SQL: Data Manipulation Language
CSC 343
Fall 2019

MICHAEL LIUT (MICHAEL.LIUT@UTORONTO.CA)
DEPARTMENT OF MATHEMATICAL AND COMPUTATIONAL SCIENCES
UNIVERSITY OF TORONTO MISSISSAUGA

UNIVERSITY OF TORONTO MISSISSAUGA

UNIVERSITY OF TORONTO MISSISSAUGA

Why SQL?

SQL is a very-high-level language.

- Structured Query 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.

DBMSs determine the “best” method of executing a query.

- This is called “query optimization”.

UNIVERSITY OF TORONTO MISSISSAUGA

Database Schemas in SQL

SQL is primarily a query language, used for retrieving data from a database.

- Data Manipulation Language (DML)

But SQL also includes a data-definition component for describing database schemas.

- Data Definition Language (DDL)

Select-From-Where Statements

UNIVERSITY OF TORONTO MISSISSAUGA

SELECT → desired attribute(s)

FROM → one or more tables (i.e. entity-sets)

WHERE → condition about tuples of the tables

Recall our Example

UNIVERSITY OF TORONTO MISSISSAUGA

Our SQL queries will be based on the following database schema.

- Underlines indicates key attributes.

Beers (name, manf)
Bars (name, addr, license)
Drinkers (name, addr, phone)
Likes (drinker, beer)
Sells (bar, beer, price)
Frequents (drinker, bar)

Example

UNIVERSITY OF TORONTO MISSISSAUGA

Using Beers(name, manf)

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

Result of Query

name
Bud
Bud Light
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

UNIVERSITY OF TORONTO MISSISSAUGA

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 - General

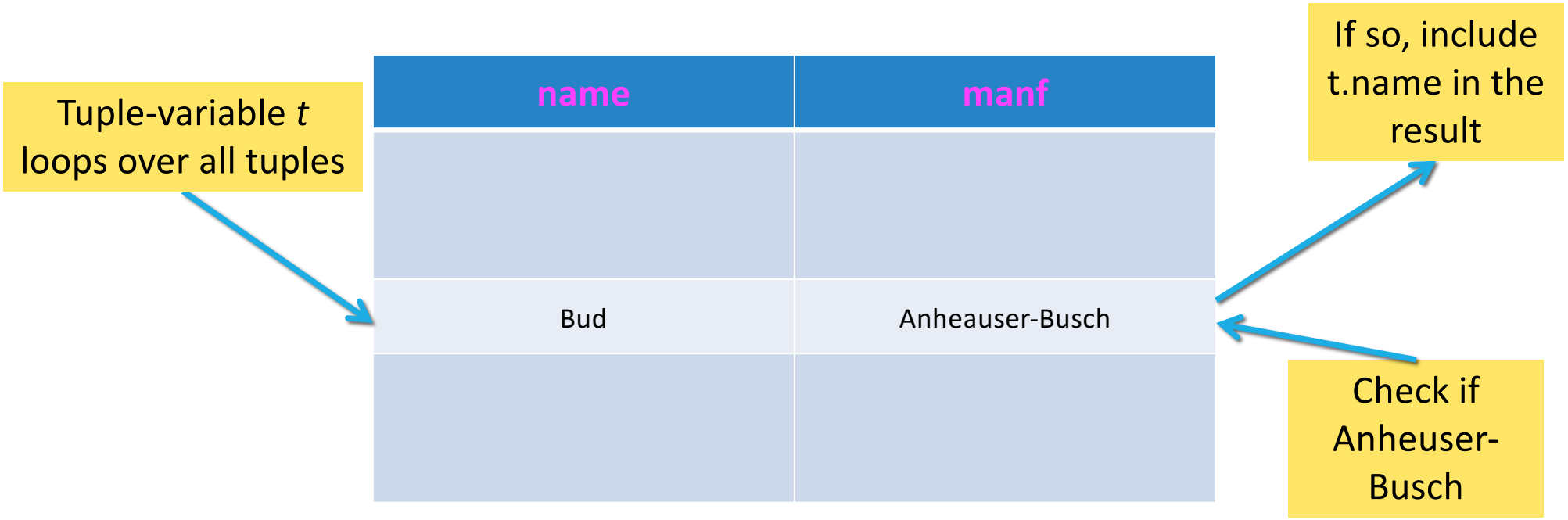
UNIVERSITY OF TORONTO MISSISSAUGA

Think of a *tuple variable* visiting each tuple of the relation mentioned in FROM.

Check if the tuple assigned to the tuple variable satisfies the WHERE clause.

If so, compute the attributes or expressions of the SELECT clause using the components of this tuple.

Operational Semantics



Example

UNIVERSITY OF TORONTO MISSISSAUGA

What beers are made by Anheuser-Busch?

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

OR

```
SELECT  t.name
FROM   Beers  t
WHERE  t.manf  =  'Anheuser-Busch';
```

NOTE: these two queries are identical.

Asterisks (*) in Select Clauses

UNIVERSITY OF TORONTO MISSISSAUGA

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';
```

UNIVERSITY OF TORONTO MISSISSAUGA

Result of Query

name	manf
Bud	Anheuser-Busch
Bud Light	Anheuser-Busch
Michelob	Anheuser-Busch
…	…

The result now has each of the attributes of Beers.

Renaming Attributes

UNIVERSITY OF TORONTO MISSISSAUGA

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

UNIVERSITY OF TORONTO MISSISSAUGA

beer	manf
Bud	Anheuser-Busch
Bud Light	Anheuser-Busch
Michelob	Anheuser-Busch
…	…

The result now has each of the attributes of Beers.

UNIVERSITY OF TORONTO MISSISSAUGA

Expressions in SELECT Clauses

Any valid expression can appear as an element of a SELECT clause.

EXAMPLE: Using Sells(bar, beer, price)

