Doing more with SQL

SQL Inbuilt Functions

Queries allow us to do basic data retrival, but sometimes we need to modify, format, or do calculations on data being used in, or returned from a query.

- Most SQL dialects have a number of inbuilt functions that allow you to do these sorts of operations in your queries.
  MySQL includes functions for:
  - String and text manipulation
  - Numeric and Math operations
  - Date and Time manipulation
  - Encryption and Compression
  - Control Flow
  - Data sorting/grouping
- You can view the MySQL functions at https://dev.mysql.com/doc/refman/8.0/en/func-op-summary-ref.html

MySQL Function Usage

MySQL functions work similarly to functions in most programming languages:

```
FUNCTION_NAME(param1,...);
```

- The function is run for every row in the table/result set it is applied to.
- The the return value of the function can be used in the query or result.

```
-- Can apply to the result
SELECT ROUND(column1) FROM TableA WHERE column1 > 5;

-- Or elsewhere in the query.
SELECT column1 FROM TableA WHERE ROUND(column1) > 5;
```

Some common MySQL Functions

These are not examinable; here for your reference

String and Text Functions

The following functions can be used to help us work with strings and text:

- `CONCAT`
  Concatenates 2 or more strings
- `FORMAT`
  Converts a number to a string formatted to a given number of decimal places
- `LENGTH`
  Return the length of a string in bytes
- `LOWER` and `UPPER`
  Converts a string to lowercase/uppercase
- `REGEXP` or `REGEXP_LIKE`
  Match using a regular expression. Similar to `LIKE` or `=`
- `SUBSTRING`
  Returns a string that is a subset of another string.

Plus many more. See https://dev.mysql.com/doc/refman/8.0/en/string-functions.html

Numeric and Math Functions

Most of the functions that you might find in the Math libraries of programming languages are also available in MySQL:

- `SIN`, `COS`, `TAN`, `DEGREES`, `RADIANS`, etc
  Functions for working with angles
- `EXP`, `LN`, `LOG10`, `LOG2`, `CONV`, etc
  Functions for working with logarithms, exponents and number bases
- `CEIL`, `FLOOR`, `ROUND`, `SIGN`, `MOD`, etc
  Functions for rounding/remainders
- `RAND`, etc
  Generate a random value
- Regular arithmetic operators `+ - * / %`

Plus more. See https://dev.mysql.com/doc/refman/8.0/en/numeric-functions.html

Date and Time Functions

The Date data type is common in MySQL, so many functions are included to help manipulate and format dates:

- `DATE_ADD`, `DATE_SUB`, `DATEDIFF`
  Functions for doing date arithmetic
- `CURDATE`, `CURTIME`, `NOW`
  Get the current date/time
- `YEAR`, `MONTH`, `DAY`, `DAYNAME`, `DAYOFWEEK`, `HOUR`, `MINUTE`, `SECOND`
  Retrieve individual components from a given date
- `TIME_FORMAT`
  Formats a date to a specific format

Again, many more as well as variants.
See https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html

Encryption and Compression Functions

The following functions can be used to help perform common crytographic and compression tasks:

- `UUID`
  Generate a unique indentifier; can be used instead of AUTO_INCREMENT for generating primary keys.
- `SHA1`, `SHA2`
  Hashing Algorithms
- `RANDOM_BYTES`
  Generate a random string
- `COMPRESS`, `UNCOMPRESS`
  Compress/uncompress data, rather than storing in plain text.

See https://dev.mysql.com/doc/refman/8.0/en/encryption-functions.html

Control Flow

Conditional functions allow us to change data under certain conditions

- `CASE`
  Equivalent to a switch statement
- `IF`, `IFNULL`
  Return a given value depending on whether a condition is true/false

See https://dev.mysql.com/doc/refman/8.0/en/control-flow-functions.html

Data Set Functions

The following functions can be used to help us work sets of data:

- `SUM`, `COUNT`
  Get the sum of or number of rows in a set
- `MIN`, `MAX`, `AVG`
  Get the min/max/average value of a set of values

See https://dev.mysql.com/doc/refman/8.0/en/group-by-functions.html

Working With Set Functions

Some functions rely on multiple rows to generate a result.

- E.g. COUNT, MAX, MIN, SUM, AVG
- These are known as aggregate functions

If you want to use an aggregate function as part of a WHERE clause however, this won't work as the conditions used in WHERE are evalueated row-by-row.

- E.g. Get a list of the cities where your customers live, but only if at least 10 of your customers live there.

Instead we can use the HAVING and GROUP BY clauses.

```
SELECT City
    FROM Customers
    GROUP BY City
    HAVING COUNT(CustomerID) >= 10;
```

Working With Set Functions

The GROUP BY clause is used to group common values together in a column.

- These groups are necessary for aggregate functions to work.
- Once we have our groups, aggregate functions can be used on the data.

The HAVING clause replaces the WHERE clause for aggregate data.

- It is evaluated after the WHERE clause on grouped/aggregate data

```
SELECT City
    FROM Customers
    WHERE Country = 'Germany'
    GROUP BY City
    HAVING COUNT(CustomerID) >= 10;
```

Using Multiple Queries together

Subqueries

We've already seen how we can combine the results of multiple queries into a single result using the UNION and INTERSECT operations.

Sometimes we may want to use the results of a query as part of another query.

- **Subqueries** allow us to use one or a set of results from a query inside another query:

```
SELECT * FROM TableA
  WHERE column1 = (SELECT column2 FROM TableB
                   WHERE column2 = 'a');
```

TableA

column1
a
b

TableB

column2
a
y

Result

column1
a

Subqueries

- A subquery must always return only a single column.
  - It can return multiple rows, but if it does, you must use set operations:

```
-- This query only works if the subquery returns 1 row
SELECT * FROM TableA
  WHERE column1 = (SELECT column2 FROM TableB);

-- This query works with multiple rows returned by the subquery.
SELECT * FROM TableA
  WHERE column1 IN (SELECT column2 FROM TableB);
```

TableA

column1
a
b
c

TableB

column2
a
b

Result

column1
a
b

Code Reuse

Stored Procedures

Stored procedures allow us to store commonly used queries that can be called for later use

Usage:

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

Run the procedure using `EXEC`:

```
EXEC procedure_name;
```

Stored Procedures with Parameters

We can also have stored procedures with Parameters:

```
CREATE PROCEDURE procedure_name @Param1 data_type, @Param2 data_type, ...
AS
sql_statement
GO;
```

```
EXEC procedure_name Param1 = "value";
```

Example:

```
CREATE PROCEDURE getUsers @Username nvarchar(30), @Email nvarchar(50)
AS
SELECT * FROM Users WHERE username = @Username OR email = @Email
GO;
```

Views

Another code reuse technique is through the use of Virtual Tables, known as Views.

- A view is a table created by a SELECT statement.
- Whenever the table is queried, the query is performed on the results of the select statement.
  - Allows us to simulate derived attributes

```
CREATE VIEW view_name
AS
select_statement;
```

- Notice unlike stored procedures no `GO` needed.

Views

Example:

```
CREATE VIEW RecentCustomers
AS
SELECT * FROM Customers
    WHERE cust_id IN
        (SELECT cust_id FROM Purcahses
            WHERE DATEDIFF(NOW(),purchase_date) < 7);
```