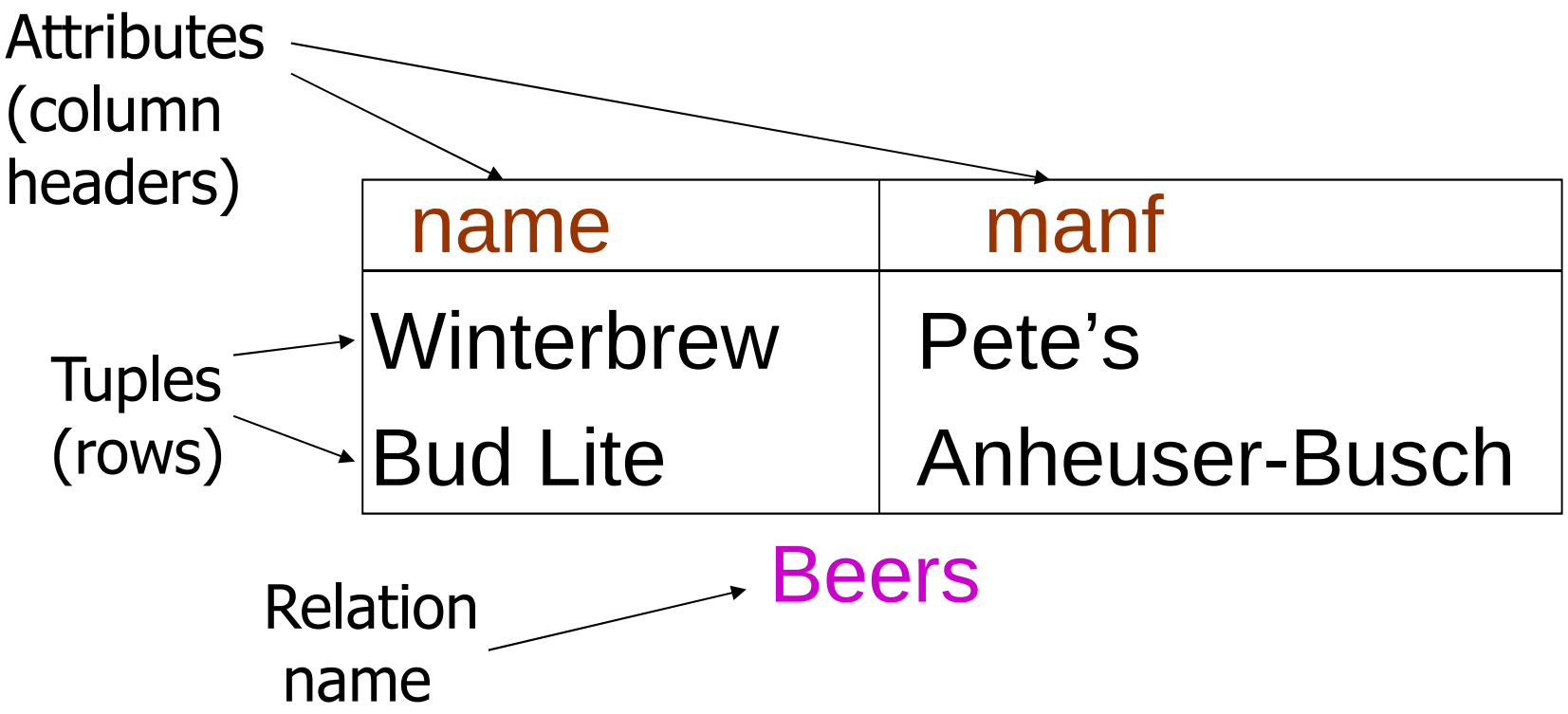
Database Management Systems - I, CS 157A

SQL Overview and SELECT

Agenda

- Create Schema (DDL)
- Query
  - Select-From-Where Statements
  - Multi-relation Queries
  - Sub-queries