```
SELECT   bar, beer, price*80 AS priceInJPY
FROM    Sells;
```

UNIVERSITY OF TORONTO MISSISSAUGA

Result of Query

bar	beer	priceInJPY
Joe's	Bud	285
Sue's	Miller	342
John's	Michelob	532
…	…	…

The result now has each of the attributes of Beers.

UNIVERSITY OF TORONTO MISSISSAUGA

Example: Constants as Expressions

Using Likes(drinker, beer)

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

Result of Query

UNIVERSITY OF TORONTO MISSISSAUGA

drinker	whoLikesBud
George	likes Bud
Frank	likes Bud
Jenny	likes Bud
…	…

UNIVERSITY OF TORONTO MISSISSAUGA

Complex Conditions in WHERE clause

Boolean operators: **AND**, **OR**, **NOT**.

Comparison operators: **=**, **<>**, **<**, **>**, **<=**, **>=**.

Example: Complex Condition

UNIVERSITY OF TORONTO MISSISSAUGA

Using Sells(bar, beer, price), find the price that 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 quotes string:

- % = “any string”;
- __ = “any character”.

Example: Like

UNIVERSITY OF TORONTO MISSISSAUGA

Using Drinkers(name, addr, phone)

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

NULL Values

UNIVERSITY OF TORONTO MISSISSAUGA

Tuples in SQL relations can have NULL as a value for one or more components.

The meaning is dependent on the context. In general, there are two common cases:

1. *Missing Values*: e.g., we know Joe's Bar has some address, but we don't know what it is.
2. *Inapplicable*: e.g., the value of attribute spouse for an unmarried person.

Comparing NULL's to Values

UNIVERSITY OF TORONTO MISSISSAUGA

The logic of a conditions in SQL are 3 pronged:

1. **TRUE**
2. **FALSE**
3. **UNKNOWN**
   - Comparing any value (including NULL) with NULL yields UNKNOWN.

A tuple is in a query answer iff the **WHERE** clause is **TRUE** (not **FALSE** or **UNKNOWN**).

RECALL

EXAMPLE: Using Sells(bar, beer, price)

```
SELECT   bar, beer, price*80 AS priceInJPY
FROM    Sells;
```

Result of Query

UNIVERSITY OF TORONTO MISSISSAUGA

bar	beer	priceInJPY
Joe's	Bud	285
Sue's	Miller	342
John's	Michelob	532
…	…	…

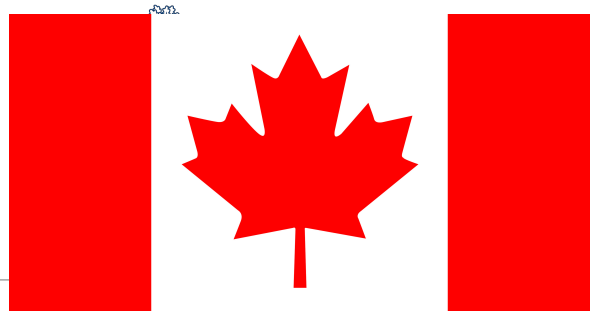
UNIVERSITY OF TORONTO MISSISSAUGA

String Functions in SELECT Clauses

EXAMPLE: Using Sells(bar, beer, price)

```
SELECT   bar,  INSERT(beer, 1, 0, 'EH ') AS 'Canadian beer',  price
FROM    Sells;
```

Result of Query



bar	Canadian beer	price
Joe's	EH Bud	4
Sue's	EH Miller	5
John's	EH Michelob	7
…	…	…

The result reflects the renamed beers to reflect their inner Canadian.

Three-Valued Logic

UNIVERSITY OF TORONTO MISSISSAUGA

To understand how **AND**, **OR**, and **NOT** work in 3- valued logic

For **TRUE** results:

- **OR:** at least one operand must be **TRUE**.
- **AND:** both operands must be **TRUE**.
- **NOT**: operand must be **FALSE**.

Three-Valued Logic

UNIVERSITY OF TORONTO MISSISSAUGA

To understand how **AND**, **OR**, and **NOT** work in 3- valued logic

For **FALSE** results:

- **OR:** both operands must be **FALSE**.
- **AND:** at least one operand must be **FALSE**.
- **NOT**: operand must be **TRUE**.

Three-Valued Logic

UNIVERSITY OF TORONTO MISSISSAUGA

To understand how **AND**, **OR**, and **NOT** work in 3- valued logic

Otherwise

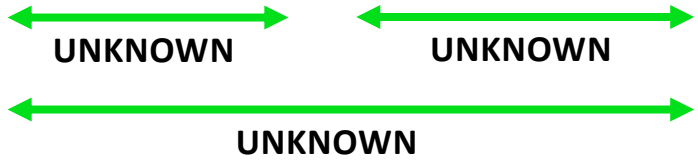
- The result is **UNKNOWN**.

Example

From the following Sells relation:

bar	beer	price
Joe's Bar	Bud	NULL
…	…	…

```
SELECT   bar
FROM     Sells
WHERE  price < 2.00  OR  price >= 5.00;
```



Multi-Relation Queries

UNIVERSITY OF TORONTO MISSISSAUGA

Usually a combination of data from multiple (i.e. more than one) relation.

- More often than not, these are interesting queries.

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>”.

UNIVERSITY OF TORONTO MISSISSAUGA

Example: Joining Two Relations

Using relations: Likes(drinker, beer) and Frequents(drinker, bar), find the beers likes 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;
```

Example: Joining Two Relations

UNIVERSITY OF TORONTO MISSISSAUGA

Alternatively, previously shown on slide #11, we can use explicit (named) tuple variables.

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

Formal Semantics

UNIVERSITY OF TORONTO MISSISSAUGA

Almost the same for single-relation queries:

- Start with the product of all the relations in the **FROM** clause.
- Apply the selection condition from the **WHERE** clause.
- Project onto the list of attributes and expressions in the **SELECT** clause.

