EXERCISE QUESTIONS

True/False:

1. An inner subquery may not use the ORDER BY clause.
2. The inner subquery does not have to be a SELECT statement.
3. A subquery may not be nested to more than three levels.
4. Operators ANY and SOME are the same.
5. The = ALL operator does not return any rows in most cases.
6. Top-N analysis uses ROWID attribute for row numbers.
7. An inline view may use the ORDER BY clause.
8. In correlated subqueries, the inner query can reference a column from the outer query.
9. = ANY means IN.
10. >ALL means greater than the minimum value.
11. A table can be created with a subquery and an INSERT statement.
12. A table can be created based on another table with a subquery.
13. When a table is created with a subquery, it inherits all constraints from the original tabl

Define the Following Terms, and Give One Example of Each:

1. Single-row subquery.
2. Multiple-row subquery.
3. Inline view.
4. Top-N query.
5. Correlated subquery.

Answer the Following Questions:

1. State the various uses of a subquery.
2. In which situations would you use a three-level subquery?
3. What constraints are transferred to the newly created table with a subquery?
4. How does a Top-N query work?
5. What is the use of the INSERT ALL and INSERT FIRST statements?
6. Why is the MERGE statement useful?
7. A subquery returns three values: 35000, 45000, and 55000. The outer query has a cond that tests a value of 40000 against these values. If the =ANY, >ANY, <ALL, >AL <>ALL operator is used, when will the value 40000 satisfy the condition? Use each ator separately.

LAB ACTIVITY

1 design the following subqueries. (U

LAB ACTIVITY

1. Use the N2 Corporation database tables to design the following subqueries. (Use the spooling method to capture all queries and results in the CHAP8SP1.LST file.)
   (a) Display employee Jinku Shaw's department name.
   (b) Find the name of the supervisor for employee number 433.

(c) Who has the same qualification as Stanley Garner?

(d) Which department has more employees than Department 20?

(e) Which employees have been working in the company longer than Larry Houston?

(f) Find all employees in the Sales Department by using a nested query.

(g) Create a new table, EMP30, and populate it with employees in Department 30 by using an existing table and a subquery. Use EmployeeId, Lname, Fname, HireDate, and Salary columns.

(h) Add more rows to the EMP30 table with employees in Department 40. Do not transfer the employee's salary.

(i) Update the salary of newly transferred employees from the EMPLOYEE table to the EMP30 table with a MERGE statement, and INSERT employees who are not in the EMP30 table.

(j) Find employees with the minimum salary in their own department with the use of a correlated subquery.

(k) Use a multiple-level subquery to display dependent information for employees who belong to the FINANCE department.

(l) Use set operator and subquery to find employees who do not have any dependents.

(m) Write a subquery that finds the average salary by each department. Check to find if employee 543's salary satisfies the =ANY, <ANY, >ANY, <ALL, or >ALL condition against those departmental average salaries.

2. Use the IU College database tables to design the following subqueries. (Use the spooling method to capture all queries and results in the CHAP8SP2.LST file)

(a) Display student Jose Diaz's faculty advisor's name and phone number.

(b) Find the rooms with the bottom-two capacities. Do not include office rooms.

(c) Find the Spring 2003 course sections with the top-three maximum count numbers.

(d) Find all information regarding classrooms (RoomType = 'C').

(e) Create a new table, SP03SECT, for Spring 2003 semester course sections by using a subquery. Include CourseId, Section, FacultyId, and RoomId columns only.

(f) Delete rows from the SP03SECT table for faculty member Mobley.

(g) Find faculty members who do not teach any course in the Spring 2003 semester. Use a correlated subquery with a NOT EXISTS operator on the SP03SECT table.

Simple View	Complex View
It is based on one table.	It is based on one or more tables.
It does not contain group functions.	It may contain group functions.
It does not contain grouped data.	It may contain grouped data.
Data manipulation is always possible.	Data manipulation is not always possible.

Figure 9-1 Types of views.

Creating a View

The general syntax is

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW viewname
[column aliases]
AS SELECT-subquery
[WITH CHECK OPTION [CONSTRAINT constraintname]]
[WITH READ ONLY];
```

A view is created with a SELECT subquery. The subquery cannot use an ORDER BY clause, but a view can.

In the syntax, OR REPLACE replaces an existing view with the same name, if it already exists. The FORCE option creates a view even if the underlying table does not yet exist. The default is NOFORCE, which does not create a view if the underlying table does not exist. The column aliases are used for the columns selected by the subquery. The number of aliases must match the number of columns selected by the subquery. The SELECT subquery can use all clauses except the ORDER BY clause. The WITH CHECK OPTION applies to the WHERE clause condition in the subquery. It allows insertion and updating of rows based on the condition that satisfies the view. The CHECK OPTION can also be given an optional constraint

name. The WITH READ ONLY option is to make sure that the data in the underlying table are not changed through the view.

Figure 9-2 shows creation of a simple view. View STU500_VU is based on the STUDENT table for students with MajorId = 500. The column aliases are given for all

```
SQL> CREATE VIEW stud500_vu
  2  (StuId, LastName, FirstName, Advisor, MajNum)
  3  AS SELECT StudentId, Last, First, FacultyId, MajorId
  4     FROM student
  5     WHERE MajorId = 500
  6  WITH CHECK OPTION;

View created.