Review: A Relation is a Table

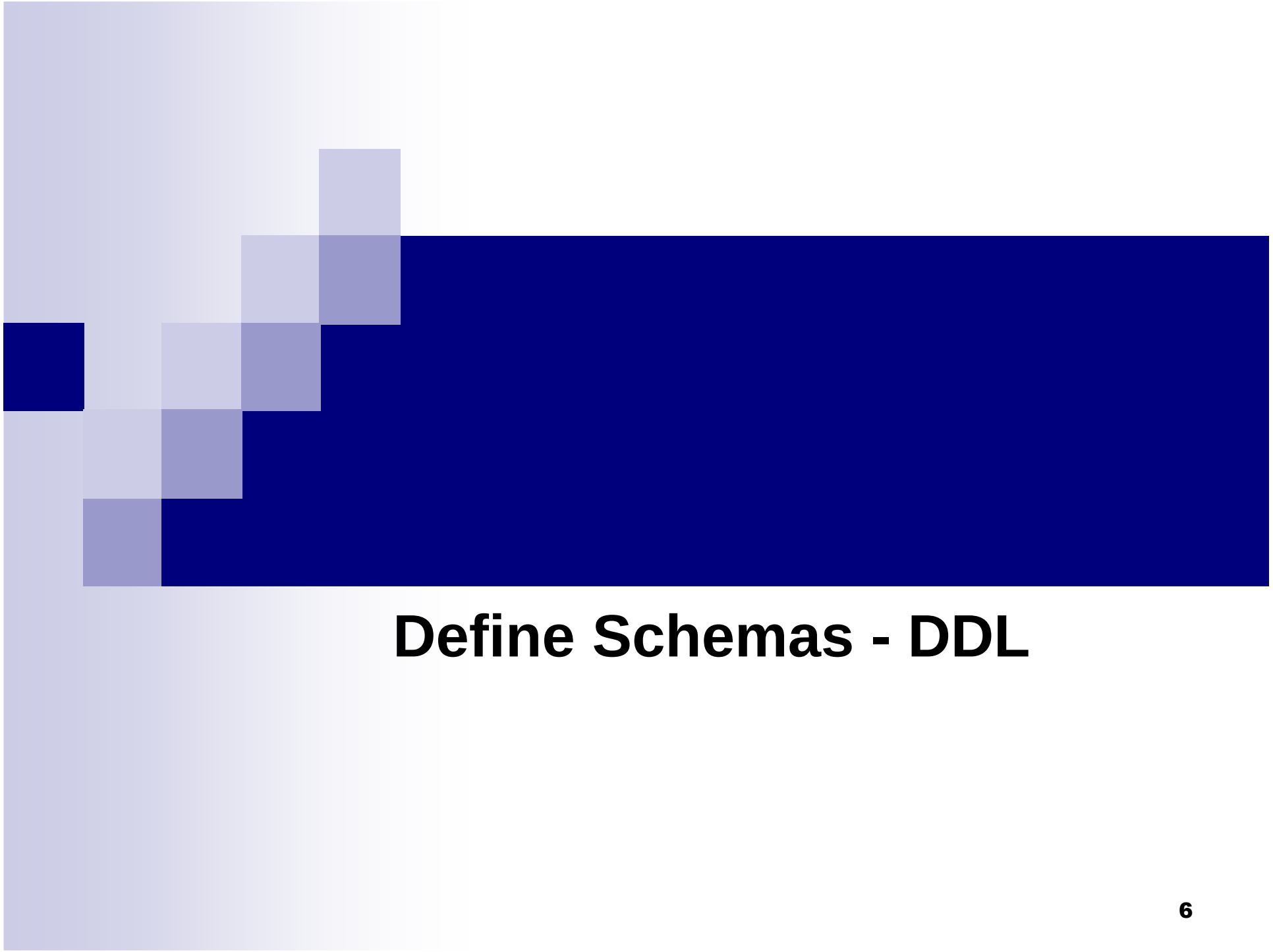


Review: From E/R Diagrams to Relations

- **Entity set** → relation.
  - Attributes → attributes.
- **Relationships** → relations whose attributes are only:
  - The keys of the connected entity sets.
  - Attributes of the relationship itself.

Review: Core Relational Algebra

- **Union, intersection, and difference:**
  - Usual set operations, but *both operands must have the same relation schema*.
- **Selection:** picking certain rows.
- **Projection:** picking certain columns.
- **Products** and **Joins**: compositions of relations.
- **Renaming** of relations and attributes.