Operational Semantics

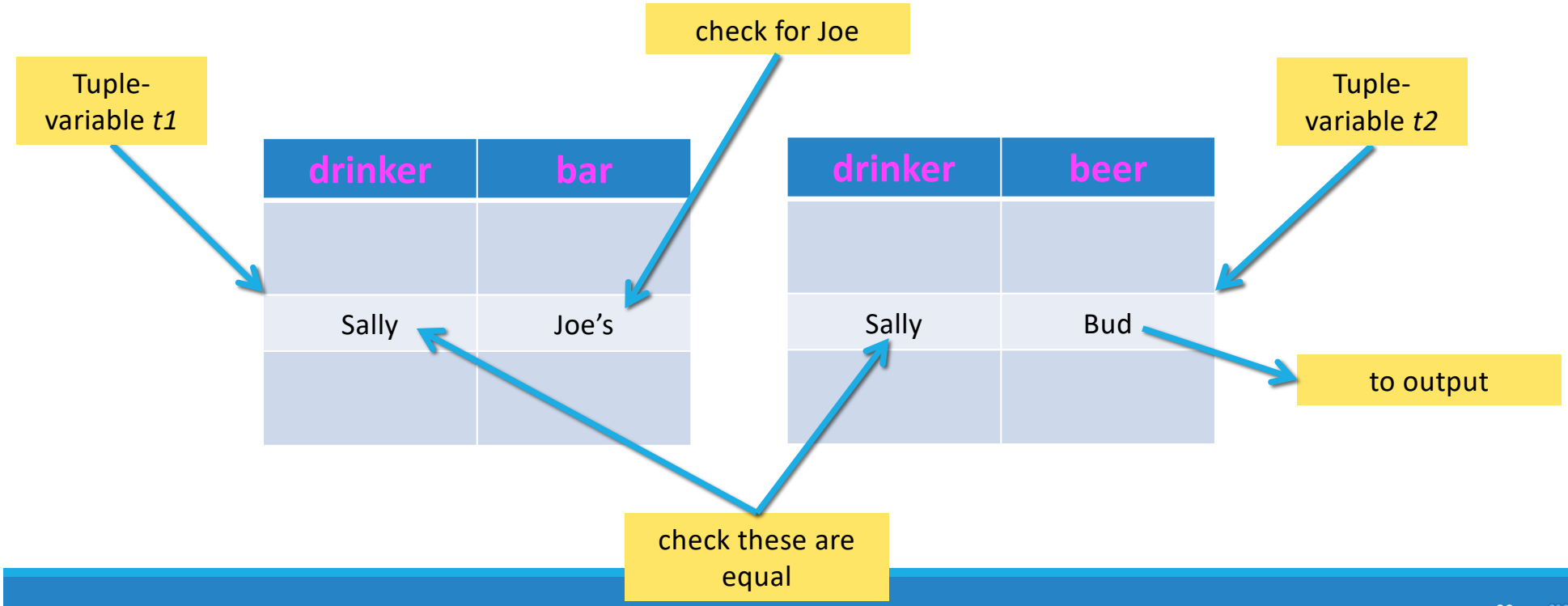
UNIVERSITY OF TORONTO MISSISSAUGA

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.

Operational Semantics



UNIVERSITY OF TORONTO MISSISSAUGA

Explicit Tuple-Variables

Sometimes a query needs to use two copies of the same relation.

Copies are distinguished 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: Self-Join

From Beers(name, manf), find all pairs of beers by the same manufacturer.

- Do not produce pairs like (Bud, Bud).
- Do not produce the same pairs twice (Bud, Miller) and (Miller, Bud).

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

Sub-queries

UNIVERSITY OF TORONTO MISSISSAUGA

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 relation in the FROM clause, we can use a subquery and then query its result.

- To do this we must use a tuple-variable to name tuples of the result.

Example: Sub-query in FROM

Find the beers liked by at least one person who frequents Joe's Bar.

```
SELECT   beer
FROM     Likes, (SELECT   drinker
                 FROM     Frequents
                 WHERE  bar = 'Joe``s Bar') JD
WHERE  Likes.drinkers = JD.drinker;
```

Drinkers who frequent Joe's Bar

UNIVERSITY OF TORONTO MISSISSAUGA

Sub-queries Often Obscure Queries

Find the beers liked by at least one person who frequents Joe's Bar.

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

Simple Join Query

UNIVERSITY OF TORONTO MISSISSAUGA

Sub-Queries That Return One Tuple

If a sub-query is guaranteed to produce one tuple, then the sub-query can be used as a value.

- Usually, the tuple has one component.
- Remember SQL's 3-valued logic.

Example: Single-Tuple Sub-query

UNIVERSITY OF TORONTO MISSISSAUGA

Using Sells(bar, beer, price), find the bars that serve Miller for the same price Joe charges for Bud.

Two queries would work:

- Find the price Joe charges for Bud.
- Find the bars that serve Miller at that price.

Query + Sub-Query Solution

- Find the price Joe charges for Bud.
- Find the bars that serve Miller at that price.

Sells(bar, beer, price)

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

The price at which Joe sells Bud

Query + Sub-Query Solution

- Find the price Joe charges for Bud.
- Find the bars that serve Miller at that price.

Sells(bar, beer, price)

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

The price at which Joe sells Bud

Questions:

1. What if the price of Bud is NULL?
2. What if the sub-query returns multiple values?

Temporary Tables

UNIVERSITY OF TORONTO MISSISSAUGA

Yes, they do exists! MySQL allows them!

Note: a declared temporary table must be declared!

- CREATE TEMPORARY TABLE <temp_table_name> …

Temporary tables exist in:

- WHERE/FROM clauses, WITH statements, and survive for the duration a single session.
- Single session must be created in a *user temporary space*.

A few useful links for further investigation:

- https://dev.mysql.com/doc/refman/5.7/en/create-temporary-table.html

UNIVERSITY OF TORONTO MISSISSAUGA

Recap: Conditions in WHERE Clause

Boolean operators: **AND**, **OR**, **NOT**.

Comparisons: =, <>, <, >, <=, >=.

LIKE operator.

SQL includes a **BETWEEN** comparison operator too.

- Let's see an example!

UNIVERSITY OF TORONTO MISSISSAUGA

Example: Between

Find the names of all Instructors with salary **BETWEEN** $90,000 and $100,000.

- i.e. >= $90,000 and <= $100,000.

```
SELECT   name
FROM     Instructor
WHERE  salary  BETWEEN  90000  AND  100000;
```

The Operator: ANY

UNIVERSITY OF TORONTO MISSISSAUGA

X = ANY(<sub-query>) is a Boolean condition that is TRUE iff *x* equals at least one tuple in the subquery result.

