SQL and Data

Learning to Retrieve Data Efficiently and Accurately
Introduce the class to structured query language (SQL) using database examples.

We will be using SQL on a number of small business databases.
Outline

• Section One
  – Overview
  – Vocabulary Overview
    • RDBMS and SQL
    • Selecting rows
  – SQL*Plus
  – Limiting selected data
  – Retrieving data from multiple tables
Overview

• Learn relational database concepts and SQL constructs needed to retrieve data
• The concepts and skills introduced will be reinforced by demonstrations and practice sessions
Vocabulary

• **Row** – A row is a combination of column values in a table; For example, the information about one customer in the table CUSTOMERS. A row is sometimes called a “record”

• **Table** – A table is the basic storage structure of an RDBMS, consisting of one or more columns and zero or more rows
Vocabulary

• **Column** – A column represents one kind of data in a table; For example, the name in the table CUSTOMERS. It is described with a column name and holds data of a specific type and size

• **Field** – at the intersection of a row and a column, you find a field. The field can contain data. If there is no data in the field, it is said to contain a null value
RDBMS and SQL

• A relational database management system (RDBMS) is a collection of objects or relations storing data and a set of operators that act upon the objects

• The basic functions provided for in an RDBMS include:
  – Storage of data
  – Access control to data
  – Mechanisms for retrieval and modification of data
RDBMS and SQL

• A database is a collection of individual, named objects, such as tables

• To access the database you execute a structured query language (SQL) command, which is the standard language for operating upon relational databases
RDBMS and SQL

• Features of SQL include:
  – Can be used by a range of users, including those with little or no programming experience
  – Is a non-procedural language
  – Is an English-like language

• There are many commands available in SQL this course will be limited to the SELECT command used for data retrieval
Did You Know??

• Mid 1970’s, developed under the name SEQUEL at IBM San Jose research facility.
  – Built for the prototype relational model, SYSTEM R.
• Renamed in 1980…..SQL
• ANSI committees in 1986 & 1987 produced a standards document which lends for the portability of SQL.
SQL*Plus

- An Oracle tool that recognizes and executes SQL statements
- SQL*Plus command categories:
  - Environment
  - Format
  - File Manipulation
  - Execution
  - Edit
  - Interaction
  - Miscellaneous
Oracle SQL*Plus

SQL*Plus: Release 8.1.5.0.0 - Production on Sun Jan 23 16:55:42 2000
(c) Copyright 1999 Oracle Corporation. All rights reserved.

Connected to:
Oracle8i Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production

SQL>
Selecting Rows

- The SELECT statement retrieves information from the database, implementing all algebraic operators.
- Example

  ```sql
  SELECT name
  FROM customers
  WHERE name LIKE 'Barrett%'
  ```
Selecting Rows

• Writing SQL commands.
  – Commands can be on one or more lines.
  – Tabs and indents can be used for readability.
  – Abbreviation and splitting of words are not allowed.
  – Commands are not case sensitive.
  – Commands are entered into the SQL buffer.
  – An SQL command is entered at the SQL prompt, and subsequent lines are numbered. This is called the SQL buffer.
  – Only one statement can be current in the buffer. To see current contents of the SQL buffer type “list” or “l”.
Selecting Rows

– To execute the statement in the SQL buffer:
  • Place a semicolon “;” at the end of last clause.
  • Place a semicolon “;” or slash “/” on the last line in the buffer.
  • Place a slash “/” at the SQL prompt.
  • Issue the “run” command at the SQL prompt.

– In its simplest form, a SELECT must include the following:
  • A SELECT clause, which specifies the columns to be displayed.
  • A FROM clause, which specifies the table containing the columns listed in the SELECT clause.
Selecting Rows

- The asterisk (*) selects all columns from the table
  - SELECT * FROM customers
- To select specific columns you must:
  - List the columns in the SELECT clause
  - Separate the columns by using a comma
  - Specify the columns in the order you want them to appear
  - SELECT id, name
    FROM customers
    WHERE name LIKE ‘Barrett%’
Selecting Rows

- SQL allows you to create arithmetic expressions on NUMBER and DATE datatypes using the following operators:
  - Add (+)
  - Subtract (-)
  - Multiply (*)
  - Divide (/)

- SELECT amount, amount-1, amount/2, amount+5, (amount*2)/2
  FROM invoices
  WHERE rownum < 10
Selecting Rows

• Multiplication and division take priority over addition and subtraction
• Operators of the same priority are evaluated from left to right
• Parentheses can be used to force prioritized evaluation and to clarify statements
  • SELECT 5+5-5+5 FROM dual; = 10
  • SELECT 5+5-(5+5) FROM dual; = 0
Selecting Rows

- When displaying the result of a query, SQL*Plus normally uses the selected column’s name as the heading. A column alias can be used to rename a column heading.