Define Schemas - DDL

Database Schemas in SQL

- Structured Query Language (SQL)
- SQL is primarily a ***query*** language, for getting information from a database.
- But SQL also includes a *data-definition* component for describing database schemas.

SQL Statements

DML (Data Manipulation Language)	SELECT
	INSERT UPDATE DELETE
DDL (Data Definition Language)	CREATE ALTER DROP
DCL and Transaction Control	GRANT REVOKE COMMIT ROLLBACK

Creating (Declaring) a Relation

- Simplest form is:

```
CREATE TABLE <name> (
        <list of elements>
);
```

- To delete a relation:

```
DROP TABLE <name>;
```

Example: Create Table

```
CREATE TABLE Sells (
        bar     CHAR(20),
        beer    VARCHAR(20),
        price   REAL
);
```

Declaring Single-Attribute Keys

- Place PRIMARY KEY or UNIQUE after the type in the declaration of the attribute.
- Example:

```
CREATE TABLE Beers (
    name    CHAR(20) UNIQUE,
    manf    CHAR(20)
);
```

Example: Multi-attribute Key

- The bar and beer together are the key for *Sells*:

```
CREATE TABLE Sells (
        bar         CHAR(20),
        beer        VARCHAR(20),
        price       REAL,
        PRIMARY KEY (bar, beer)
);
```

PRIMARY KEY vs. UNIQUE

1. There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes.
2. No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared UNIQUE may have NULL's, and there may be several tuples with NULL.

Query

SQL Query

- SQL is a very-high-level language:
  - Say "**what to do**" rather than "how to do it."
  - Avoid a lot of data-manipulation details needed in procedural languages like C++ or Java.
- Database management system figures out "best" way to execute query:
  - Called "query optimization."

Select-From-Where Statements

SELECT desired attributes

FROM one or more tables

WHERE condition about tuples of the tables

Our Running Example

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

Beers(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Example

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

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

Result of Query

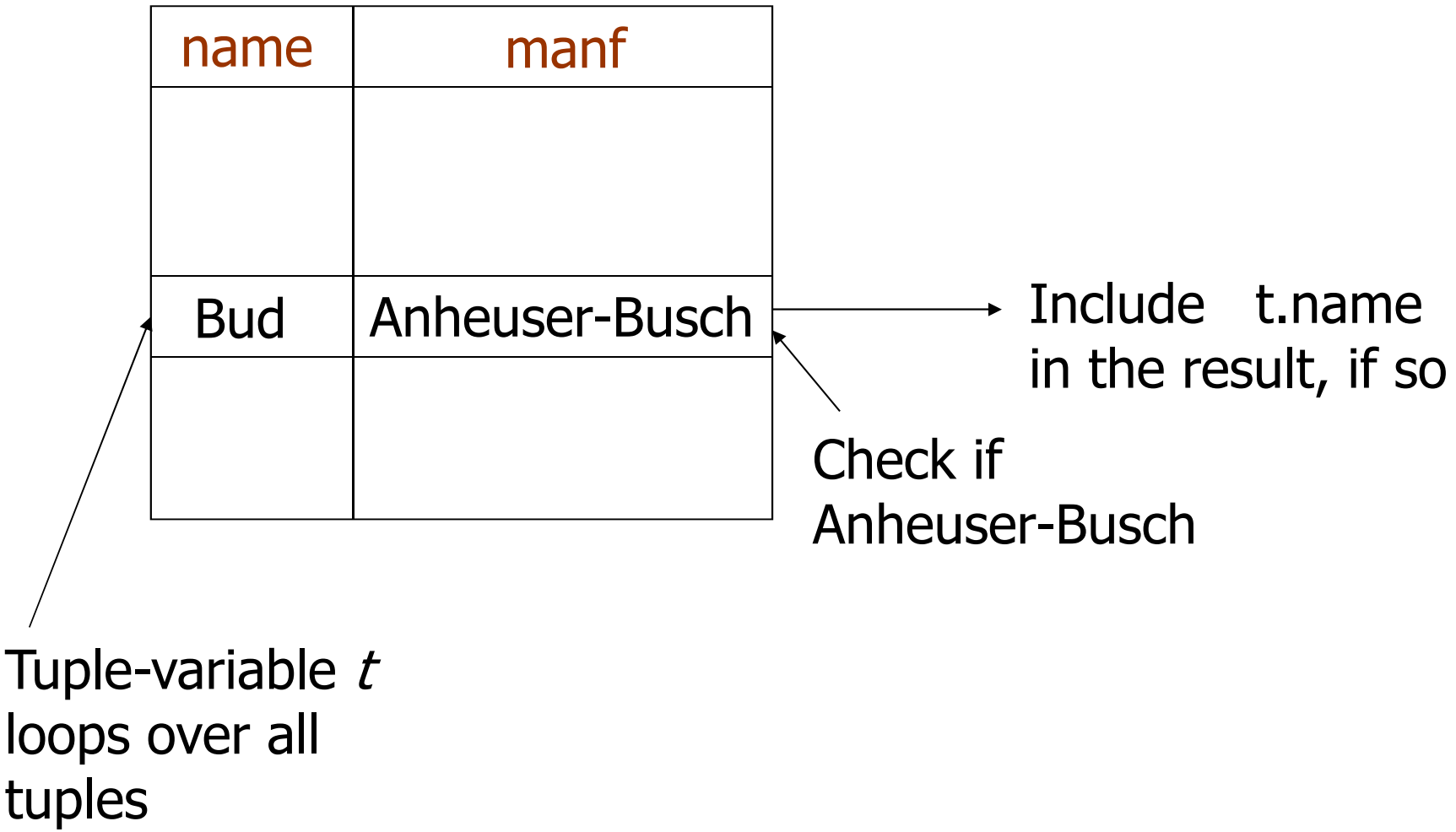
name
Bud
Bud Lite
Michelob
. . .

