Relational Database Development

152-156

Structured Query Language (SQL)

Single Table Queries

<table>
<thead>
<tr>
<th>Notes</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text References</td>
<td>Import Employee database in MySQL</td>
</tr>
<tr>
<td>- Basic Select</td>
<td>Pages 97 – 98, 100</td>
</tr>
<tr>
<td>- Selecting Fields</td>
<td>Pages 98 – 99</td>
</tr>
<tr>
<td>- Selecting Records (Criteria)</td>
<td>Pages 100 – 107, 110 – 112, 128</td>
</tr>
<tr>
<td>- Distinct</td>
<td>Pages 117 – 119</td>
</tr>
<tr>
<td>- Sorting Results</td>
<td>Pages 112 – 114</td>
</tr>
<tr>
<td>- Calculated Fields</td>
<td>Pages 107 – 110</td>
</tr>
<tr>
<td>- Calc Fields and Order By, Where</td>
<td>Pages</td>
</tr>
<tr>
<td>- Formatting Results</td>
<td>Pages</td>
</tr>
<tr>
<td>- Grouping</td>
<td>Pages 114 – 117, 123 – 128</td>
</tr>
<tr>
<td>- Select Statement Clause Order</td>
<td>Pages</td>
</tr>
<tr>
<td>- Sub-Queries</td>
<td>Pages 120 – 123</td>
</tr>
</tbody>
</table>

Access the MySQL documentation online for more information on MySQL commands and functions:
http://dev.mysql.com/doc

Basic Query

- Select * From tblName;
  - * represents all fields

Selecting Specific Fields for Display

- List the fields in the order desired between Select and From
- Separate fields with a comma
**Selecting Records**

- Add the **Where** keyword after the table name
- Follow the keyword with a condition that records must meet to be included in the query results
- **Select** * From tblName Where condition**
- SQL Conditional operators
  - =, <, >, <=, >=, <>, !=
  - Between…And…
- Literals for character data comparison must be surrounded with ‘apostropes’ or “quotes”
- Literals for date data comparison must be surrounded with ‘apostrophes’ in yyyy-mm-dd format or entered as yyyymmdd (no dashes, apostrophes optional)
- Compound condition operators
  - And, Or, Not, &&, ||
  - Seems to be no order of precedence. All conditions evaluated left to right.

- Bit fields require a little extra thought
  - You can compare to the bit values 0 (false) and 1 (true)
  - Alternatively, you can compare a bit field to the values False and True
    - This seems more self-documenting
  - The **best** way to use bit (yes/no true/false) values in conditions is to just use the field name (see first examples below)
    - 0 = false any non-zero = True
    - Works with imported Access data where true = -1
  - Examples:

```sql
Select * From tblStudent
   Where fullTime;

Select * From tblStudent
   Where Not fullTime;

Select * From tblStudent
   Where fullTime = 1;

Select * From tblStudent
   Where fullTime = True;
```

- Import was from Access.
  - Use Update command to change all -1 to 1
  - set sql_safe_updates=0;

**Activity**

<table>
<thead>
<tr>
<th>Notes</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notes</td>
<td>Same fields, but only show Coloma (plant ID=1)</td>
</tr>
<tr>
<td></td>
<td>Display (all fields) employees hired between 1997 and 2001 (17)</td>
</tr>
<tr>
<td></td>
<td>Display employees not working in Polonia=2 (37) using &lt;&gt; != not</td>
</tr>
<tr>
<td></td>
<td>Display employees who are not union members (=0, =False, not unionmember) (8 rows)</td>
</tr>
<tr>
<td></td>
<td>Display employees who are union members (True) (no match)</td>
</tr>
<tr>
<td></td>
<td>Display where union = -1</td>
</tr>
<tr>
<td></td>
<td>Import was from Access. Use Update command to change all -1 to 1 set sql_safe_updates=0;</td>
</tr>
<tr>
<td></td>
<td>Display employees who are union members (True) (42 rows)</td>
</tr>
<tr>
<td></td>
<td>Display females in Coloma(1) who are union (2 rows)</td>
</tr>
</tbody>
</table>
### Notes

**In**
- `fieldname In (valuelist)`
- Records that match any one of the values in `valuelist` are included in the query results
- Shorter than using multiple OR conditions
- Example:
  - `Where Color In ('Red', 'White', 'Blue')`

**Like**
- `fieldname Like substring`
- `Like` allows you to search for records where a field contains a given substring
- The substring can appear at the beginning, end or in the middle of the field.
- The `Like` keyword is always used with the wildcard character: `%`
  - The wildcard character represents any character or characters (including no characters).
  - The wildcard character can be placed at the beginning of the substring, end of the substring, in the middle of the string, or any combination of those locations.