SQL>
```

Figure 9-2 Creating a simple view.

four columns selected by the subquery. The user with access to the view can use it like any other table. The user does not even have to know the existence of the STUDENT table and other columns in it. The user only sees what you let him or her see! Now, Figure 9-3 shows use of the newly created view like a table with DESCRIBE and SELECT.

In Figure 9-4, you will see a complex view that is created from two tables, EMPLOYEE and DEPT. It uses group functions and derived data. It is not possible to

```
SQL> set linesize 50
SQL> DESCRIBE stu500_vu
Name                      Null?    Type
------------------------  -------- ------------
STUID                     NOT NULL CHAR(5)
LASTNAME                  NOT NULL VARCHAR2(15)
FIRSTNAME                 NOT NULL VARCHAR2(15)
ADVISOR                            NUMBER(3)
MAJNUM                             NUMBER(3)

SQL> set linesize 100
SQL> SELECT *
  2  FROM stu500_vu;

STUID LASTNAME        FIRSTNAME       ADVISOR    MAJNUM
----- --------------- --------------- ---------- ----------
00101 Tyler           Mickey                 555        500
00103 Rickles         Deborah                555        500

SQL>
```

Figure 9-3 Using a view like a table.

Removing a View

A user who owns a view—or who has the privilege to remove it—can remove a vie
The removal of a view does not affect data in the underlying table. When a view is r
moved, a "View dropped" message is displayed (see Fig. 9-6). The general syntax

```
SQL> SELECT VIEW_NAME FROM USER_VIEWS;

VIEW_NAME
------------------------------

DEPTSALVU
DEPT_SAL_VU
STU500_VU

SQL> DROP VIEW deptsalvu;

View dropped.

SQL>
```

Figure 9-6 Listing and dropping view.

DROP VIEW viewname;

For example,

```
DROP VIEW stuvu500;
```

INDEX

An index is another Oracle object that is used for faster retrieval of rows from a table. An index can be created explicitly by using the CREATE INDEX statement

or implicitly by Oracle. Once an index exists for a table, the user does not have to open or use the index with a command or a statement. The Oracle server uses the index to search for a row rather than scanning through the entire table. Indexing reduces both search time and disk input/output. All indexes are maintained separately from the table on which they are based. Creating and removing an index does not affect the table at all. When a table is dropped, all indexes based on that table are also removed.

Implicit indexes are created when the PRIMARY KEY or UNIQUE constraint is defined. Such indexes get the name of the constraint. A user can create explicit indexes based on non–primary key or nonunique columns or on any combination of columns for faster table access. An index based on a combination of columns is called a *composite index* or *concatenated index*. The general syntax is

```
CREATE INDEX indexname
    ON tablename (columnname1 [, columnname2], . . .);
```

The TABLESPACE and STORAGE clauses can be used with the CREATE INDEX statement. Indexes are stored in a different tablespace and, preferably, on a different physical device from the table data to optimize the index's performance. For example, Figure 9-14 creates an index based on student's last and first names to

```
SQL> CREATE INDEX stu_idx
  2  ON student(Last,  First);

Index created.

SQL> SELECT index_name, table_name FROM user_indexes
  2  WHERE table_name = 'STUDENT';

INDEX_NAME                     TABLE_NAME
------------------------------ ------------------------------
STUDENT_STUDENTID_PK           STUDENT
STU_IDX                        STUDENT

SQL>
```

Figure 9-14 Index.

speed searching of student information when the search involves the last name and first name as search key. The information from the USER_INDEXES table shows two indexes in the STUDENT table, because one index was created automatically for the primary key constraint.

You would create an index based on a column if a column is used often in querying or joining, has a big domain of values, or contains many Null values. When an index is created, it does not store Null values, so the searching would eliminate those rows. Do not create an index for a very small table, a column not used often in queries, or a table that often gets updated. Every insertion and deletion in [illegible]ble

updates the index, which is added overhead on the system.

Oracle also supports *bitmapped* indexes, which are used in data warehousing but are not suitable for tables with a large number of updates. Each data manipulation statement updates the index file, thus slowing down the data manipulation operation.

Rebuilding an Index

When a table goes through many changes (insertions, deletions, and updates), it is advisable to rebuild indexes based on that table. You can drop an index and recreate it, but it is faster to just rebuild an existing index. Rebuilding compacts index data and improves performance. For example,

```
ALTER INDEX stu_idx REBUILD;
```