The objectives of this assignment are to reacquaint you with the basics of data manipulation in a relational database through SQL\textsuperscript{1}, and to (re)familiarize with Oracle\textsuperscript{9i} and SQL Plus. You will need to create a user in SQL Plus, download and uncompress H1.exe to extract the H1.dmp dump file, import the CDs, MUSIC\_CATEGORIES and RECORD\_LABELS tables (contained in H1.dmp) using the IMP utility, and build 10 queries in SQL Plus. Refer to the video presentation posted on (the “videos” link):

www.csus.edu/indiv/c/chingr/oracle/indexorcl.htm

To receive credit for this assignment, you will need to capture your session in a spool file and print its contents.

**Suggestion.** Plan ahead, and read chapters 13 and 14 of the Database Systems textbook\textsuperscript{2}, or any other book with a discussion of various SQL commands before attempting this assignment. Also, review the Oracle\textsuperscript{9i} Introduction Screen Cam for importing tables through the IMP utility and using SQL Plus.

**Creating a User**

Create a user in SQL Plus. Be sure to give your user privileges to create and import tables (a DBA role is sufficient). Refer to Figure 1 for the syntax.

```
SQL> create user mis180 identified by mis180;
User created.
SQL> grant dba to mis180;
Grant succeeded.
```

**Figure 1.** Creating a user in SQL Plus. User name = mis180, password = mis180.

\textsuperscript{1}These were introduced in MIS 150.

\textsuperscript{2}MIS 150 required textbook.
Importing the Table

Import the CDs, MUSIC_CATEGORIES and RECORD_LABELS tables from the H1.dmp file using Oracle’s IMP utility. Both the tables’ structures and data will be inserted into the database.

Create a DOS window and change to the drive/directory containing the dump (dmp) file (cd:\temp). The general syntax to launch IMP is as follows:

```
IMP user-name/password
```

where: user-name and password The user name and password assigned to your user. Be sure to separate the name and password with the “/”.

Follow the prompts and make the required entries. Refer to the Screen Cam video for a demonstration.

You can view the table’s definition in SQL Plus through the DESCRIBE command:

```
describe table-name
```

Queries

Now that all tables contain data, you can perform the following queries. The details of the queries (i.e., predicate, column names) are of your own doing. Thus, it would be uncanny for two assignments to contain identical entries.

**Note.** When designing your queries, be sure they convey usable information. Queries that are comprised of a nonsensical assembly of columns and/or rows will not awarded full points. For example, constructing a query that only lists prices or retrieves no rows is not useful information. Please apply your common sense! This isn’t just an exercise, it’s an investment in your future!

Carefully read the relational algebra!

1. Projection (Π) and Selection (σ). Perform a project on your selection on the CDs table. The number of rows returned by your query should not exceed 15.

```
Π_{column-list}(σ_{predicate}(CDs))
```
2. Equi-join. Expand your query in (1) and perform an equi-join with the music_categories or record_labels table. The number of rows returned by your query should not exceed 15.

\[ \Pi_{\text{column-list}} (\sigma_{\text{predicate}} (\text{CDs})) \Join <\text{CDs.column-name}=\text{music_categories.column-name} \Pi_{\text{column-list}} (\text{music_categories}) \]

\[ \text{or} \]

\[ \Pi_{\text{column-list}} (\sigma_{\text{predicate}} (\text{CDs})) \Join <\text{CDs.column-name}=\text{record_labels.column-name} \Pi_{\text{column-list}} (\text{record_labels}) \]

Note. \( \Join < \) denotes equi-join.

3. Equi-join and Range. Perform another join between the CDs table and either the record_labels or music_categories table. Incorporate into your predicate a condition using the BETWEEN clause. The number of rows returned by the query should not exceed 15.

\[ \Pi_{\text{column-list}} (\sigma_{\text{predicate}} (\text{CDs})) \Join <\text{CDs.column-name}=\text{music_categories.column-name} \Pi_{\text{column-list}} (\text{music_categories}) \]

\[ \text{or} \]

\[ \Pi_{\text{column-list}} (\sigma_{\text{predicate}} (\text{CDs})) \Join <\text{CDs.column-name}=\text{record_labels.column-name} \Pi_{\text{column-list}} (\text{record_labels}) \]

4. Equi-join and Pattern Matching. Replace the range in (3) with the LIKE. Include wildcards and the UPPER or LOWER functions. The number of rows returned by the query should not exceed 15.

\[ \Pi_{\text{column-list}} (\sigma_{\text{predicate}} (\text{CDs})) \Join <\text{CDs.column-name}=\text{music_categories.column-name} \Pi_{\text{column-list}} (\text{music_categories}) \]

\[ \text{or} \]

\[ \Pi_{\text{column-list}} (\sigma_{\text{predicate}} (\text{CDs})) \Join <\text{CDs.column-name}=\text{record_labels.column-name} \Pi_{\text{column-list}} (\text{record_labels}) \]

Note. Since pattern matching applies to string or character manipulation, it should be performed ONLY on character data, and applied in the predicate (i.e., WHERE). In Oracle, use the % character for an all wildcard, _ (underscore) character for a single character wildcard, and || (double vertical bars) to append characters and character strings. Be sure to enclose characters and character strings in single quotes, and the column matches the same case as your character string.