- The column alias immediately follows column name in the select statement. Double quotation marks are required if a alias contains spaces, special characters or is case sensitive.

  - SELECT name “Customer Name”, id FROM customers
    WHERE name LIKE ‘Barrett%’;
Selecting Rows

• SQL*Plus features a concatenation operator.
  – The concatenation operator is represented by “||”
  – The operator links columns or character strings to other columns.
  – The operator creates a resultant column that is a character expression.

• SELECT name||soc_sec_num
  FROM customers
  WHERE name LIKE ‘Barrett%’;
Selecting Rows

- A literal is any character, expression, or number included in the SELECT list that is not a column name or a column alias.
  - Date and character literal values must be enclosed within single quotation marks.
  - Each character string is output once for each row returned.

- SELECT name||’ SSN: ‘||soc_sec_num
  FROM customers
  WHERE name LIKE ‘Barrett%.’
Selecting Rows

- A NULL value is a value that is unavailable, unassigned, unknown, or inapplicable
  - NULL is not the same as zero or space
  - Arithmetic expressions containing a NULL value evaluate to NULL
- The NVL function is used to convert a NULL to an actual value
- Datatypes to use are date, character, and number
  - `SELECT name, NVL(tax_certificate_id, 'xxx')
    FROM customers
    WHERE name LIKE 'Barrett%'`
Selecting Rows

• Unless you specify otherwise, SQL*Plus displays the results of a query without eliminating duplicate rows
  • SELECT name
    FROM customers
    WHERE name LIKE ‘Grey%’;

• To eliminate duplicate rows in the result, include the DISTINCT keyword in the SELECT clause immediately after the command word
  • SELECT DISTINCT name
    FROM customers
    WHERE name LIKE ‘Grey%’
Selecting Rows

• The DISTINCT keyword applies to all columns in the SELECT column list. The result represents the distinct combination of the columns

  • SELECT DISTINCT name, id
  FROM customers
  WHERE name LIKE ‘Grey%’
Selecting Rows Summary

• Data is retrieved from a database using the SELECT statement.
• All commands are entered into the SQL buffer. The buffer contains one and only one command.
• The asterisk (*) selects all columns.
• Aliases are used to change columns headings.
• There are arithmetic, concatenation, and NULL value operators.
Selecting Rows Summary

• Each literal string is output once for each row returned.
• The DISTINCT keyword is used to eliminate duplicates.
SQL*Plus

- Execute the Start/Programs/Oracle for Windows NT/SQL Plus 3.3 program
- Enter User Name, Password, and Host
- Type “describe customers”
- Type “select name from customers where name like ‘Barrett%’”
- Examine the CUSTOMERS table on the SQL TRN data structure diagram
SQL*Plus

- The DESCRIBE (DESC) command displays the structure of a table including column names and datatypes.
  - desc customers.
- Column datatypes include:
  - Number\((a,b)\) – Number value having a maximum number of digits \(a\) and decimal places \(b\).
  - VARCHAR2\((z)\) – Variable length character value with a maximum size \(z\).
  - DATE – Date and time.
  - CHAR\((y)\) – Fixed length character value of size \(y\).
SQL*Plus

• SQL*Plus file commands include:
  – Save filename
  – Get filename
  – Start filename
  – @ filename
  – Edit filename
  – Spool filename
  – Exit
SQL*Plus

• To control the display of a column in a report use the COLUMN command. For example, you can change the heading, width, and format.

  • COLUMN name HEADING ‘Customer|Name’ FORMAT A70.