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



"*" 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.

Renaming Attributes

- If you want the result to have different attribute names, use "AS <new name>" to rename an attribute.

- Example: Using 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: Using Sells(bar, beer, price):

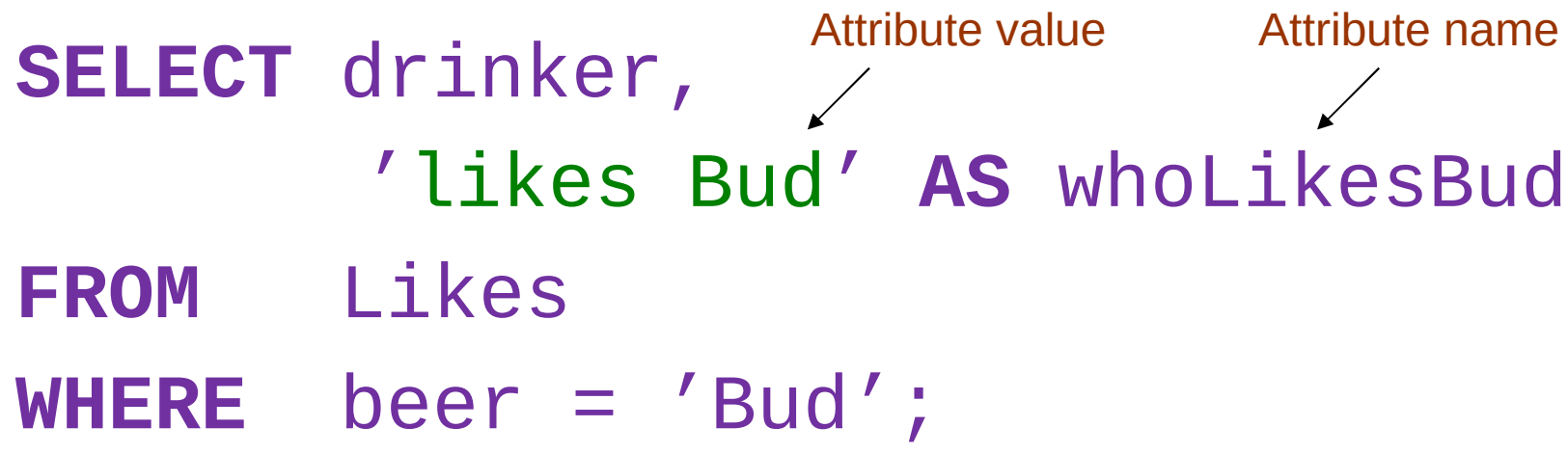
```
SELECT bar, beer,
       price*114 AS priceInYen
FROM   Sells;
```

Result of Query

bar	beer	priceInYen
Joe's	Bud	285
Sue's	Miller	342
...	...	...

