HOW TO CREATE SQL QUERY IN MS ACCESS:

Open “Create” tab at the top menu. Click “Query Design” and close QBE window that shows tables. Right click on the query name and select “SQL View”. Write your SQL in the blank page and click on “Run” button at the top panel. If you have syntax errors in SQL, it will show no results. You can also choose the type of SQL query at the top menu (e.g. choosing UPDATE query).

1. BASIC SQL QUERIES

SELECT attributes
FROM tables
WHERE condition
GROUP BY attribute
HAVING condition
ORDER BY attributes

Every select statement must have SELECT and FROM clauses, however other clauses are optional meaning that you will use these clauses whenever necessary in your query.

**Query 1:** Find the employee_no of employees who have assigned a duty for shift 3.

```sql
SELECT employee_no
FROM duty_allocation
WHERE shift = 3
```

**Query 2:** Find the employee_no, name, and skill of employees who have assigned duty on October 24, 2002.

```sql
SELECT employee.employee_no, name, skill
FROM employee, duty_allocation
WHERE employee.employee_no = duty_allocation.employee_no AND
day = '#24/10/2002#';
```

**Query 3:** Get the employee_no and name of employees who are older than "John Doe".

```sql
SELECT e1.employee_no, e1.name
FROM employee e1, employee e2
WHERE e2.name = "John Doe" AND
    e1.date_of_birth < e2.date_of_birth;
```

**BETWEEN Operator:**

**Query 4:** Find the employee_no and date_of_birth of each employee born between 01.01.1972 and 01.01.1980.

```sql
SELECT employee_no, date_of_birth
FROM employee
WHERE date_of_birth >= '#01/01/1972#' AND date_of_birth <= '#01/01/1980#';
```

```sql
SELECT employee_no, date_of_birth
FROM employee
WHERE date_of_birth BETWEEN '#01/01/1972#' AND '#01/01/1980#';
```
IN Operator

Query 5: Find the number, name, and city of each employee who lives in Ankara, Istanbul or Izmir.

SELECT employee_no, name, city
FROM employee
WHERE city IN ("Ankara", "Istanbul", "Izmir");

LIKE Operator

Query 6: Get the name and number of each employee whose name has the letter e as the second last letter.

SELECT employee_no, name
FROM employee
WHERE name LIKE "%e?";

NULL Operator

Query 7: Find all employees who has a skill.

SELECT employee_no, name
FROM employee
WHERE skill IS NOT NULL;

2. RELATIONAL ALGEBRA and SQL

A basic SQL query usually includes SELECT-FROM-WHERE keywords. The correspondence of basic SQL keywords and RA operators can be thought as follows:

SELECT attribute1, attribute2…  <------- Projection: \( \Pi_{\text{attribute1, attribute2}} \)
FROM relation1, relation 2…  <------- Cartesian Product: relation1 X relation2
WHERE condition1 (AND|OR) condition2…  <------- Selection: \( \sigma_{\text{condition1 and/or condition2}} \)

Thus, some example queries and their RA equivalents are as follows:

1. SELECT employee_no
   FROM duty_allocation DA
   WHERE shift = 3
   \(<==\> \Pi_{\text{employee_no}}(\sigma_{\text{shift =3}}(DA))\)

2. SELECT E.employee_no, DA.employee_no, name, skill
   FROM employee E, duty_allocation DA
   \(<==\> \Pi_{E,employee_no, DA.employee_no, name, skill}(E \times DA)\)

Instead of only computing a Cartesian product (as in the second query above), queries usually put some constraints on the result of the Cartesian product. So, it may become an equi-join or theta-join. Consider the following query.

3. SELECT E.employee_no, name, skill
   FROM employee E, duty_allocation DA
   WHERE E.employee_no = DA.employee_no (equi-join condition between E and DA)

The corresponding RA expression is

\( \Pi_{E,employee_no, name, skill}(E \bowtie_{E.employee_no \bowtie DA.employee_no} DA), \)
Here, the join condition specifies an equality condition on some attributes (e.g., employee_no). Recall that, join conditions do not always specify equalities, and if they involve inequalities (E.age < DA.age, etc.) they correspond to theta-join operator.