Select name from customers where name like ‘Barrett%’;
SQL*Plus

• The COLUMN command options include:
  – Clear – Clears column formats
  – Format – Changes the display of column data
  – Heading – Sets the column heading
  – Justify – Justifies the column heading
  – Noprint – Hides the column
  – Null – The text to be displayed for null values
  – Print – Shows the column
  – Truncated – Truncates the string at the end of the first line of display
  – Wrapped – Wraps the end of the string to the next line
  – Word_Wrapped – Same as Wrap but will not split words
SQL*Plus

- COLUMN command examples:
  - Create column headings
    - COLUMN name HEADING ‘Customer|Name’ FORMAT A70
    - COLUMN bact_balance JUSTIFY LEFT FORMAT $999,999,990.00
    - COLUMN inv_date FORMAT A9 NULL ‘No Invoice’
  - Display the current settings
    - COLUMN name
  - Clear Settings
    - COLUMN name CLEAR
SQL*Plus

• COLUMN Format Models
  – Character and Date columns
    • An Sets the display width of n
  – Number Columns
    • 9 Single zero-suppression digit
    • 0 Enforces leading zero
    • $ Floating dollar sign
    • L Local currency
    • . Position the decimal point
    • , Thousands separator
SQL*Plus Summary

• The DESCRIBE (DESC) command displays the structure of a table including column names and datatypes
• SQL*Plus includes a set of file commands
• To control the display of a column in a report use the COLUMN command
• Enter SQL*Plus and try the commands described in this section
Limiting Selected Rows

• The order of rows returned in a query is undefined. The ORDER BY clause may be used to sort the rows.
  • SELECT name, id, bact_balance
    FROM customers
    WHERE name LIKE 'Barrett%'
    ORDER BY name;

• ORDER BY clause options:
  – ASC – Ascending order (default).
  – DESC – Descending order.
Limiting Selected Rows

• Default Ordering of Data:
  – Numeric values are displayed with the lowest values first, example 1-999.
  – Date values are displayed with the earliest value first, for example 01-JAN-90 before 01-JAN-99.
  – Character values are displayed last for ascending sequences and first for descending sequences.
Limiting Selected Rows

• To restrict the rows returned from a query use the WHERE clause. A WHERE clause contains a condition that must be met, and directly follows the FROM clause.
  • SELECT name
    FROM customers
    WHERE name LIKE 'Barrett%';

• Character strings and dates must be enclosed in single quotation mark (‘’). Number constants, however, must not.
Limiting Selected Rows

• Comparison operators are divided into two categories: logical and SQL. They are used in the WHERE clause to compare one expression to another.
  
  • WHERE expression operator value

• Comparison Operators:
  – = Equal to
  – > Greater than
  – >= Greater than or equal to
  – < Less than
  – <= Less than or equal to
Limiting Selected Rows

- SQL Operators:
  - **BETWEEN … AND …** Between two values inclusive
    - WHERE Inv_date BETWEEN ’01-OCT-99’ AND ’22-OCT-99’
  - **IN (list)** Match any of a list of values
    - WHERE name IN (‘Barrett’,’Grey’,’Smith’)
  - **LIKE** Match a character pattern
    - WHERE name like ‘Barrett%’
  - **IS NULL** Is a null value
    - WHERE tax_certificate_id IS NULL
Limiting Selected Rows

• Logical Operators:
  – **AND**  If both component conditions return TRUE, then the result is TRUE.
  – **OR**   If either component condition returns TRUE, then the result is TRUE.
  – **NOT**  Returns the opposite condition.
Limiting Selected Rows

• By combining AND and OR operators in the same logical expression. The results of all of the conditions are combined in the order determined by the precedence of the connecting operators. Where operators are of equal precedence are used next to each other, they are performed from left to right. You may override precedence rules by placing part of the expression in parentheses.

• Precedence is as follows: Comparison operators, AND, then OR.
  • `WHERE name LIKE 'Barrett%' AND tax_certificate_id IS NULL`
  • `WHERE inv_date = '22-OCT-99' OR inv_balance < 9999.99`
Limiting Selected Rows

Summary

• The ORDER BY clause is used to override the default order of displayed data
• The WHERE clause is used in conjunction with logical and SQL operators to restrict the results of a query
• Execute SQL*Plus and query data using the ORDER BY and WHERE clauses
Retrieving Data from Multiple Tables

• A join is used to query data from more than one table

• Rows are joined using common values existing in corresponding columns
  • Customers.id = Clients.cust_id

