CALIFORNIA STATE UNIVERSITY, SACRAMENTO
College of Business Administration

MIS 155 - 4GL Application Development

SQL and Oracle9i

Points: 30 Due: Wednesday, February 1

The objectives of this assignment are to reacquaint you with the basics of SQL[1] and refamiliarize with Oracle9i and SQL Plus. You will need to create a user in Oracle9i (through SQL Plus), import the import the CDs, MUSIC_CATEGORIES and RECORD_LABELS tables (contained in H1.dmp) using the IMP utility, and execute 10 queries in SQL Plus.

To receive credit for this assignment, you will need to capture your session(s) in a spool file and print its contents. Be sure the file does not exceed 8 characters, is assigned (written) to a file in a folder and not the desktop, and does not share its name with another file.

Suggestion. Plan ahead, and review your MIS 150 textbook, or any other book with a discussion of various SQL commands before attempting this assignment. Also, review the Oracle9i Introduction Camtasia video for creating a user and tables, loading data into a table through SQL Loader (SQLLDR), importing a table through the IMP utility, and using SQL Plus (http://www.csus.edu/indiv/c/chingr/ oracle/indexorcl.htm).

Creating a User

Create a user (use any name and password) in SQL Plus. Be sure to grant your user privileges to create and import tables (a DBA role is sufficient).

```
CREATE USER user-name IDENTIFIED BY password;

GRANT DBA TO user-name;
```

Where…	*user-name*	Your user name
	password	Your user's password

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 (Start → Run… → command) and change to the drive/directory containing the dump (dmp) file (e.g., cd:\temp). The general syntax to launch IMP is as follows:

[1]These were introduced in MIS 150 and 211.

`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 Camtasia video for a demonstration.

To close the command window, enter "exit."

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

`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!*[2] *This isn't just an exercise, it's an investment in your future!*

1. Projection (Π) and Selection (σ). Perform a projection on a selection on the CDs table. The number of rows returned by your query should not exceed 15, but be more than 3.

$$\Pi_{column\text{-}list}(\sigma_{predicate}(CDs))$$

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, but be more than 3.

$$\Pi_{column\text{-}list}(\sigma_{\ predicate}(CDs)) >\!\!< {}_{CDs.column\text{-}name=music\_categories.column\text{-}name}\ \Pi_{column\text{-}list}(music\_categories)$$

or

$$\Pi_{column\text{-}list}(\sigma_{\ predicate}(CDs)) >\!\!< {}_{CDs.column\text{-}name=record\_labels.column\text{-}name}\ \Pi_{column\text{-}list}(record\_labels)$$

Note. >< mean 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

[2] When in doubt, ask someone else to make sense of the results.

the BETWEEN clause. The number of rows returned by the query should not exceed 15, but be more than 3.

$$\Pi_{column\text{-}list}(\sigma_{\ predicate}(CDs)) >\!\!< _{CDs.column\text{-}name=music\_categories.column\text{-}name}\ \Pi_{column\text{-}list}(music\_categories)$$
or
$$\Pi_{column\text{-}list}(\sigma_{\ predicate}(CDs)) >\!\!< _{CDs.column\text{-}name=record\_labels.column\text{-}name}\ \Pi_{column\text{-}list}(record\_labels)$$

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, but be greater than 2.

$$\Pi_{column\text{-}list}(\sigma_{\ predicate}(CDs)) >\!\!< _{CDs.column\text{-}name=music\_categories.column\text{-}name}\ \Pi_{column\text{-}list}(music\_categories)$$
or
$$\Pi_{column\text{-}list}(\sigma_{\ predicate}(CDs)) >\!\!< _{CDs.column\text{-}name=record\_labels.column\text{-}name}\ \Pi_{column\text{-}list}(record\_labels)$$

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, the wildcard is represented by the percent sign (%) or underscore (_) for one character.

For example…

```
WHERE UPPER(column-name) LIKE UPPER('%'||'word'||'%')
```

5. Equi-join and IN (Specific Categories). Replace the LIKE in (4) with the IN (e.g., where *column-name* IN (*value1*, *value2*, ...)). The number of values should exceed 1. The number of rows returned by the query should not exceed 15, but be greater than 2.

$$\Pi_{column\text{-}list}(\sigma_{\ predicate}(CDs)) >\!\!< _{CDs.column\text{-}name=music\_categories.column\text{-}name}\ \Pi_{column\text{-}list}(music\_categories)$$
or
$$\Pi_{column\text{-}list}(\sigma_{\ predicate}(CDs)) >\!\!< _{CDs.column\text{-}name=record\_labels.column\text{-}name}\ \Pi_{column\text{-}list}(record\_labels)$$

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), but be greater than 2.

$$\Pi_{column\text{-}list}(\sigma_{\ predicate}(CDs)) >\!\!< _{CDs.column\text{-}name=music\_categories.column\text{-}name}\ \Pi_{column\text{-}list}(music\_categories)$$
or
$$\Pi_{column\text{-}list}(\sigma_{\ predicate}(CDs)) >\!\!< _{CDs.column\text{-}name=record\_labels.column\text{-}name}\ \Pi_{column\text{-}list}(record\_labels)$$

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, and an alias to the calculation. The number of rows returned by the query should not exceed 15, but be greater than 2.

$$\Pi_{column\text{-}list}(\sigma_{predicate}(CDs)) \gtrless_{CDs.column\_name=music\_categories.column\text{-}name} \Pi_{column\text{-}list}(music\_categories)$$
or
$$\Pi_{column\text{-}list}(\sigma_{predicate}(CDs)) \gtrless_{CDs.column\text{-}name=record\_labels.column\text{-}name} \Pi_{column\text{-}list}(record\_labels)$$

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

TO_CHAR({ *column-name* | *calcuation* }, *'mask'*)

For example…

to_char(sys_date - 90,'fmMonth dd, yyyy')

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 =
$$\Pi_{column\text{-}list}(\sigma_{predicate}(CDs)) \gtrless_{CDs.column\text{-}name=music\_categories.column\text{-}name} \Pi_{column\text{-}list}(music\_categories)$$
or
$$\Pi_{column\text{-}list}(\sigma_{predicate}(CDs)) \gtrless_{CDs.column\text{-}name=record\_labels.column\text{-}name} \Pi_{column\text{-}list}(record\_labels)$$

The number of rows produced in the view should not exceed 15, but be greater than 4. Display the contents (all columns and rows) of your view with the SELECT * FROM *view-name.*

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 receive credit for this assignment, submit a printed copy of your spool file in a 9 × 12-inch manila envelope; a diskette is NOT required. The listing should include queries (1) through (10). Tab the pages[3], 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.

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

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

SQL Commands to Create Selected Database Objects

Dropping a Table or User

```
DROP {USER | TABLE} user-name;
```

For example…

DROP USER MIS;

DROP TABLE CDS;

Permanently Removing Deleted Database Objects

`COMMIT;` Permanently removes all dropped or deleted objects from the database. This should only be used after you are certain you want to take this action since the objects cannot be recovered with the ROLLBACK command.

[3] You may use post-its as tabs.