- = could be any comparison operator.

Example: x >= ANY(<subquery>) means x is not the uniquely smallest tuple produced by the sub-query.

- NOTE: tuples must have one component only.

The Operator: ALL

UNIVERSITY OF TORONTO MISSISSAUGA

x <> ALL (<subquery>) is true iff for every tuple *t* in the relation, *x* is not equal to *t*.

<> can be any comparison operator.

Example: x >= ALL (<subquery>) means there is no tuple larger than *x* in the sub-query result.

Example: ALL

From Sells(bar, beer, price), find the beer(s) sold at the highest price.

```
SELECT   beer
FROM    Sells
WHERE  price >= ALL(
        SELECT   price
        FROM    Sells
        );
```

The price from the outer Sells **must not be** less than any price.

The Operator: IN

<value> IN (<subquery>) is true iff the <value> is a member of the relation produced by the sub-query.

- Opposite: <value> NOT IN (<subquery>)

IN-expression can appear in the WHERE clauses.

WHERE <column> IN (value1, value2, …, valueN)

“IN” is Concise


```
SELECT   *
FROM    Cartoons
WHERE  LastName
        IN  (‘Jetsons’, ‘Smurfs’, ‘Flinstones’);
```

VS.

```
SELECT   *
FROM    Cartoons
WHERE  LastName = ‘Jetsons’
        OR LastName = ‘Smurfs’
        OR LastName = ‘Flinstones’;
```

Example: IN

Using Beers(name, manf) and Likes(drinker, beer), find the name and manufacturer of each beer that Fred likes.

```
SELECT   *
FROM    Beers
WHERE  name IN  (SELECT   beer
                 FROM    Likes
                 WHERE  drinker = 'Fred');
```

The set of beers that Fred likes.

Another Example: Operator “IN” and “NOT IN”

UNIVERSITY OF TORONTO MISSISSAUGA

Using IN and NOT IN operators with a Multiple-Row Sub-query:

- http://www.w3resource.com/sql/subqueries/multiplee-row-column-subqueries.php

IN vs. Join


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

VS.

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

Note: IN and JOIN are different queries that can yield different results.
Unless S.b is unique!

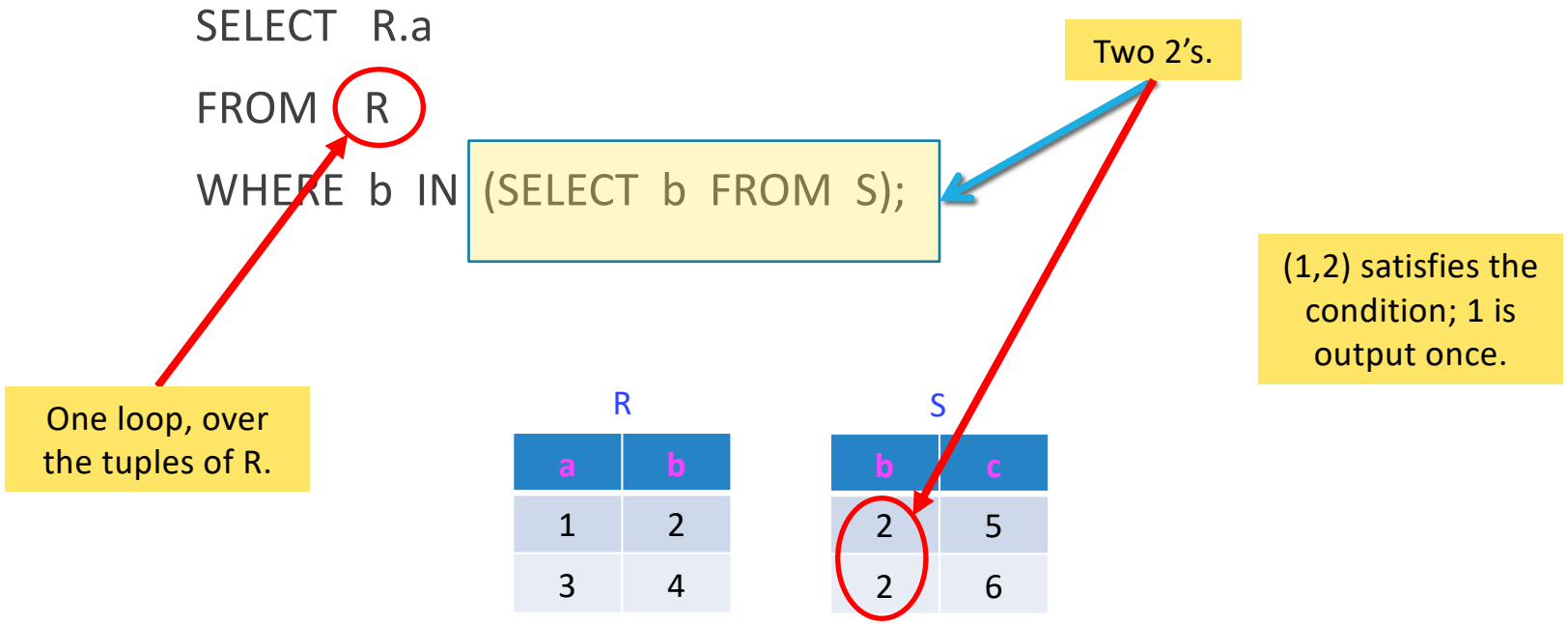
IN vs. Join

Equivalent to:

```
SELECT   R.a
FROM    R,  S
JOIN    (SELECT  DISTINCT  b
          FROM   S
         )
ON       R.b  =  S.b;
```

Performance Note: if the joining column is not UNIQUE then IN is faster than JOIN on DISTINCT.

IN is a Predicate About R's Tuples



This Query Pairs Tuples from R, S


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

Double loop, over the tuples of R and S.

(1,2) with (2,5) and (1,2) with (2,6) both satisfy the condition; 1 is output twice.