• Join methods:
  – Equijoin
  – Non-equijoin
  – Outer join
  – Self join
  – Set operators
Retrieving Data from Multiple Tables

Caution

• When a join condition is invalid or omitted completely, the result is a **Cartesian Product**, in which all combinations of rows will be displayed. All rows of the first table are joined with the rows in the second table.
Retrieving Data from Multiple Tables

• An *Equijoin* is created when the relationship between two table values are equal.

  ```sql
  SELECT customer.name, client.name,
          client.commission_rate
  FROM    customers, clients
  WHERE customers.id = clients.cust_id
  AND      customer.name LIKE 'Barrett%';
  ```

• Every customer that has a client is displayed including the client name and commission_rate.
Retrieving Data from Multiple Tables

• You need to qualify the names of the columns in the WHERE clause with the table name to avoid ambiguity. Without the table prefix the name column could be from either clients or customers table.

• If there are no column names that are the same between tables, then there is no need to qualify the columns.
Retrieving Data from Multiple Tables

• You need to qualify the names of the columns in the WHERE clause with the table name to avoid ambiguity. Without the table prefix the name column could be from either clients or customers table.

• If there are no column names that are the same between tables, then there is no need to qualify the columns.
Retrieving Data from Multiple Tables

• You need to qualify the names of the columns in the WHERE clause with the table name to avoid ambiguity. Without the table prefix the name column could be from either clients or customers table.

• If there are no column names that are the same between tables, then there is no need to qualify the columns.
Retrieving Data from Multiple Tables

- Table alias can be defined to keep your SQL statement smaller and avoid ambiguity. Like column aliases, table aliases are a method of giving the table another name for the purpose of the SELECT statement.

  - Before:
    ```sql
    SELECT customer.name, client.name, client.commission_rate FROM customers, clients
    WHERE customers.id = clients.cust_id
    AND customer.name LIKE 'Barrett%';
    ```

  - After:
    ```sql
    SELECT cus.name, cl.name, cl.commission_rate FROM customers cus, clients cl
    WHERE cu.id = cl.cust_id
    AND cus.name LIKE 'Barrett%';
    ```
Retrieving Data from Multiple Tables

- A Non-**Equijoin** results when no column value in one table corresponds directly to a column value in the second table.
  
  - SELECT c.name, c.id
    FROM customers c, billing_accounts b, invoices i
    WHERE c.id = b.cust_id
    AND b.id = i.bact_id
    AND i.bact_balance > 9999999.99;

- If a row does not satisfy a join condition, then the row will not appear in the query resulting in a “no rows selected” message.
Retrieving Data from Multiple Tables

- Rows not displayed by a *Equijoin* or *Non-Equijoin* condition can be displayed if an *Outer Join* operator is used in the join condition. The outer join operator is a plus sign enclosed in parentheses “(+)

- **SELECT count(*)**
  FROM customers c, billing_accounts b, addresses a
  WHERE c.id = b.cust_id
  AND b.addr_id_taxed_by a.id;

  *Returns 1680 rows*

- **SELECT count(*)**
  FROM customers c, billing_accounts b, addresses a
  WHERE c.id = b.cust_id
  AND b.addr_id_taxed_by a.id(+);

  *Returns 9906 rows*
Retrieving Data from Multiple Tables

• *Outer Join* restrictions:
  
  – The outer join operator can only appear on one side of the expression – the side that has the information missing. It returns those rows from one table which have no direct match in the other.
  
  – A condition involving an outer join may not use the IN operator or be linked to another condition by the OR operator.
Retrieving Data from Multiple Tables

- **Self Joins** link rows in a table to rows in the same table. You must simulate two tables in the FROM clause by creating two aliases for the table.
  - SELECT c.name, b.name, o1.name, o2.name
    FROM customers c, billing_accounts b, orders o1, orders o2
    WHERE c.id = b.cust_id
    AND    b.id = o1.bact_id
    AND    o1.id = o2.from_ord_id;
Retrieving Data from Multiple Tables

Summary

• You can use multiple methods to join tables:
  – Equijoin
  – Non-Equijoin
  – Outer join
  – Self join

• You can create a Cartesian product if you omit the WHERE clause.

• Use aliases to avoid ambiguity.