Example: Constants as Expressions

- Using Likes(drinker, beer):



Result of Query

drinker	whoLikesBud
Sally	likes Bud
Fred	likes Bud
…	…

Example: Information Integration

- We often build “data warehouses” from the data at many “sources.”
- Suppose each bar has its own Menu relation Menu(beer, price) .
- To contribute to Sells(bar, beer, price) we need to query each bar and insert the name of the bar.

Information Integration (cont.)

- For instance, at Joe's Bar we can issue the query:

```
SELECT 'Joe''s Bar', beer,
       price
FROM   Menu;
```

Complex Conditions in WHERE Clause

- Boolean operators AND, OR, NOT.
- Comparisons =, <>, <, >, <=, >=.
  - And many other operators that produce boolean-valued results.

Example: Complex Condition

- Using 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';
```

Patterns

- A condition can compare a string to a pattern by:
  - <Attribute> LIKE <pattern> or <Attribute> NOT LIKE <pattern>
- *Pattern* is a quoted string with
  - % = “any string.”
  - _ = “any single character.”

Example: LIKE

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

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

NULL Values

- Tuples in SQL relations can have NULL as a value for one or more components.
- **Meaning depends on context. Two common cases:**
  - ***Missing value*:** e.g., we know Joe's Bar has some address, but we don't know what it is.
  - ***Inapplicable*:** e.g., the value of attribute spouse for an unmarried person.

Comparing NULL's to Values

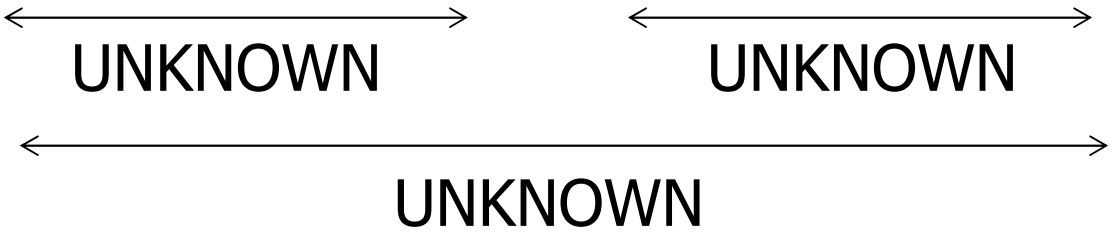
- The logic of conditions in SQL is really 3-valued logic: TRUE, FALSE, UNKNOWN.
- Comparing any value (including NULL itself) with NULL yields UNKNOWN.
- A tuple is in a query answer iff the WHERE clause is TRUE (not FALSE or UNKNOWN).

Surprising Example

- **From the following Sells relation:**

bar	beer	price
Joe's Bar	Bud	NULL