R

a	b
1	2
3	4

S

b	c
2	5
2	6

Recall: Original Query

```
SELECT   bar
FROM    Sells
WHERE   beer = 'Miller' AND
        price = (SELECT   price
                 FROM    Sells
                 WHERE  beer = 'Bud');
```

Option 1: use IN

Option 2: use = ANY()

RECAP

UNIVERSITY OF TORONTO MISSISSAUGA

The IN() operator is equivalent to = ANY().

For ANY(), you can use other comparison operators such as:

- >, <, etc… but this is NOT applicable for the IN() operator.
  - The ability to use these operators can give you more control in your query depending on your desired output/result.

UNIVERSITY OF TORONTO MISSISSAUGA

RECAP

NOTE: the <> ANY operator differs from NOT IN:

1. <> ANY means != (a.k.a. does not equal) any…
   - e.g. <> ANY means != a **or** != b **or** != c …

2. NOT IN means != (a.k.a. does not equal) any…
   - e.g. != a **and** != b **and** != c …

3. <> ALL means the same as NOT IN.

Example: = ANY()

UNIVERSITY OF TORONTO MISSISSAUGA

Sells

bar	beer	price
Jane	Miller	3.00
Joe	Miller	4.00
Joe	Bud	3.00
Jack	Bud	4.00
Tom	Miller	4.50

Result

bar
Jane
Joe

```
SELECT   bar
FROM    Sells
WHERE  beer = 'Miller'  AND  price =
          ANY(SELECT   price
                FROM    Sells
                WHERE  beer = 'Bud');
```

The Operator: Exists

UNIVERSITY OF TORONTO MISSISSAUGA

Exists (<subquery>) is true iff the sub-query is **not empty**.

Example: From Beers(name, manf), find the beers that are unique (i.e. only) beer made by their respective manufacturer.

Example: Exists

```
SELECT    name
FROM      Beers  b1
WHERE     NOT EXISTS (
            SELECT    *
            FROM      Beers
            WHERE     manf = b1.manf  AND  name <> b1.name
          );
```

Notice the scope rule:
Manf refers to the closest nested FROM with a relation having that attribute.
NOTE: Some DBMSs consider this to be ambiguous.

Set of beers with the same manf as b1, but not the same beer.

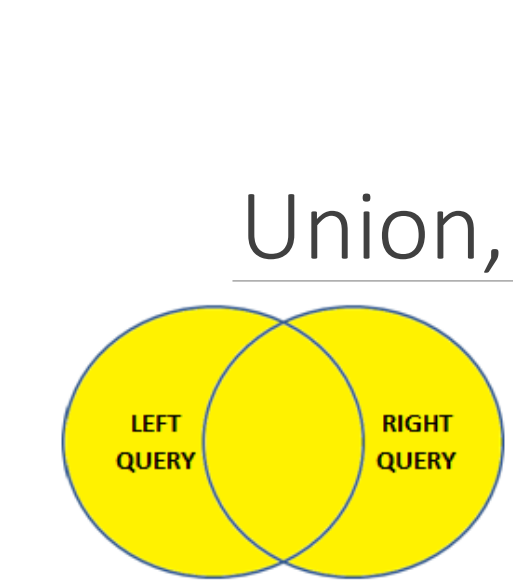
Notice the SQL "not equals" operator.

UNIVERSITY OF TORONTO MISSISSAUGA

Union, Intersection, and Difference

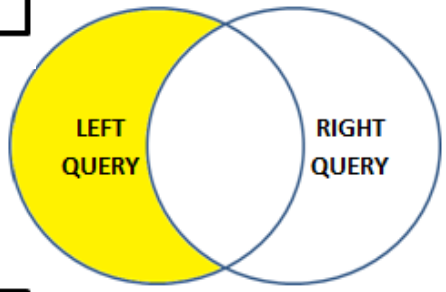
Union, intersection, and difference of relations are expressed by the following forms, each involving sub-queries:

- (<subquery>) UNION (<subquery>)
- (<subquery>) INTERSECTION (<subquery>)
- (<subquery>) EXCEPT (<subquery>)

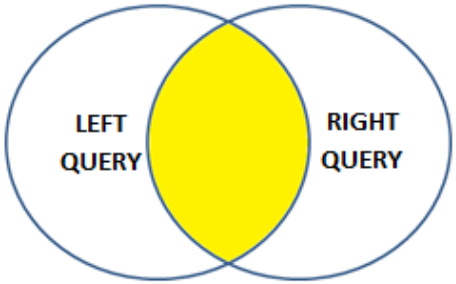


Union, Intersection, and Difference

UNION operator returns all the unique rows from both the left and the right query. UNION ALL includes the duplicates as well



EXCEPT operator returns unique rows from the left query that aren't in the right query's results



INTERSECT operator retrieves the common unique rows from both the left and the right query