• Execute SQL*Plus and try the example queries.
Single Row Functions

• Single Row functions are used to manipulate data items. The accept one or more arguments and return one or value for each row returned by the query.

• Single Row function features:
  – Manipulate data items
  – Accept arguments and return one value
  – Act on each row returned
  – Return one result per row
  – Modify the datatype
  – Can be nested
Single Row Functions

• Character Functions:
  x = column name or expression
  - Lower – Converts to lowercase
   \text{LOWER}(x)
  - Upper – Converts to uppercase
   \text{UPPER}(x)
  - InitCap – Converts to initial capitalization
   \text{INITCAP}(x)
  - Concat – Concatenates values
   \text{CONCAT}(x,y) \text{ same as } “x||y”
Single Row Functions

- **Substr** – *Returns substring*
  \[
  \text{SUBSTR}(x, m, n)
  \]
  \[m = \text{starting position}\]
  \[n = \text{length of returned string}\]

- **Length** – *Returns number of characters*
  \[
  \text{LENGTH}(x)
  \]

- **NVL** – *Converts a null*
  \[
  \text{NVL}(x, z)
  \]
  \[z = \text{value returned if } x \text{ is null}\]
Single Row Functions

• Number Functions:
  – **Round** – *Rounds value to specified decimal*
    \[
    \text{ROUND}(x, n)
    \]
    \(n = \text{number of decimal places}\)
  – **Trunc** – *Truncates value to specified decimal*
    \[
    \text{TRUNC}(x, n)
    \]
    \(n = \text{number of decimal places}\)
  – **Mod** – *Returns remainder of division*
    \[
    \text{MOD}(a, b)
    \]
    equivalent to \(a / b\)
Single Row Functions

- Oracle stores dates in an internal numeric format including century, month, day, hours, minutes, and seconds.
- Default date display is DD-MON-YY.
- SYSDATE is a function returning date and time.
- DUAL is a dummy table used to view SYSDATE.
Single Row Functions

• You can perform calculations using arithmetic operators with dates.
• For example:
  – Add or subtract a number to or from a date for resultant date value.
  – Subtract two dates to find the number of days between those dates.
  – Add hours to a date by dividing the number of hours by 24.
Single Row Functions

• Date Functions:
  – **Months_Between** – *Number of months between two dates*
    MONTHS_BETWEEN(x,y)
  – **Add_Months** – *Add calendar months to a date*
    ADD_MONTHS(x,n)
    n = # of months
  – **Next_Day** – *Next day of the date specified*
    NEXT_DAY(x,’d’)
    d = day of the week ‘MONDAY’
  – **Last_Day** – *Last day of the month*
    LAST_DAY(x)
Single Row Functions

- **Round** – *Round to date at midnight*
  \[
  \text{ROUND}(x, \text{’fmt’})
  \]
  \[fmt = \text{date format}\]

- **Trunc** – *Remove time portion from date*
  \[
  \text{TRUNC}(x, \text{’fmt’})
  \]
  \[fmt = \text{date format}\]

- **Example date formats include:**
  - mm/dd/yy or mm/dd/yyyy
  - hh24:mi:ss or hh24:mi:ss AM
Single Row Functions

- SQL provides three functions to convert a value from one datatype to another.
- Conversion Functions:
  - To_Char – Converts a number or date to a character string
    TO_CHAR(x,'fmt')
  - To_Number – Converts a character string to a number
    TO_NUMBER(x)
  - To_Date – Converts a character string of a date to a date value
Single Row Functions Summary

- Single row functions work on character, number, and date data.
- Conversion functions are `TO_CHAR`, `TO_DATE`, and `TO_NUMBER`.
- `SYSDATE` is a pseudo-column used to return current date and time.
- `DUAL` is a dummy table in the database.
- Enter SQL*Plus and create queries using Single Row functions.
Group Functions

• Unlike Single Row functions, Group Functions operate on sets or row to give one result per group.

• By default, all the rows in a table are treated as one group. Use the GROUP BY clause in the SELECT statement to divide rows into smaller groups.
Group Functions

• Group functions:
  “x” can be a column name or expression. DISTINCT can be used in conjunction with “x” for example COUNT(DISTINCT(x)). Any datatype may be used.