### 3. USING GROUP BY and HAVING CLAUSES

**Query:** Find the employee_no for each employee who had assigned more than one duty for shift 3.

```sql
SELECT employee_no
FROM duty_allocation
WHERE shift = 3
GROUP BY employee_no
HAVING COUNT(*) > 1
ORDER BY employee_no;
```

Assume that you have the following tuples in your DUTY_ALLOCATION table:

<table>
<thead>
<tr>
<th>posting_no</th>
<th>employee_no</th>
<th>day</th>
<th>shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>1/1/2002</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>44</td>
<td>11/10/2000</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>104</td>
<td>10/10/2000</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>7</td>
<td>1/1/2001</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>57</td>
<td>3/5/2000</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>57</td>
<td>3/1/2001</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>57</td>
<td>4/3/2001</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>57</td>
<td>3/5/2001</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>39</td>
<td>10/10/2000</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>39</td>
<td>1/1/2001</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>39</td>
<td>1/1/2002</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>5/5/2000</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>3/5/2001</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>5/5/2002</td>
<td>3</td>
</tr>
</tbody>
</table>

Consider the processing of the above query in the following.
Processing of the select statement:

Select statements are processed step by step. At each step an intermediate result is obtained.

1. After the FROM clause:

<table>
<thead>
<tr>
<th>posting_no</th>
<th>employee_no</th>
<th>day</th>
<th>shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>1/1/2002</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>44</td>
<td>11/10/2000</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>104</td>
<td>10/10/2000</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>7</td>
<td>1/1/2001</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>57</td>
<td>3/5/2000</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>57</td>
<td>3/1/2001</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>57</td>
<td>4/3/2001</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>57</td>
<td>3/5/2001</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>39</td>
<td>10/10/2000</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>39</td>
<td>1/1/2001</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>39</td>
<td>1/1/2002</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>5/5/2000</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>3/5/2001</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>5/5/2002</td>
<td>3</td>
</tr>
</tbody>
</table>

2. After the WHERE clause:

<table>
<thead>
<tr>
<th>posting_no</th>
<th>employee_no</th>
<th>day</th>
<th>shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>57</td>
<td>3/1/2001</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>57</td>
<td>3/5/2001</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>39</td>
<td>1/1/2001</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>5/5/2000</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>5/5/2002</td>
<td>3</td>
</tr>
</tbody>
</table>

3. After the GROUP BY clause:

<table>
<thead>
<tr>
<th>posting_no</th>
<th>employee_no</th>
<th>day</th>
<th>shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>39</td>
<td>1/1/2001</td>
<td>3</td>
</tr>
</tbody>
</table>

4. After the HAVING clause:

<table>
<thead>
<tr>
<th>posting_no</th>
<th>employee_no</th>
<th>day</th>
<th>shift</th>
</tr>
</thead>
</table>

5. After ORDER BY clause:

<table>
<thead>
<tr>
<th>posting_no</th>
<th>employee_no</th>
<th>day</th>
<th>shift</th>
</tr>
</thead>
</table>

6. After the SELECT statement:

<table>
<thead>
<tr>
<th>employee_no</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
<tr>
<td>57</td>
</tr>
</tbody>
</table>
Removing "Duplicate Rows" from the Query Result

Query1: Find all different cities from the EMPLOYEE table.

SELECT DISTINCT city
FROM employee;

Statistical Functions: COUNT, MIN, MAX, SUM, AVG, STDDEV, VARIANCE

Query2: How many employees are registered in the EMPLOYEE table?

SELECT COUNT(*)
FROM employee;

Query3: What is the highest pay_rate?

SELECT MAX(pay_rate)
FROM employee;

STDDEV and VARIANCE functions are used to find the standard deviation and variance of a column respectively. They are used similar to the AVG function.

SUM, AVG, STDDEV, and VARIANCE functions are only applicable to columns and expressions with a numeric data type whereas COUNT, MIN, and MAX can be applied to both numeric and alphanumeric data types.

NULL values are not included in the calculation of the statistical functions, except COUNT(*) which counts the number of rows.

Set Operators

UNION, INTERSECTION, and MINUS are equivalent to the union, intersection, and difference operators respectively in set.

Query 4: Give the number and name of each employee who has skill engineer or consultant.

SELECT employee_no, name
FROM employee
WHERE skill = "engineer"
UNION
SELECT employee_no, name
FROM employee
WHERE skill = "consultant";

ACCESS'2010 does not support INTERSECTION and MINUS operators. You can use AND, OR, NOT, IN, and EXISTS operators where appropriate, in order to simulate INTERSECTION and MINUS operators.
4. IMPORTANT NOTES ABOUT THE USE OF GROUP BY and AGGREGATE FUNCTIONS IN SQL

1) For GROUP BY:

   SELECT att_list1, aggregate expression(s) 
   ... 
   GROUP BY att_list2 
   HAVING group_qualifier (involving att_list3)

   - Att_list1 ⊆ att_list2. Every attribute that appears in the select clause must also appear in grouping list.
   - The expressions appear in having clause must have a SINGLE value per group. Thus, any attribute in att_list3 must either appear as an argument to an aggregate operator, or must also appear in att_list2 (i.e., att_list3 ⊆ att_list2)

2) For AGGREGATE FUNCTIONS:

   Aggregate functions take a collection of values as input and return a single value. Some example are: Avg, sum, min, max, count, etc. The latter three functions can be applied to both numeric and alphanumeric fields.