- Examples:
  - `Where Name Like 'Dav%'`
    Shows all names that start with Dav
  - `Where Name Like '%son'`
    Shows all names that end with son
  - `Where Name Like '%son%'`
    Shows all names that contain the letters son anywhere in the field.
  - `Where Name Like 'A%son'`
    Shows all names that start with the letter A and end in son

- There is a second wildcard character, underscore (_), that matches any single character.
  - `Name Like T_m`
    Shows all names that are exactly 3 characters long, starting with T, ending with m (Tom, Tim, T2m).

### Activity

- Select employees who work in Coloma(1) and Polonia(2). (25 rows)
- Use Or
- Use In
- Display trips where the destination starts with “C” (59 rows)
- That contains “c”
- Display employees whose phone contains 333 (2 rows)
Notes

- **RLike**
  - Similar to **Like** but using regular expressions
    ```sql
    Where custName RLike '^[afw].*$'
    ```
  - All customers whose names start with a, f or w

- **Is Null, Is Not Null**
  - To check for empty fields use:
    ```sql
    fieldname Is Null
    Not fieldname = Null
    ```
  - (doesn’t work, but not syntax error)

**DISTINCT**

- Removes duplicate values from a queries results
- **Select** **Major** From **tblEnroll**
  ```sql
  Where Course='152-120';
  ```
  - This example will simply list the major of each student in this class. If more than one student has the same major, the major will be repeated for each student.

- **Select Distinct(Major)** From **tblEnroll**
  ```sql
  Where Course='152-120';
  ```
  - This version only shows each major once.

- **Select Count(Distinct(CrsNumber))** From **tblOfferings**
  ```sql
  Where Semester='SP2002';
  ```
  - Counts the number of courses offered in the SP2002 semester, ignoring duplicate offerings of the same course.

Activity

- List employees whose lastnames starts with a, d or p
- List trips where duration is null.
- Expenses is null
- Expenses null and duration not null

Using **tblEmployee**

- generate a list of plants
- Generate a list of destinations that have been traveled to.
- Generate a list of destinations each employee (ID) has traveled to (sort by city, employee)
  (232 rows)
- Note # of rows in trips
  (300)
### Sorting Query Results

- Query results are displayed in an undefined order unless you specify otherwise.
- **Order By** clause allows you to specify how records are to be sorted.
- **Order By** `field1[Desc] [, field2]` etc.
- If **Order By** clause is used with the **Where** clause, the **Where** clause must be specified first.
- The **Order By** keywords are followed by one or more field names that specify which fields’ values should be used to sort the list.
  - The first field listed is the **primary sort key**—the list sorted by these values first.
  - The second field is the **secondary sort key**—if the values in the primary sort key are the same, those records are sorted by the values in this sort key.
  - You can specify as many secondary sort keys as you need to break ties in the previous sort keys.
- By default, records are sorted in ascending order (A-Z, 0-9). To reverse the sort order (**descending**), use the keyword **Desc** after any sort key.
- Examples:
  - **Order By** ZipCode
  - **Order By** LName, FName
  - **Order By** AmtOwed **Desc**
  - **Order By** AmtOwed **Desc**, LName, FName
  - **Where** ZipCode='54481' **Order By** LName

### Calculated Fields

- As you know, calculated fields are not typically stored in the database. Queries can be used to display calculated fields users need.
- To create a calculated value, simply enter the equation for the calculated field in the field list. Use parenthesis to improve readability.
- Calculated values can also be used in comparisons (**Where**).
- `+-*/` are the only operators available. Everything else is a function.
- Note, you can include the select all operator in the field list along with calculated fields:

  ```sql
  Select *, InstCost*0.055 From tblInstall
  ```

- **Activity**
  - Provide a list of trips to Chicago order by expenses.
  - Most expensive first.
  - Where expenses > 500 (8 rows)
  - Provide a list of cities and expenses sorted by destination then expenses, descending
  - Calc cost per day for every trip where cost per day > 200 order by cost per day desc
### Date Calculations
- To add a certain number of days to a date, you can use the `AddDate` or `SubDate` functions.
  - Examples: `AddDate(StartDate(), 45)`
  - `SubDate(StartDate(), 30)`
- To add months or years, you can use the second version of the `AddDate` or `SubDate` function.
  - `AddDate(datevalue, Interval value intType)`
  - `intType` is either `Day`, `Week`, `Month` or `Year`
  - `value` is the number of days, months or years you want to add
  - `datevalue` is the name of the field or another date value you want to add to.
- Newer versions of MySQL even allow you to add intervals to dates by simply using the `+` operator.
  - Subtraction works too