  – Avg – Average value of x, ignoring null values
    AVG(x)
  – Count – Number of rows where x evaluates to something other than null
    COUNT(x)

  Use COUNT(*) to count all selected rows including nulls.
Group Functions

- Max – *Maximum value of x*
  \[ \text{MAX}(x) \]
- Min – *Minimum value of x*
  \[ \text{MIN}(x) \]
- Stddev – *Standard deviation of x, ignoring nulls*
  \[ \text{STDDEV}(x) \]
- Sum – *Sum of values of x, ignoring nulls*
  \[ \text{SUM}(x) \]
- Variance – *Variance of x, ignoring nulls*
  \[ \text{VARIANCE}(x) \]
Group Functions

• The COUNT function has two formats: COUNT(*) and COUNT(x). COUNT(*) returns the number of rows in a table, including duplicate rows and rows containing null values.
  • SELECT COUNT(*) FROM customers;
  • SELECT COUNT(name) FROM customers
      WHERE name like ‘Barrett%’;
Group Functions

- You can use the GROUP BY clause to divide the rows in a table into smaller groups. You can then use the group functions to return summary information for each group.
Group Functions

- GROUP BY guidelines:
  - If you include a group function in a SELECT clause, you cannot select individual results as well unless the individual column appears in the GROUP BY clause.
  - Using a WHERE clause, you can pre-exclude rows before dividing them into groups.
  - You must include the columns in the GROUP BY clause.
  - You cannot use the positional notation or column alias in the GROUP BY clause.
  - By default, rows are sorted by ascending order of the GROUP BY list. The ORDER BY clause is used to override the default.
Group Functions

- You use the HAVING clause to specify which groups are to be displayed. Therefore, you further restrict the groups on the basis of aggregate information.
- When using the HAVING clause the following steps are performed.
  - Rows are grouped
  - The group function is applied
  - The groups that match the criteria in the HAVING clause are displayed
Group Functions

• GROUP By examples:
  – SELECT cust_id, count(*)
    FROM billing_accounts
    GROUP BY cust_id HAVING count(*) > 20;
  – SELECT cust_id, name
    FROM billing_accounts
    WHERE cust_id = 1000200110
    GROUP BY cust_id,name;
  – SELECT cust_id, name
    FROM billing_accounts
    WHERE cust_id = 1000200110
    GROUP BY cust_id,name,id;
Group Functions Summary

• There are seven group functions available in SQL.
• You can create subgroups by using the GROUP BY clause.
• Groups can be excluded by using the HAVING clause.
• Place the GROUP BY and HAVING clauses after the WHERE clause and before the ORDER BY clause.
Billing Engine Examples

- Show any billing account where an invoice last status is ‘CONFIRMED’.
  - Execute SQL*Plus
  - Type “describe billing_accounts”
  - Type “desc invoices”
  - Type “SELECT ?????
    FROM   billing_accounts bact, invoices inv
    WHERE  bact.id = inv.bact_id
    AND     inv.status_code = ‘CONFIRMED’;”
  - No CONFIRMED status invoices so substitute RENDERED and execute the query
Billing Engine Examples

• Show any billing accounts which have a “CDDS” alias.
  – Execute SQL*Plus
  – Type “describe billing_accounts”
  – Type “desc bactaliases”
  – Type “SELECT ??? FROM billing_accounts bact, bact_aliases bal WHERE bact.id = bal.bact_id AND bal.atype_code = ‘CDDS’;”
Billing Engine Examples

• Show any billing accounts which have a product “715P”.
  – Execute SQL*Plus
  – Type “describe billing_accounts”
  – Type “desc orders”
  – Type “SELECT ???
    FROM   billing_accounts bact, orders ord
    WHERE bact.id = ord.bact_id
    AND       substr(ord.name,1,3) = ‘715’
    AND       bact.rentyp_id > 2000009;”
  – No ‘715’ orders so remove criteria and execute the query.
Summary

• What have we learned?
  – Relational database approach, concepts, terminology, and operators.
  – How to retrieve data from database structures using SQL.
  – How to use SQL*Plus.
  – Examined the structure supporting a significant portion of the Billing Engine.
Where to Get More Information

- ORACLE corporation offers training sessions
- ORACLE publications:
  - SQL*Plus Quick Reference
  - SQL*Plus User’s Guide and Reference