```
SELECT AVG(price)
FROM Sells
WHERE beer = 'x';
```

Sells(bar, beer, price)

t,	x,	5
t,	y,	4
u,	x,	7
u,	y,	8



5
7

6

Group-by and Having

Group-By and Having

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

1. Enumerate all combinations of tuples from R1, …, Rn
2. Keep only combinations satisfying condition C1
3. Group them by a1, …, ak
4. Keep only groups satisfying condition C2
5. Pull out from each group the values requested in S
   + if any aggregation, then apply within the group

Likes(drinker, beer)

a	x
b	y
c	z

Frequents(drinker, bar)

a	t
b	u
c	t

Sells(bar, beer, price)

t,	x,	5
t,	y,	4
u,	x,	7
u,	y,	8

For each beer find the minimum price that it is being sold in a bar

Select beer, min(price)
From Sells
Groupby beer

(t, x, 5)
(u, x, 7)

(t, y, 4)
(u, y, 8)

Output table: (x, 5)
(y, 4)

Likes(drinker, beer)

a	x
b	y
c	z

Frequents(drinker, bar)

a	t
b	u
c	t

Sells(bar, beer, price)

t,	x,	5
t,	y,	4
u,	x,	7
u,	y,	8

For each bar find the average price at which it sells beers

Select bar, avg(price)
From Sells
Groupby bar

(t, x, 5)
(t, y, 4)

(u, x, 7)
(u, y, 8)

Output table: (t, 4.5)
(u, 7.5)

Likes(drinker, beer)

a	x
b	y
c	z

Frequents(drinker, bar)

a	t
b	u
c	t

Sells(bar, beer, price)

t,	x,	5
t,	y,	4
u,	x,	7
u,	y,	8

Find all bars where the avg price of beer exceeds $5 and then list the most expensive price for each such bar.

Select bar, max(price)
From Sells
Groupby bar
Having avg(price) > 5

(t, x, 5)
(t, y, 4)

(u, x, 7)
(u, y, 8)

Output table: (u, 8)

Likes(drinker, beer)

a	x
b	y
c	z

Frequents(drinker, bar)

a	t
b	u
c	t

Sells(bar, beer, price)

t,	x,	5
t,	y,	4
u,	x,	7
u,	y,	8
v,	z,	100

Find all bars that sell only beer x or y, and where the avg price of beer exceeds $5, and then list the most expensive price for each such bar.

Select bar, max(price)
From Sells
Where beer = x OR beer = y
Groupby bar
Having avg(price) > 5

(t, x, 5)
(t, y, 4)

(u, x, 7)
(u, y, 8)

Output table: (u, 8)

Likes(drinker, beer)

a	x
b	y
c	z

Frequents(drinker, bar)

a	t
b	u
b	t

Sells(bar, beer, price)

t,	x,	5
t,	y,	4
u,	x,	7
u,	y,	8
v,	z,	100

For each drinker find the average price of beer x at the bars they frequent

Select drinker, avg(price)
From Frequents F, Sells S
Where F.bar = S.bar and S.beer = x
Groupby drinker

(a,t) (t,x,5)
(b,u) (u,x,7)
(b,t) (t,x,5)

Output table: (a, 5)
(b, 6)

“Common Sense” Requirements

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

S = may contain attributes $a_1,\ldots,a_k$ and/or any aggregates but NO OTHER ATTRIBUTES

C1 = is any condition on the attributes in $R_1,\ldots,R_n$

C2 = is any condition on aggregate expressions

General form of Grouping and Aggregation

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

Evaluation steps:

1. Compute the FROM-WHERE part, obtain a table with all attributes in $R_1,\ldots,R_n$
2. Group by the attributes $a_1,\ldots,a_k$
3. Compute the aggregates in C2 and keep only groups satisfying C2
4. Compute aggregates in S and return the result