```
SELECT bar
FROM   Sells
WHERE  price < 2.00 OR price >= 2.00;
```



Multi-table Queries

- Interesting queries often combine data from more than one table.
- We can address several tables in one query by listing them all in the FROM clause.
- Distinguish attributes of the same name in 2 tables by using “<table>.<attribute>” .

Example: Joining Two tables

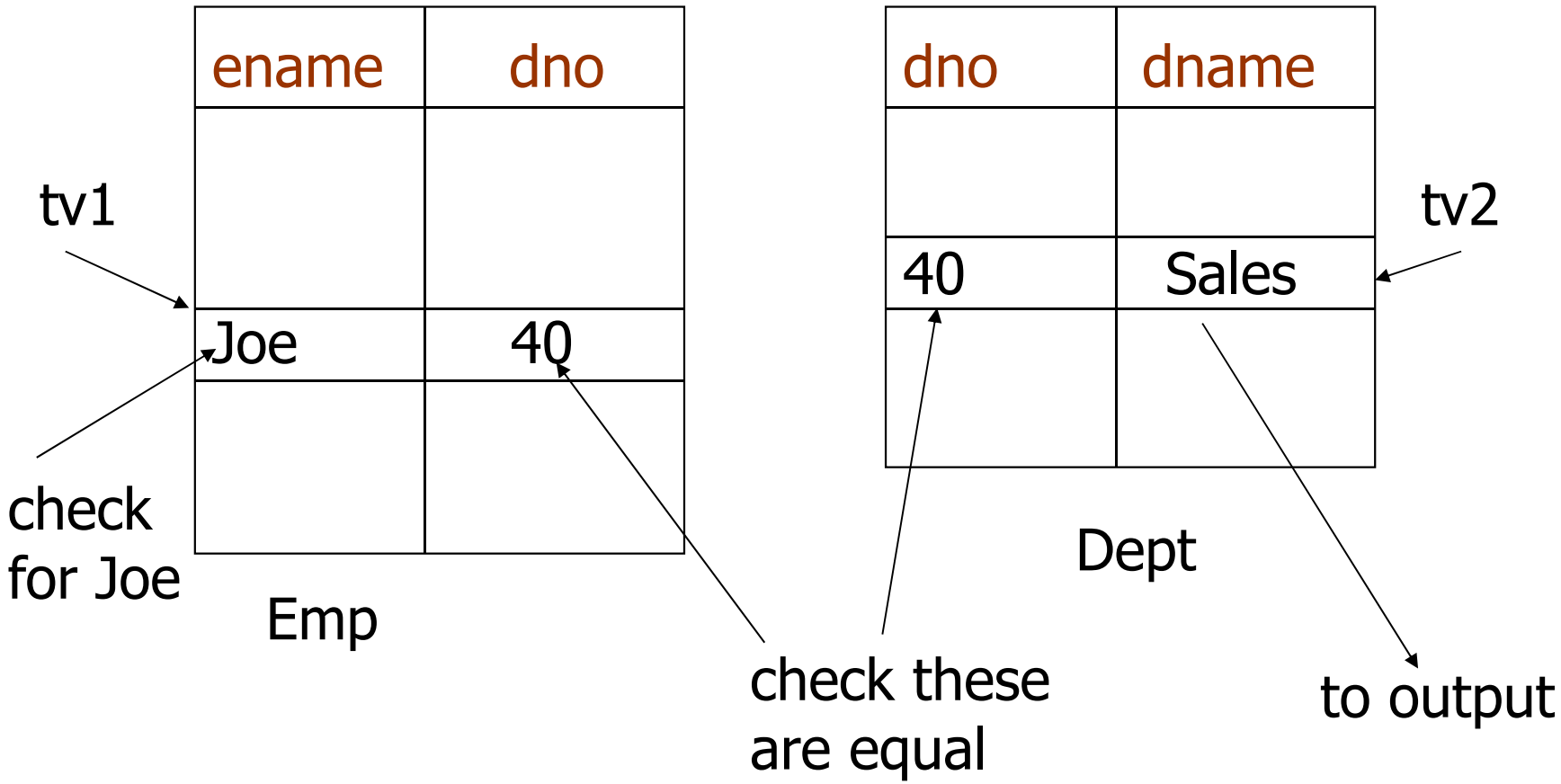
- Using tables Emp(ename, dno) and Dept(dno, dname), find the department name of employee *Joe*.