Examples:
```
Select checkOutDate, 
    AddDate(checkOutDate, Interval 14 Day) 
    As dueDate 
From tblLibrary;

Select CustName, 
    AddDate(startDate, Interval 18 Month) 
    As subscriptionEnd 
From tblSubscribers;

Select checkOutDate, 
    checkOutDate + Interval 14 Day 
    As dueDate 
From tblLibrary;
```

### Concatenating Strings
- To concatenate strings in MySQL, you must use the `Concat` function: neither “+” nor “&” work.
- Surround literals with apostrophes.
- You CAN include numeric fields in a Concat statement.
  - Example:
```
Select Concat(City, ',', 'State', ',', Zip) As CSZ 
From tblCustomer;
```

<table>
<thead>
<tr>
<th>Notes</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date Calculations</td>
<td>Calc return date from every trip</td>
</tr>
<tr>
<td></td>
<td>Using interval</td>
</tr>
<tr>
<td></td>
<td>Using + (note format)</td>
</tr>
<tr>
<td></td>
<td>Using + interval</td>
</tr>
<tr>
<td>Concatenating Strings</td>
<td>Show a list all employees and their hire dates in one field.</td>
</tr>
</tbody>
</table>
Renaming Calculated Fields
- To improve the appearance of the query results and simplify Order By clauses, you can assign field names (an alias in SQL speak) to calculated fields
  - If you don’t, the formula appears as the column header
- Include the keyword As or, simply a space, after the formula, followed by the name for the field
  - Example:
    (See examples above)

Using Calculated Fields in Order By and Where Clauses
- You CANNOT use an alias in a Where clause
- For example (using the tblCustomer example from above), you CAN NOT append the following Where clause to the SQL statement

```sql
WHERE CSZ LIKE 'Steven%'
```
- Instead, you’d have to repeat the formula

```sql
WHERE CONCAT(City, ',', ', ', State, ' ', Zip) LIKE 'Steven%'
```
- Actually, depending on your needs, you might be able to leave out the literals (comma and spaces) and maybe even the zip code in the Where clause. They shouldn’t affect the results of the filter.
- You CAN use an alias in an Order By clause (see below)
  - Can also repeat the formula though that’s much longer.
  - For example (using the tblCustomer example from above), you CAN append the following Order By clause to the SQL statement

```sql
ORDER BY CSZ
```

<table>
<thead>
<tr>
<th>Notes</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Renaming Calculated Fields</td>
<td>Set calc field name to Combined</td>
</tr>
</tbody>
</table>
|   ➢ To improve the appearance of the query results and simplify Order By clauses, you can assign field names (an alias in SQL speak) to calculated fields
|   ➢ If you don’t, the formula appears as the column header
|   ➢ Include the keyword As or, simply a space, after the formula, followed by the name for the field
|   ➢ Example:
|       (See examples above)               |                      |
| • Using Calculated Fields in Order By and Where Clauses | Sort by Combined |
|   ➢ You CANNOT use an alias in a Where clause
|   ➢ For example (using the tblCustomer example from above), you CAN NOT append the following Where clause to the SQL statement
|       WHERE CSZ LIKE 'Steven%'
|   ➢ Instead, you’d have to repeat the formula
|       WHERE CONCAT(City, ',', ', ', State, ' ', Zip) LIKE 'Steven%'
|   ➢ Actually, depending on your needs, you might be able to leave out the literals (comma and spaces) and maybe even the zip code in the Where clause. They shouldn’t affect the results of the filter.
|   ➢ You CAN use an alias in an Order By clause (see below)
|   ➢ Can also repeat the formula though that’s much longer.
|   ➢ For example (using the tblCustomer example from above), you CAN append the following Order By clause to the SQL statement
|       ORDER BY CSZ

(10 rows)
Formatting Results

- Formatting is normally done by *front end* programs (written in a programming language) that access the database tables.
- If you need to however, you can format the output of SQL queries.

- Formatting Decimal Values
  - To format numbers, use the `Format` function

  ```
  Format(field, decimals)
  ```

  - The `Format` function requires two parameters:
    - the field to format
    - the number of decimal places to display
  - All numbers include commas when appropriate
  - Example:

    ```sql
    Select Format(Avg(Salary),2) As AvgSalary
    From tblEmployee;
    ```

- Formatting Dates
  - Use the `Date_Format` function to format dates

  ```
  Date_Format(datefield, 'formatstring')
  ```

  - `Datefield` is the field name or date literal to be formatted
  - `Formatstring` will probably include some of the following:
    - `%d` (*day*)
    - `%m` (*month*)
    - `%Y` (*4-digit year*)
  - See the `Date_Format` documentation for additional formatting codes

  ```sql
  List trip date, duration and return date of each trip.
  ```

  ```
  Format MMMM d, yyyy
  (see documentation)
  ```

  ```sql
  Sort by return date desc
  ```
### Functions

