Session Summary

Advanced SQL

Structured Query Language (SQL) is the language used to communicate with databases. SQL is used almost everywhere when it comes to data communication between a server and a client. SQL has many implementations like SQLServer, MySQL, PostgreSQL etc. SQL queries are used to perform operations on a database like modifying, deleting, updating and subsetting.

Designing a Database Schema

A data warehouse is a collection of data with the following properties:

1. **Subject-oriented**: Data is categorised and stored by business subject rather than by the source, i.e. the data in the data warehouse should pertain to the particular problem you are trying to solve.
2. **Integrated:** Data is stored in various platforms in an organisation and you need to integrate/bring the data residing in various databases to a unified platform which can help you solve the business problem. It is constructed by integrating multiple heterogeneous data sources.
3. **Non-volatile:** Typically, data in the data warehouse is not deleted or altered without a legitimate reason.
4. **Time-variant:** Data is updated with the variation in time.

A data warehouse is essential for an enterprise. It gives an overall view of the entire organisation and makes it easy to carry out analytical operations on the data.

Updating Table

This section includes commands used in the session on the employee database

ALTER TABLE: Used to change the table structure
ADD: Used to add a new column to an existing table
DEFAULT: Used to populate the new column with a default value

Syntax: ALTER TABLE table_name
ADD column_name column_data_type
DEFAULT default_value;

Usage: ALTER TABLE employee
ADD des varchar(50)
DEFAULT 'manager';

UPDATE: Used to update the entries in a table.

Syntax: UPDATE TABLE table_name
SET column_name = value
WHERE condition

Usage: UPDATE TABLE employee
SET des = 'CEO'
WHERE super_ssn IS NULL;

DROP COLUMN: Used to delete a column from an existing table.

Syntax: ALTER TABLE table_name
DROP COLUMN column_name;

Usage: ALTER TABLE employee
DROP COLUMN des;

Changing Column Name and Data Type

MODIFY: Used to change the data type of an existing column.

Syntax: ALTER TABLE table_name
MODIFY column_name new_data_type;

Usage: ALTER TABLE employee
MODIFY salary float(10,4);

CHANGE: Used to change the name of an existing column.

Syntax: ALTER TABLE table_name

CHANGE old_column_name new_column_name column_data_type

Usage: ALTER TABLE employee
CHANGE minit minitial char(1);

Creating Table from Existing Table

CREATE TABLE AS: To create a new table containing elements of an existing table.

Syntax: CREATE TABLE new_table_name **AS**
Select (The components you need in the new table)

Usage: CREATE TABLE e1 **AS**
Select concat(fname, minit, lname) as 'Name', ssn, bdate from emp;

Changing Constraints

DROP PRIMARY KEY: To remove the primary key from an existing table

Syntax: ALTER TABLE table_name
DROP PRIMARY KEY

Usage: ALTER TABLE dependent
DROP PRIMARY KEY;

ADD PRIMARY KEY: To add a primary key from an existing table

Syntax: ALTER TABLE table_name
ADD PRIMARY KEY (column_name)

Usage: ALTER TABLE dependent
ADD PRIMARY KEY (dependent_name)**;**

DROP FOREIGN KEY: To remove an existing foreign key

Syntax: ALTER TABLE table_name
DROP FOREIGN KEY constraint_name

Usage: ALTER TABLE employee

DROP FOREIGN KEY fk_super_ssn

ADD FOREIGN KEY: To add a foreign key to an existing table

Syntax: ALTER TABLE table_name
ADD FOREIGN KEY constraint_name **FOREIGN KEY** (col_where_constraint_isadded)
REFERENCES table_name_to_be_refernced (column_name)

Usage: ALTER TABLE employee
ADD FOREIGN KEY fk_super_ssn **FOREIGN KEY** (super_ssn)
REFERENCES employee (ssn)

String and Date functions

STRING FUNCTIONS: Read about the various string functions here.