Example:

WHERE LOWER(product_description) LIKE ‘%’ || ‘color’ || ‘%’

LOWER changes the text in product_description to lower case to ensure differences in case do not affect the search (i.e., ‘color’ ≠ ‘Color’). The LIKE will return all rows with the character string ‘color’ appearing anywhere in the product description. (Note. Upper case used to add emphasis.)
5. Equi-join and IN (Specific Categories). Replace the LIKE in (4) with the IN (e.g., where \textit{column-name} IN \textit{(value1, value2, ...)}). The number of values should exceed 1. The number of rows returned by the query should not exceed 15.

\[
\Pi_{\text{column-list}}(\sigma_{\text{predicate}}(\text{CDs})) \bowtie \text{CDs.column-name=music_categories.column-name} \Pi_{\text{column-list}}(\text{music_categories})
\]

or

\[
\Pi_{\text{column-list}}(\sigma_{\text{predicate}}(\text{CDs})) \bowtie \text{CDs.column-name=record_labels.column-name} \Pi_{\text{column-list}}(\text{record_labels})
\]

6. Equi-join, Aggregation and GROUP BY / HAVING. Perform a query using the COUNT, SUM, AVG, MIN and MAX aggregation functions that are applied over selected groups of entities (i.e., rows sharing a common attribute value). Apply an alias to the functions (in the column list). The number of rows returned by the query should exceed five (i.e., six or more).

\[
\Pi_{\text{column-list}}(\sigma_{\text{predicate}}(\text{CDs})) \bowtie \text{CDs.column-name=music_categories.column-name} \Pi_{\text{column-list}}(\text{music_categories})
\]

or

\[
\Pi_{\text{column-list}}(\sigma_{\text{predicate}}(\text{CDs})) \bowtie \text{CDs.column-name=record_labels.column-name} \Pi_{\text{column-list}}(\text{record_labels})
\]

\textbf{Alias Example:}

\[
\text{SELECT (sum(retail_price * quantity) * tax_rate) as total_price, ...}
\]

The result of the calculation will appear under the alias (column name) ‘total_price.’ AS is optional syntax.

7. Equi-join and Calculation. Perform a query that includes a calculation on two columns. Include a join between CDs and either record_labels or music_categories. Apply a format mask to the output (refer to posted handout for the different masks), and an alias to the calculation. The number of rows returned by the query should not exceed 15.

\[
\Pi_{\text{column-list}}(\sigma_{\text{predicate}}(\text{CDs})) \bowtie \text{CDs.column-name=music_categories.column-name} \Pi_{\text{column-list}}(\text{music_categories})
\]

or

\[
\Pi_{\text{column-list}}(\sigma_{\text{predicate}}(\text{CDs})) \bowtie \text{CDs.column-name=record_labels.column-name} \Pi_{\text{column-list}}(\text{record_labels})
\]

To apply a format mask, the numeric or date column, or calculation must be converted to character with \textit{to_char} and a valid mask must be specified. The basic syntax appears below:

\[
\text{TO_CHAR(\{ column-name | calculation \}, \textit{mask}')}
\]

\textbf{Example:}

\[
\text{to_char(sys_date - 90,'fmMonth dd, yyyy')}
\]
“to_char” converts the binary system date (sys_date) less 90 days (i.e., 90 days prior to today’s date) to a character string specified by the format mask, ‘Month dd, yyyy.’ “fm” truncates all trailing spaces.

8. View. A view using a projection on a selection with an equi-join between the CDs and music_categories or record_labels tables. The number of columns specified in the projection for the view is at your discretion. However, it (projection) should include at least (i.e., a minimum of) two common columns.

view-name =

\[ \Pi_{\text{column-list}}(\sigma_{\text{predicate}}(\text{CDs})) \bowtie \text{CDs.column-name=music_categories.column-name} \Pi_{\text{column-list}}(\text{music_categories}) \]

or

\[ \Pi_{\text{column-list}}(\sigma_{\text{predicate}}(\text{CDs})) \bowtie \text{CDs.column-name=record_labels.column-name} \Pi_{\text{column-list}}(\text{record_labels}) \]

The number of rows produced in the view should not exceed 15. Display the contents (all columns and rows) of your view.

9. ANY/ALL with a Subquery. Perform a subquery that incorporates either ANY or ALL. Use either CDs, music_categories, record_labels and/or your views in (9). Explain the purpose of your subquery and the information it presents next to your output.

```
SELECT column-list-1 FROM {table-1 | view-1}
WHERE column-name-1 {relational operator} {ANY | ALL}
(SELECT column-list-2 FROM {table-2 | view-2}
WHERE condition)
```

10. EXIST/NOT EXIST with a Subquery. Following the table specification in (9), create a EXIST/NOT EXIST in your subquery. Explain the purpose of your subquery and the information it presents next to your output.

```
SELECT column-list-1 FROM {table-1 | view-1}
WHERE {EXISTS | NOT EXISTS}
(SELECT {* | column-list-2} FROM {table-2 | view-2}
WHERE condition)
```

Tangibles

To received credit for this assignment, submit a printed copy of your spool file in a 9 x 12-inch manilla envelope; a diskette is NOT required. The listing should include queries (1) through (10). Tab the pages, highlight the SQL command (with a marker), and write the number corresponding to the requirement next
to the query; this indicates the *attempt* you want graded. Credit cannot be awarded for assumed work. Unmarked work will not be graded.

Use a text editor, such as Note Pad or Word Pad, to print your spool file.

**Note.** Do not worry about errors in your spool file listing. This provides you with more evidence that the work is yours!

Please also remember the *one assignment, one grade* rule. Any assignments that have an uncanny resemblance will be considered a violation of ethical class behavior.

Assign your spool file a name of 8 characters or less (excluding the extension). Anything greater will cause problems (i.e., nothing will be recorded).

**Acceptable:** h1homework.lst

**Not Acceptable:** h1homework.lst