SQL Overview

Dr. Villanes

Software for class

sas

Proc SQL

SQLite

Open source SQL database engine

SQLite Studio

SQLiteStudio is a SQLite database manager

Getting started…

Download data for SAS and create a library: Jupiter.

Download *"Exercise"* SQLite database.

Overview of SQL

Terminology

SQL	SAS
Table	Data Set
Row	Observation
Column	Variable

Select Statement

A SELECT statement is **used to query one or more tables**. The results of the SELECT statement are written to the default output destination:

```
SELECT column1, column2, ...
FROM table_name;
```

Select Statement

A SELECT statement contains smaller building blocks called clauses.

```
SELECT column1, column2, ...
FROM table_name
WHERE column1='F'
ORDER BY column1;
```

clauses

Although it can contain multiple clauses, each SELECT statement begins with the SELECT keyword

Select Statement: Required Clauses

A SELECT statement contains smaller building blocks called clauses.

```
SELECT column1, column2, ...
FROM table_name;
```

- The **SELECT** clause specifies the columns and column order.
- The **FROM** clause specified the data sources

Select Statement: Optional Clauses

```
SELECT column1, column2, ...
FROM table_name
WHERE sql-expression
GROUP BY column_name
HAVING sql-expression
ORDER BY column_name <DESC>;
```

- The **WHERE** clause specifies data that meets certain conditions.
- The **GROUP BY** clause groups data for processing.
- The **HAVING** clause specifies groups that meet certain conditions.
- The **ORDER BY** clause specifies an order for the data.

The specified order of the above clauses within the SELECT statement is required.

Additional SQL Statements

```
SELECT expression;
CREATE expression;
DESCRIBE expression;
INSERT expression;

… and many more.
```

Specifying Columns

Querying All Columns in a Table

To print all of a table's columns in the order in which they were stored, specify an asterisk in a SELECT clause.

```
SELECT * FROM table_name;
```

```
proc sql;
SELECT * FROM table_name;
quit;
```

Querying Specific Columns in a Table

List the columns that you want and the order to display them in the SELECT clause.

```
SELECT column1, column2, ...
FROM table_name;
```

```
proc sql;
SELECT column1, column2
FROM table_name;
quit;
```

Naming a column

Name the new column using the AS keyword.

```
SELECT column1 as name
FROM table_name;
```

```
proc sql;
SELECT column1 as name
FROM table_name;
quit;
```

Table: **`jupiter.employee_information`**

Display: Employee_ID, Salary and create a new Column **Bonus,** which contains an amount equal to 10% of the employee's salary

Table: **`exercise.records`**

Display: ID, Capital_Gain and create a new Column **Bonus,** which contains an amount equal to 10% of the Capital_Gain

The SAS advantage

Create a report that includes the employee identifier, gender, and age for an upcoming audit.

```
proc sql;
select Employee_ID, Employee_Gender,
       int((today()-Birth_Date)/365.25)
       as Age
   from jupiter.employee_information;
quit;
```

Creating a Table

Creating and Populating a Table

```
CREATE TABLE table-name AS
SELECT column1 as name
FROM table_name;
```

```
proc sql;
CREATE TABLE table-name AS
SELECT column1 as name
FROM table_name;
quit;
```

Table: `jupiter.employee_information`

Create a table TEMP:

Employee_ID, Salary and Bonus

Table: **`exercise.records`**

Create a table Exercise.TEMP:

ID, Capital_Gain and Bonus

Specifying Rows

Eliminating Duplicate Rows

Use the *DISTINCT* keyword to eliminate duplicate rows.

```
SELECT distinct Department
FROM employee_information;
```

The DISTINCT keyword applies to all columns in the SELECT list. One row is displayed for each unique combination of values.

Subsetting with the WHERE Clause

Use a WHERE clause to specify a condition that the data must satisfy **before being selected**.

```
SELECT Department
FROM employee_information
WHERE salary > 30000;
```

A WHERE clause is evaluated before the SELECT clause.

Using the previous query:
Display: only those employees who receive bonuses less than $3000

Using the previous query:
Display: only those employees who receive bonuses greater than $200

ANSI Standard

One solution is to repeat the calculation in the WHERE clause.

```
proc sql;
select Employee_ID, Employee_Gender,
       Salary, Salary*.10 as Bonus
   from jupiter.employee_information
   where Salary*.10<3000;
quit;
```

ANSI standard

How does SQLite allow it?

- SQL standard: column aliases can be referenced in ORDER BY, GROUP BY and HAVING clauses.

- As an **extension**, SQLite also allows column aliases in WHERE and JOIN ON clauses, but again, such usage is non-standard (though very convenient at times).

- Neither the standard nor SQLite implementation allow referencing aliases in the SELECT clause.

Subsetting with Calculated Values

An alternate method is to use the CALCULATED keyword in the WHERE clause.

```
proc sql;
select Employee_ID, Employee_Gender,
       Salary, Salary*.10 as Bonus
   from jupiter.employee_information
   where calculated Bonus<3000;
quit;
```

SAS extension

The CALCULATED keyword is required when referring to any calculated column, character or numeric, in the SELECT or WHERE clause. It is not necessary with the ORDER BY or HAVING clause.

Subsetting with Calculated Values

You can also use the CALCULATED keyword in other parts of a query.

```
proc sql;
select Employee_ID, Employee_Gender,
       Salary, Salary*.10 as Bonus,
       calculated Bonus/2 as Half
   from jupiter.employee_information
   where calculated Bonus<3000;
quit;
```