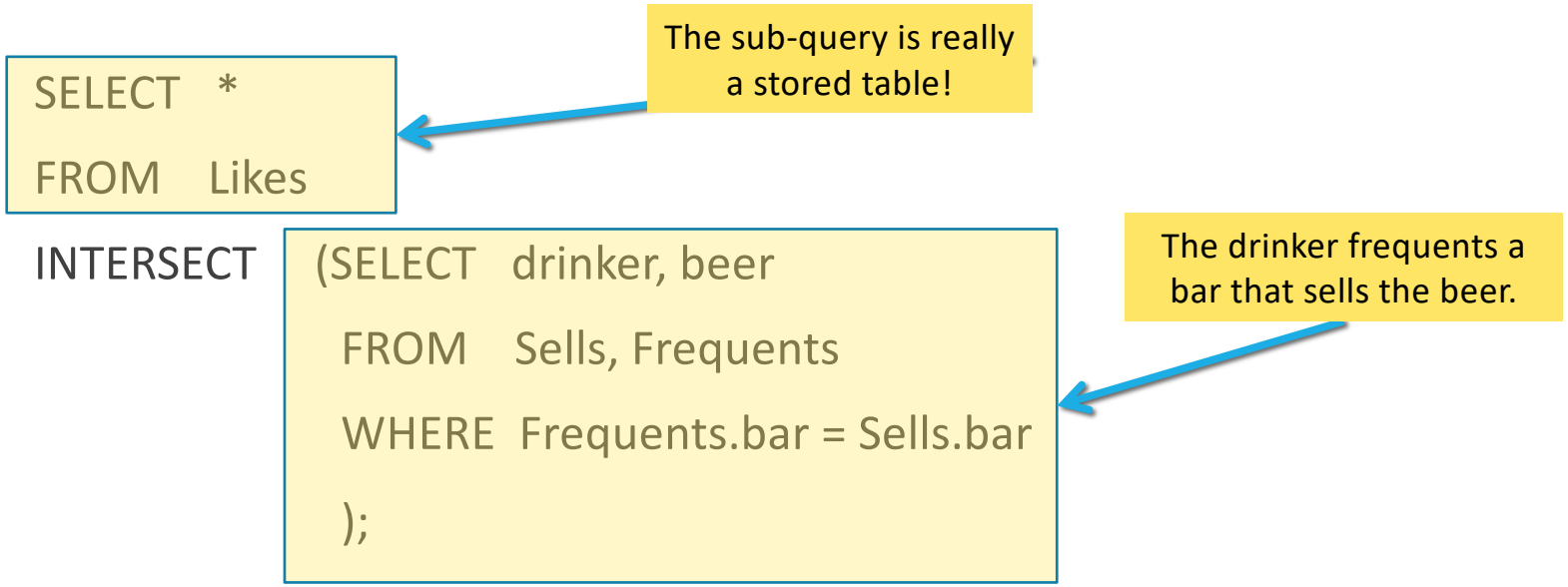
Example: Intersection

UNIVERSITY OF TORONTO MISSISSAUGA

Using Likes(drinker, beer) , Sells(bar, beer, price), and Frequents(drinker, bar), find the drinkers and beers such that:

1. The drinker likes the beer, and
2. The drinker frequents at least one bar that sells the beer.

Example: Intersection – Solution



Ordering the Display of Tuples

UNIVERSITY OF TORONTO MISSISSAUGA

List in alphabetical order the names of all instructors

SELECT attribute

FROM Table

ORDER BY attribute <asc/desc>

We may specify desc for descending order or asc for ascending order, for each attribute.

Note: Ascending order is the default.

Example: Order By

UNIVERSITY OF TORONTO MISSISSAUGA

List the names of all instructors in descending alphabetical.

```
SELECT name
FROM Instructor
ORDER BY name;
```

vs.

```
SELECT name
FROM Instructor
ORDER BY name desc;
```

UNIVERSITY OF TORONTO MISSISSAUGA

Bag Semantics

A *bag* (or *multi-set*) is like a set, but an element may appear more than once.

Example: {1,2,1,3} is a bag.

Example: {1,2,3} is also a bag that happens to be a set.

Bag (multi-Set Semantics)

UNIVERSITY OF TORONTO MISSISSAUGA

SQL primarily uses bag semantics.

The SELECT-FROM-WHERE statement uses bag semantics.

- Originally for efficiency reasons.

The default for union, intersection, and difference is set semantics.

- That is, duplicates are eliminated as the operation is applied.

Motivation: Efficiency

UNIVERSITY OF TORONTO MISSISSAUGA

When doing projection, it is easier to avoid eliminating duplicates.

- Work on one tuple-at-a-time.

For intersection or difference, it is most efficient to sort the relations first.

- At that point you may as well eliminate the duplicates anyway.

Control Duplicate Elimination

UNIVERSITY OF TORONTO MISSISSAUGA

Force the result to be a set by SELECT DISTINCT…

Force the results to be a bag (i.e. don't eliminate duplicates) by the ALL operator

i.e. … UNION ALL …

Example: DISTINCT

UNIVERSITY OF TORONTO MISSISSAUGA

From Sells(bar, beer, price), find all the different prices charges 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.

Example: ALL

UNIVERSITY OF TORONTO MISSISSAUGA

Using the relations: Likes(drinker, beer) and Frequents(drinker, bar):

- List drinkers who frequent more bars than they like beers, and do so as many times as the difference of those counts.

```
(SELECT  drinker  FROM  Frequents)
        EXCEPT ALL
(SELECT  drinker  FROM  Likes);
```

Let's Add Some HUMOUR!

A SQL query walks into a bar, and approaches two tables. The SQL query proceeds to ask "can I join you?"

The Query Containment Problem: Set Semantics *vs.* Bag Semantics

UNIVERSITY OF TORONTO MISSISSAUGA

First some background info…

RDBMS = Relational Database Management System.

- Created by E.F. Codd at IBM's San Jose Research Lab.
- Used to house everything from financial records to manufacturing and logistical info.
- This is based on the Relational Model (RM).

A conjunctive query is a restricted form first-order query.

- They have many desirable properties that larger classes of queries do not share.

e.g. a large class of query is relational algebra.

The Query Containment Problem: Set Semantics *vs.* Bag Semantics

UNIVERSITY OF TORONTO MISSISSAUGA

Relational Algebra, also created by E.F. Codd while at IBM, is a well-founded semantics for modelling the data stored in relational databases, and defining queries.

This theory is used to provide a theoretical foundation for relational databases, particularly query languages (especially SQL).

The Query Containment Problem: Set Semantics *vs.* Bag Semantics

UNIVERSITY OF TORONTO MISSISSAUGA

Keynote talk by: Phokion Kolaitis from the University of California, Santa Cruz.

Presented in 2013 at: 7th Annual Mendelzon International Workshop on Foundation of Data Management.

Abstract. Query containment is a fundamental algorithmic task in database query processing and optimization. Under set semantics, the query-containment problem for conjunctive queries has long been known to be NP-complete. SQL queries, however, are typically evaluated under bag semantics and return multisets as answers, since duplicates are not eliminated unless explicitly specified. The exact complexity of the query-containment problem for conjunctive queries under bag semantics has been an outstanding and rather poorly understood open problem for twenty years. In fact, to this date, it is not even known whether conjunctive-query containment under bag semantics is decidable. The goal of this talk is to draw attention to this fascinating problem by presenting a comprehensive overview of old and not-so-old results about the complexity of the query-containment problem for conjunctive queries and their variants, under both set semantics and bag semantics.