```
SELECT dname
FROM   Emp, Dept
WHERE  ename = 'Joe' AND
       Emp.dno = Dept.dno;
```

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.

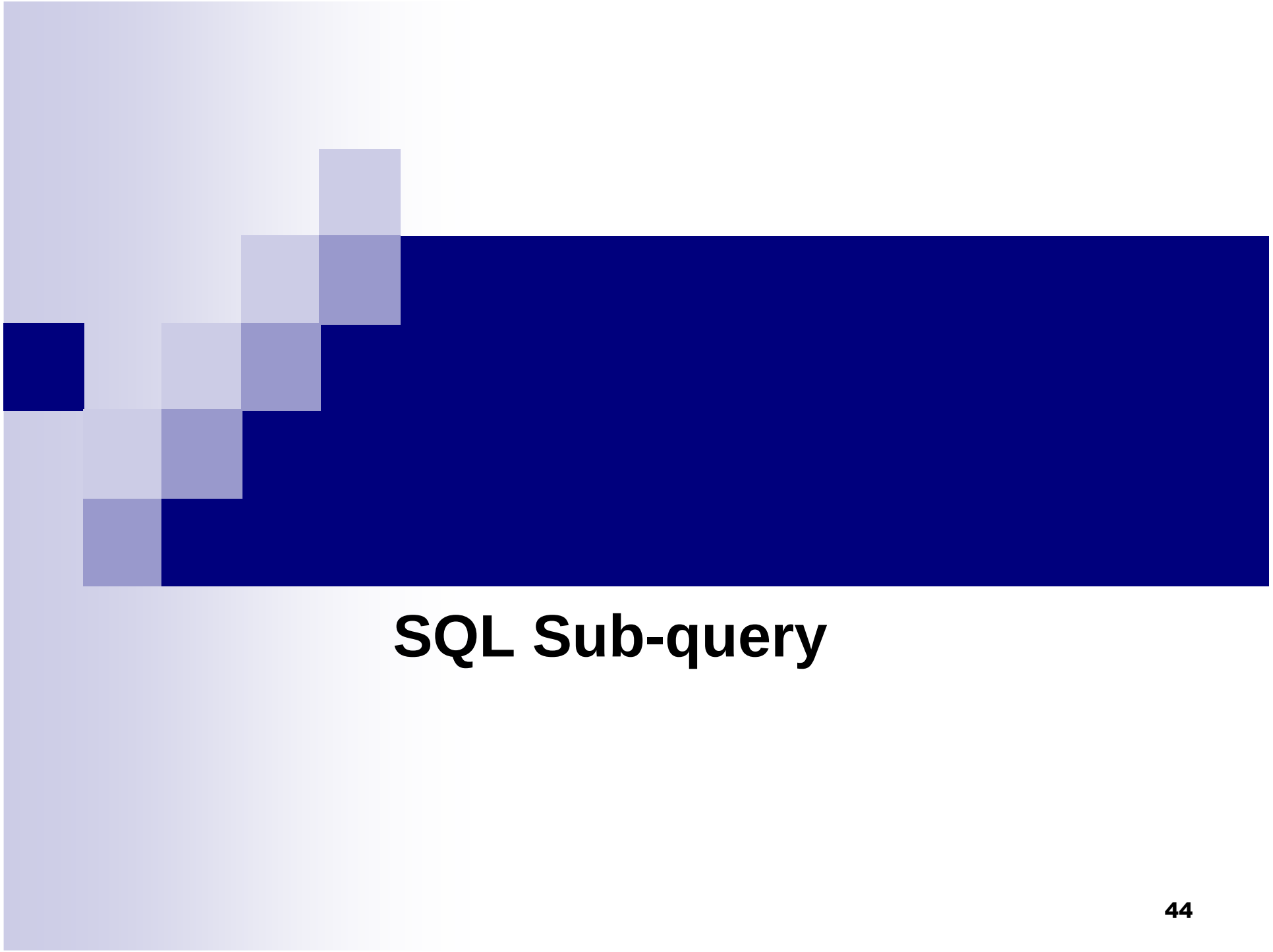
Example


ename
dno
tv1
Joe
40
check
for Joe
Emp
dno
dname
40
Sales
tv2
Dept
check these
are equal
to output

Example: Self-Join

- 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;
```



SQL Sub-query

Sub-queries

- A parenthesized SELECT-FROM-WHERE statement (*subquery*) can be used as a value in a number of places, including FROM and WHERE clauses.