   - The result of aggregate function (a relation with a single attribute, containing a single row with a computed value) can be converted to just a value by SQL, so we can make a comparison such as “age = SELECT (max(age) from employee)” in the WHERE clause of some query.
   - DISTINCT can be used with aggregate functions. (Except COUNT(*)). Using DISTINCT is meaningless for min and max. Recall that retaining duplicates is sometimes important in computing AVG.
   - Aggregate functions can NOT be composed, i.e., avg(sum()) is not allowed!
   - Count (*) counts NULL values. In all other cases, NULL is discarded.
   - If the SELECT clause uses an aggregate operation, then it MUST use ONLY AGGREGATE operators UNLESS the query contains a GROUP BY clause. (You CAN'T say “SELECT name, max(age) FROM employee”. This is illegal in SQL! But, you can say SELECT e_no, count(*) FROM duty_allocation GROUP BY e_no”. Here, the use of GROUP BY in the query allows you to use both e_no and count(*) in the SELECT clause.)

3) For SET operators: By DEFAULT, duplicates ARE eliminated. To have duplicates, use versions suffixed with ALL, e.g., UNION ALL, etc.

5. CREATING, MODIFYING AND REMOVING A TABLE USING SQL

1. Table creation:

   CREATE TABLE table-name (attribute list)

2. Table modification:

   ALTER TABLE existing-table-name ADD column-name data-type [, …]
   ALTER TABLE existing-table-name DROP column-name

3. Table removal:

   DROP TABLE existing-table-name
6. ADD/DELETE/UPDATE QUERIES using SQL

Up to now, we have restricted our attention to the extraction of information (using SQL queries) from the database. Now, we consider deleting, inserting and updating the tuples in the database.

1. Deletion

- We can delete only whole tuples from a database table, but not values on only particular attributes.

```
DELETE FROM table-name
WHERE condition
```

The delete statement **first** finds all tuples satisfying the condition, and **then** deletes them from the table.

**Example 1.** Delete all employees who were born before 1/1/1960.

```
DELETE FROM employee
WHERE date_of_birth < #01/01/1960#;
```

- If you omit the where clause, all tuples will be deleted from the table (but the table itself will still exist)!

```
DELETE FROM employee;
```

- The condition in the where clause may be as complex as a select command’s where clause.

**Example 2.** Delete all employees who live in Milano, Roma or Napoli.

```
DELETE FROM employee
WHERE city IN ("Milano", "Roma", "Napoli");
```

- A delete command operates on only one relation. If you want to delete tuples from several tables, use one delete command for each table. But, you may still reference any number of tables in a select-from-where block nested in the where clause of delete command. You can even reference the relation from which tuples are to be deleted.

Note that, the condition of where clause is evaluated **before any deletion** is performed. (Otherwise, the average will continuously change, and the result of the query will be wrong!)

2. Insertion

- To insert data, we either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted. Clearly, the attribute values for inserted tuples must match the attribute domains specified in the table schema.

**Example 3.** Insert a duty with posting_no 5, employee_no 10, day 10/10/2002 and shift 1.

```
INSERT INTO duty_allocation
VALUES (5, 10, #10/10/2002#, 1);
```

Alternatively, you can specify the attribute names in the insert statement as well.

```
INSERT INTO duty_allocation (posting_no, employee_no, day, shift)
VALUES (5, 10, #10/10/2002#, 1);
```

Or,

```
INSERT INTO duty_allocation (day, posting_no, employee_no, shift)
VALUES (#10/10/2002#, 5, 10, 1);
```

- You can also insert tuples that are obtained as a result of a query.
• The select statement is fully evaluated before any insertion operation is performed.

Example 4.

```sql
INSERT INTO employee
SELECT *
FROM employee;
```

In the above example, what happens if the select clause is not evaluated first?

3. Update

• We can update a value in a tuple using the update statement. As before, we can choose the tuples to be updated using a query.

Example 5. Increase all pay_rates in employee table by 5%.

```sql
UPDATE employee
SET pay_rate = pay_rate * 1.05;
```

• The where clause of the update statement may include any construct legal in the where clause of a select statement (including nested selects). As before, a nested select within an update may refer the table to be updated, and the update is performed only after the where clause is evaluated (and the tuples to be updated are determined).