The Query Containment Problem: Set Semantics *vs.* Bag Semantics

UNIVERSITY OF TORONTO MISSISSAUGA

Link to conference paper: http://ceur-ws.org/Vol-1087/keynote2slides.pdf

Link to Hilbert's 23 problems: https://en.wikipedia.org/wiki/Hilbert%27s_problems

UNIVERSITY OF TORONTO MISSISSAUGA

Database Modifications

A modification command does not return a result (as a query does), but changes the database in some way.

Three types of modifications are:

1. Insert → inserting a tuple or tuples.
2. Delete → deleting a tuple or tuples.
3. Update → updating the value(s) of an existing tuple or tuples.

Insertion

To insert a single tuple:

INSERT INTO <relation>
VALUES (<list_of_values>);

Example: add to Likes(drinker, beer)

INSERT INTO Likes
VALUES ('Sally', 'Bud');

Specifying Attributes in INSERT

UNIVERSITY OF TORONTO MISSISSAUGA

We may add to the relation name a list of attributes.

Two reasons to do so:

1. We forget the standard order of attributes for the relation.
2. We don't have values for all attributes, and we want the system to fill in missing components with NULL of a default value.

Example: Specifying Insert Attributes

UNIVERSITY OF TORONTO MISSISSAUGA

Another way to add the statement Sally likes Bud to Likes(drinker, beer).

```
INSERT INTO Likes (beer, drinker)
VALUES ('Bud', 'Sally');
```

Adding Default Values

UNIVERSITY OF TORONTO MISSISSAUGA

In a CREATE TABLE statement, we can follow an attribute by DEFAULT and a value.

When an inserted tuple has no value for that attribute, the default value will be used.

Example: Default Values

```
CREATE TABLE  Drinkers  (
        name CHAR(30)  PRIMARY KEY,
        addr  CHAR(50)  DEFAULT  '123 Main St.',
        phone  CHAR(16)
);
```

Example: Default Values Continued

UNIVERSITY OF TORONTO MISSISSAUGA

```
INSERT INTO Drinkers (name)
VALUES ('Sally');
```

Result:

name	address	phone
Sally	123 Main St.	NULL

Insert Many Tuples

UNIVERSITY OF TORONTO MISSISSAUGA

We may insert the entire result of a query into a relation, using the form:

INSERT INTO <relation>

(<subquery>);

Example: Insert a SubQuery

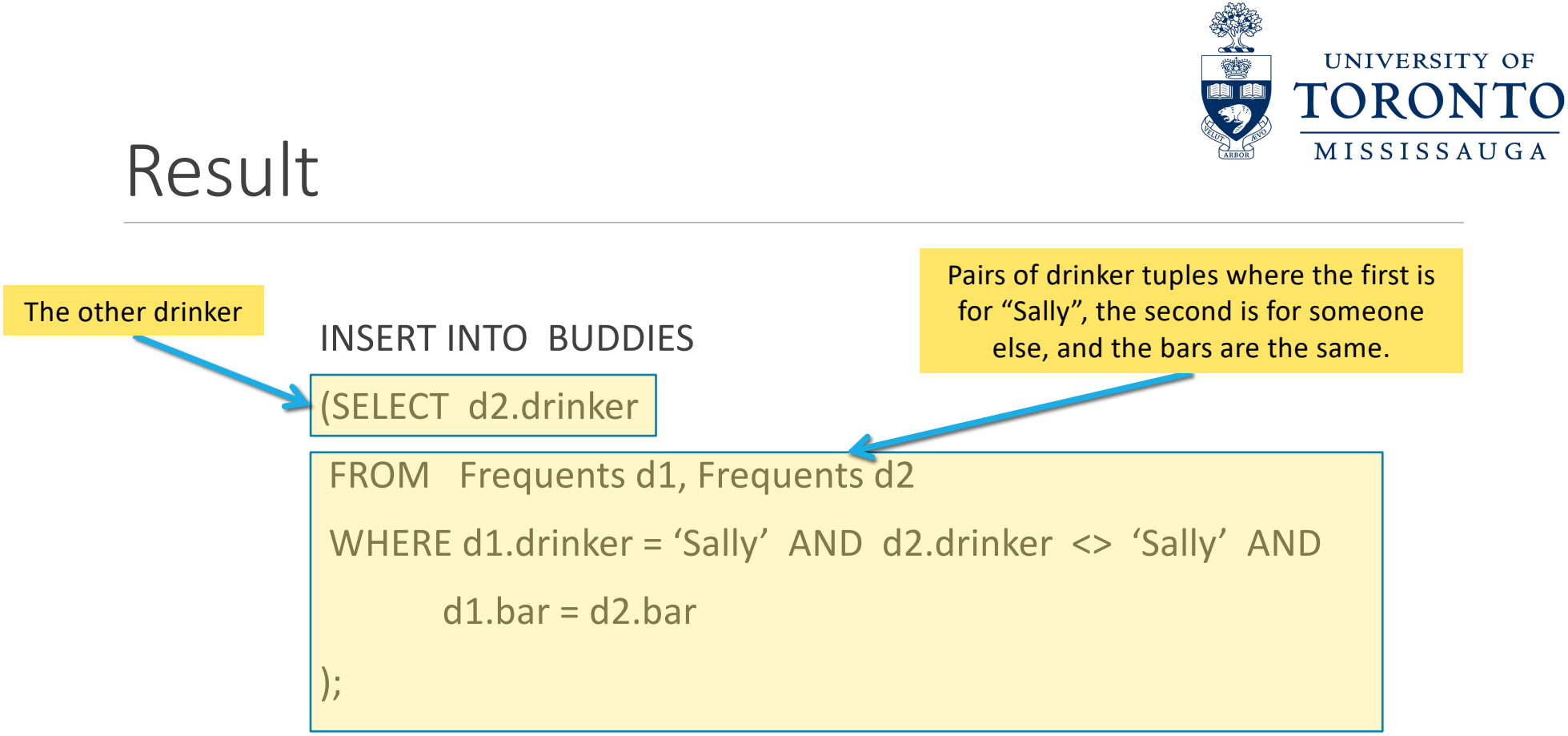
UNIVERSITY OF TORONTO MISSISSAUGA

Using Frequents(drinker, beer), enter into the new relation Buddies(name) all of Sally's "potential buddies".