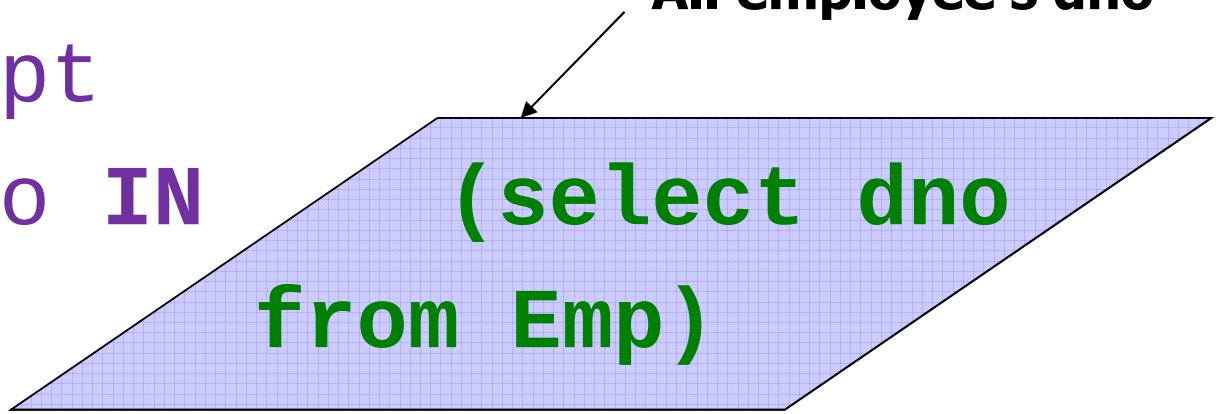
- **Example:** in place of a table in the WHERE clause, we can use a subquery and then query its result.

Example: Subquery in WHERE

- Find the department name that has at least one employee.

Emp(ename, dno) and Dept(dno, dname) relations

```
SELECT dname
FROM   Dept
WHERE  dno IN
```



Subqueries That Return One Tuple

- If a subquery is guaranteed to produce one tuple, then the subquery can be used as a value:
  - Usually, the tuple has one component.
  - A run-time error occurs if there is no tuple or more than one tuple.

Example: Single-Tuple Subquery

- Using tables Emp(ename, dno, sal) and Dept(dno, dname), find the name of employee who gets the highest salary

```
SELECT ename
FROM   Emp
WHERE  sal >= (select max(sal)
               from Emp);
```

The IN Operator

- <tuple> IN (<subquery>) is true if and only if the tuple is a member of the table produced by the subquery.
  - Opposite: <tuple> NOT IN (<subquery>).
- IN-expressions can appear in WHERE clauses.

IN is a Predicate About R's Tuples

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

Two 7's

One loop, over the tuples of R

a	b
1	7
3	4

R

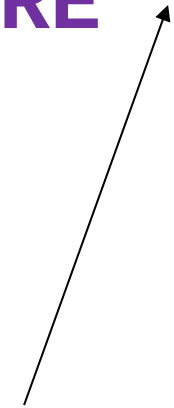
b	c
7	5
7	6

S

(1,7) satisfies the condition; 1 is output once.

This Query Pairs Tuples from R, S

```
SELECT a
FROM   R, S
WHERE  R.b = S.b;
```



Double loop, over
the tuples of R and S

a	b
1	7
3	4

R

b	c
7	5
7	6

S

(1,7) with (7,5) and (1,7) with (7,6) both satisfy the condition;
1 is output twice.

The Exists Operator

- EXISTS(<subquery>) is true if and only if the subquery result is not empty.

- Example: From Emp(ename, dno) , find those employees that are the only employee in their department.

Example: (Not) EXISTS

```
SELECT ename
FROM   Emp  e1
WHERE  NOT EXISTS (
    SELECT  *
    FROM    Emp
    WHERE   dno = e1.dno   AND
            ename <> e1.ename);
```

Notice scope rule: dno refers to closest nested FROM with a table having that attribute.

Notice the SQL "not equals" operator

Union, Intersection, and Difference

- Union, intersection, and difference of tables are expressed by the following forms, each involving subqueries:
  - (<subquery>) **UNION** (<subquery>)
  - (<subquery>) **INTERSECT** (<subquery>)
  - (<subquery>) **EXCEPT** (<subquery>)

DISTINCT

- From Sells(bar, beer, price), find all the different prices charged for beers:

```
SELECT DISTINCT  price
FROM   Sells;
```

- Notice that without DISTINCT, each price would be listed as many times as there were bar/beer pairs at that price.

Summary

- DDL - Create Schema
- Query
  - Select-From-Where Statements
  - Multi-relation Queries
  - Sub-queries

END