<table>
<thead>
<tr>
<th>Notes</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td></td>
</tr>
<tr>
<td>Returns the number of records that would have appeared in the query results instead of the records themselves</td>
<td></td>
</tr>
<tr>
<td>Select Count(*) From STUDENT Where Major=’CIS’</td>
<td></td>
</tr>
<tr>
<td>If implementation of SQL doesn’t support the *, use the primary key field name instead</td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td></td>
</tr>
<tr>
<td>Returns the sum of a field of all the records that would have appeared instead of the records themselves</td>
<td></td>
</tr>
<tr>
<td>Select Sum(Credits) From ENROLL Where LName=’Gaul’</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td></td>
</tr>
<tr>
<td>Returns the average of a field from all the records.</td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td></td>
</tr>
<tr>
<td>Returns the smallest value of a field from all the records.</td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td></td>
</tr>
<tr>
<td>Returns the largest value of a field from all the records.</td>
<td></td>
</tr>
</tbody>
</table>

Display total number of trips (to Chicago)
Display total amount of expenses (format) (to Chicago)
Display min and max duration
Display average cost per day
Notes

Grouping

- Grouping combines records with the same value in a designated field and calculates a statistical value(s) for every group
  
  **Group By name**
  
  **Select** LName, Sum(Credits) **From** ENROLL  
  **Where** Semester='SP2014'  
  **Group By** LName;
  
  - Displays the total credits for each student enrolled in the SP2014 semester (sorted by last name).
  - Results of queries that use grouping are sorted by the group field by default. You can specify an **Order By** clause to override this.

- **Having** clause
  
  - The **Having** clause allows you to control which groups show in the query results
  - You can use an alias in a **Having** clause
  
  **Select** LName, Sum(Credits) **As** totalCredits **From** tblEnroll  
  **Where** Semester='SP2014'  
  **Group By** LName  
  **Having** totalCredits>12  
  **Order By** totalCredits **Desc**;
  
  - Displays the total credits for each student enrolled in the SP2014 semester (sorted by total credits descending)
  
  **(2013)** Having is compatible with format. However, formatting converts numbers to strings. When doing a Having comparison, compare to a string.
  
  ```sql
  select plantid, format(sum(salary),1) as TotalSalary from tblemployees group by plantid having totalsalary>'400,000';
  ```

- **Having** vs. **Where**
  
  - **Where** restricts records from being included in the query results
  - **Having** restricts group statistics from being included in the query results

- **With Rollup**
  
  - Adding this clause to a grouped query includes the grand total (or grand average) in an additional row at the bottom of the query.
  - With Rollup must follow the Group By clause
  - With Rollup is not compatible with Order By

Activity

- Calc total salary for all employees format
- For non-union employees
- Calc total salary grouped by union status
- Calc total salary grouped by plant
- Grouped by plant, then union status
- Add grand totals
- Calc average expenses per day for each employee
- Sort descending (format)
- Only include people who average over $300/day
- Only include Chicago
- Display the number of trips taken to each city in descending order of number trips.
## Notes

### Clause Order

- The last example under grouping above shows the correct order of clauses when many SQL clauses are included in the same statement.
- These clauses must always be specified in this order—the clauses cannot be rearranged.
  - For example, you cannot designate an `Order By` clause before the `Where` clause.

<table>
<thead>
<tr>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display name and salary of person with highest salary (using order by)</td>
</tr>
<tr>
<td>Display name and salary of union member with lowest salary.</td>
</tr>
<tr>
<td>Display name and salary of person with highest salary.</td>
</tr>
<tr>
<td>Display name an salary of union member with lowest salary.</td>
</tr>
<tr>
<td>Display destination and cost per day for any trip over the average cost per day sorted by cost per day, highest to lowest.</td>
</tr>
<tr>
<td>Display the city and avg cost with the highest average cost per day.</td>
</tr>
<tr>
<td>Note: can be done using Limit 1 (end of statement) but Limit is MySQL, not standard</td>
</tr>
</tbody>
</table>

### Sub-Queries

- Sub-queries allow you to use the results of one query as a criteria for another.
- For this unit, the sub-query must return a single value.
  - In the next unit, sub-queries will return multiple fields and multiple rows.
- Example:

  ```sql
  Display the student(s) with the highest GPA
  
  Select LastName, FirstName, GPA From tblStudent
  Where GPA= (Select Max(GPA) From tblStudent);
  ```

Select * for all employees who work in the Coloma plant using a subquery to determine plant.