i.e. The drinkers who frequent at least one bar that Sally also frequents.

```
INSERT INTO Buddies
(SELECT);
```

Result


The other drinker
Pairs of drinker tuples where the first is for "Sally", the second is for someone else, and the bars are the same.
INSERT INTO BUDDIES
(SELECT d2.drinker
FROM Frequents d1, Frequents d2
WHERE d1.drinker = 'Sally' AND d2.drinker <> 'Sally' AND
d1.bar = d2.bar
);

Deletion

To delete tuples satisfying a condition from some relation:

```
DELETE FROM <relation>
WHERE <condition>;
```

Example: Deletion

UNIVERSITY OF TORONTO MISSISSAUGA

Delete from Likes(drinker, beer) the tuple stating that Sally likes Bud.

```
DELETE  FROM  Likes
WHERE  drinker = 'Sally'  AND  beer = 'Bud';
```

Example: Delete all Tuples

UNIVERSITY OF TORONTO MISSISSAUGA

Alternatively, you can make the relation Likes empty.

```
DELETE FROM Likes;
```

Note: No WHERE clause is required.

Example: Delete Some Tuples

Delete from Beers(name, manf) all beers for which there is another beer by the same manufacturer.

```
DELETE   FROM  Beers b
WHERE  EXISTS(
            SELECT   name
            FROM    Beers
            WHERE  manf = b.manf  AND  name <>  b.name
           );
```

Beers with the same manufacturer and a different name from the name of the beer represented by tuple *b*.

Semantics of Deletion

UNIVERSITY OF TORONTO MISSISSAUGA

Suppose Anheuser-Busch makes only Bud and Bud Lite.

Suppose we come to the tuple *b* for Bud first.

The sub-query is non-empty, because of the Bud Lite tuple, so we delete Bud.

Now, when *b* is the tuple for Bud Lite, do we delete that tuple too?

We do delete Bud Lite as well!
Let's see how!

Semantics of Deletion

UNIVERSITY OF TORONTO MISSISSAUGA

Answer: Yes, we do delete Bud Lite as well.

The deletion proceeds in two stages:

1. Mark all tuples for which the WHERE condition is satisfied.
2. Delete the marked tuples.

Updates

UNIVERSITY OF TORONTO MISSISSAUGA

To change certain attributes in certain tuples of a relation:

UPDATE <relation>

SET <list of attribute assignments>

WHERE <condition on tuples>;

Example: Updates

UNIVERSITY OF TORONTO MISSISSAUGA

Change drinker Fred's phone number to 555-1234.

```
UPDATE  Drinkers
SET  phone = '555-1234'
WHERE  name = 'Fred';
```

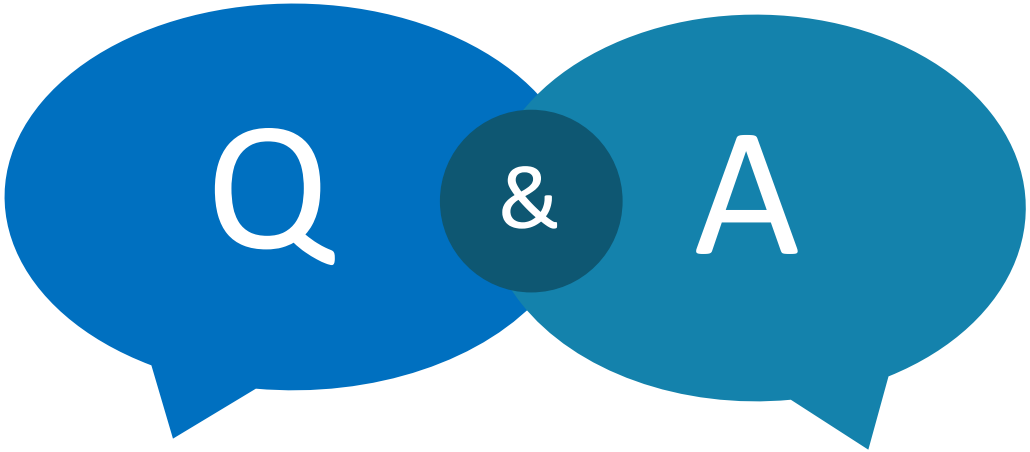
Example: Update Several Tuples

UNIVERSITY OF TORONTO MISSISSAUGA

Make $4 the maximum price for beer.

```
UPDATE  Sells
SET  price = 4.00
WHERE  price > 4.00;
```

Questions?



THANKS FOR LISTENING

I'LL BE ANSWERING QUESTIONS NOW

UNIVERSITY OF TORONTO MISSISSAUGA

Citations, Images and Resources

Database Management Systems (3rd Ed.), Ramakrishnan & Gehrke

Some content is based off the slides of Dr. Fei Chiang - http://www.cas.mcmaster.ca/~fchiang/

http://csharpcorner.mindcrackerinc.netdna-cdn.com/UploadFile/BlogImages/06112016031910AM/sql.png

http://www.acoa-apeca.gc.ca/eng/investment/InvestmentHome/PublishingImages/Canada_flag.jpg

http://4.bp.blogspot.com/-lYiQHrCXpvE/Ve3o5LdGsII/AAAAAAAAd88/Tl16JTc17Ho/s1600/difference%2Bbetween%2Bunion%2Bintersect%2Band%2Bexcept%2Bin%2Bsql%2Bserver.png

http://www.ibm.com/support/knowledgecenter/SSEPEK_10.0.0/intro/src/tpc/db2z_creationoftemporarytables.html

http://www.cs.newpaltz.edu/~pletcha/DB/db2_TempTables.html

http://www.clipartkid.com/images/85/thumbs-up-happy-smiley-emoticon-clipart-royalty-free-public-domain-pyaRBJ-clipart.png

https://web.stanford.edu/~bobonich/glances%20ahead/III.logic.language.html

https://en.wikipedia.org/wiki/Relational_algebra

https://en.wikipedia.org/wiki/Conjunctive_query

http://ceur-ws.org/Vol-1087/keynote2.pdf

http://ceur-ws.org/Vol-1087/keynote2slides.pdf