EXTRACT: To extract the year/month/date from a Date.

Syntax: EXTRACT(YEAR/MONTH/DAY FROM column_name)

Usage: SELECT EXTRACT(YEAR/MONTH/DAY FROM bdate)
FROM employee;

Window Functions

Used to display the group info in a way so as to preserve the properties of individual rows.

OVER: Used to invoke window function.

Syntax: Group_Function(column_name) **OVER** (window definition)

Usage: SELECT ssn, dno,
SUM(salary) **OVER()** as total_salary
from employee;

Frames

Syntax: Group_function(column_name) **OVER** (**PARTITION BY** frame_required)

Usage: SELECT ssn, dno,
SUM(salary) **OVER(PARTITION BY** dno**)** as dep_salary
from employee;

Named Windows

WINDOW: Used to define a named window.
Syntax: WINDOW window_name **AS** (window_definition)

Usage: SELECT ssn, salary,
row_number() over w as 'row_number',
FROM employee
WINDOW w **AS** (**ORDER BY** salary);

User Defined functions and Stored procedures

USER DEFINED FUNCTION: SQL provides an option to create functions. Once created and executed, they work the same as any other inbuilt function.

Syntax: CREATE FUNCTION function_name (input_variable_name input_variable_data_type)
RETURNS return_variable_date_type **DETERMINISTIC**
RETURN return_value

USAGE: CREATE FUNCTION hello (s char(20))
RETURNS char(50) **DETERMINISTIC**
RETURN concat('Hello, ', s, ' !');

DETERMINISTIC: Used to make sure that the function returns that same value for a particular input any number of times.

User Defined functions' Application

DELIMITER: Signals the end of a SQL statement. By default it is ';'. Can be changed using the delimiter command

Syntax: DELIMITER character_that_has_to_be_made_delimiter

Usage: DELIMITER $$

Application:
```
DELIMITER $$
CREATE FUNCTION project_pay_calc( pno int, num_of_hours float(4,2))
RETURNS float(8,2)
DETERMINISTIC
BEGIN
DECLARE project_pay_per_hour float(8,2);
IF (pno > 0) then
SET project_pay_per_hour = 1000;
ELSE
SET project_pay_per_hour = 3000;
END IF;
RETURN (project_pay_per_hour * num_of_hours);
END
$$
DELIMITER;
```

DECLARE: Used to declare variables in SQL.

BEGIN: Used to signify the beginning of a single SQL statement

END: Used to signify the end of a single SQL statement

Calling User defined function: A UDF can be called in a way similar to any other function.

Usage:
```
SELECT essn, pno, hours,
    project_pay_calc(pno, hours) as total_project_pay
    FROM works_on;
```

Stored Procedure

STORED PROCEDURE: Similar to a function, SQL provides the option for a stored procedure. It is a combination of SQL statements that can be called by using a keyword similar to a function. One of the biggest difference between a User defined function and a stored procedure is that where a UDF **necessarily** returns a single value, a stored procedure can return any number of values starting from zero.

Syntax: CREATE PROCEDURE procedure_name

Usage:
```
DELIMITER $$
CREATE PROCEDURE employee_details
(IN n char(9))
BEGIN
SELECT *
FROM employee
WHERE ssn = n;
END $$
DELIMITER ;
```

Query Optimisation

1) Use the **LIMIT** clause while writing initial queries.
2) While using the select clause, use only the columns that are required.
3) Prefer using the **WHERE** clause on **KEYS.**
4) If possible, calculate the result of where clause manually and try to combine conditions if possible.
5) Prefer using the **GROUP BY** clause on **KEYS.**
6) Prefer using the **ORDER BY** clause on **KEYS.**
7) Prefer using the **JOIN** clause on **KEYS.**
8) If there is a where clause in any or both the tables used in the join clause, it is preferable to transfer them to the join condition.
9) Use **NAMED WINDOWS** if any window is used more than once.
10) Follow good coding guidelines in general.