TEMPORARY TABLES

- Yes, they do exists! IBM DB2 version 10.0.0 allows them!
- Note: a declared temporary table must be declared!
  - DECLARE GLOBAL 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:
  - http://www.ibm.com/support/knowledgecenter/SSEPEK_10.0.0/intro/src/tpc/db2z_creationofftemporarytable.html
  - http://www.csnewpaltz.edu/~pletcha/DB/db2_TempTables.html

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!

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

- 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.
- <u>Example:</u> 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

- *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"

- Using IN and NOT IN operators with a Multiple-Row Sub-query:
  - http://www.w3resource.com/sql/subqueries/multiple-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

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

Equivalent to:

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

IN IS A PREDICATE ABOUT R's TUPLES

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

Two 2's.

(1,2) satisfies the condition; 1 is output once.

One loop, over the tuples of R.

R

a	b
1	2
3	4

S

b	c
2	5
2	6

THIS QUERY PAIRS TUPLES FROM R, S

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

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

Double loop, over the tuples of R and S.

R

a	b
1	2
3	4

S

b	c
2	5
2	6

RECALL: ORIGINAL QUERY

Option 1: use IN

Option 2: use = ANY()

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


RECAP

- 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.

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()

Sells

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

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

Result

Bar
Jane
Joe

THE OPERATOR: EXISTS

- 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

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.

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

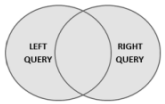
Set of beers with the same manf as b1, but not the same beer.

Notice the SQL "not equals" operator.

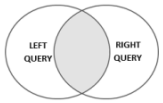
UNION, INTERSECTION, AND DIFFERENCE

- 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>)

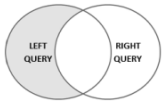
UNION, INTERSECTION, AND DIFFERENCE



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



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



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

EXAMPLE: INTERSECTION

- 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

```
SELECT  *
FROM    Likes
INTERSECT (SELECT  drinker, beer
           FROM    Sells, Frequents
           WHERE   Frequents.bar = Sells.bar
           );
```

The sub-query is really a stored table!

The drinker frequents a bar that sells the beer.