

Single Table Queries

Week 2 Chapter 4 Video 1
SELECT – FROM statements
Dr. Anne Powell

Agenda

- ❖ 1. Basic SELECT commands
- ❖ 2. Column Formatting
- ❖ 3. Common Errors
- ❖ 4. DISTINCT – WHERE commands
- ❖ 5. ORDER BY command
- ❖ 6. In-class practice

Query

- ❖ Mechanism for extracting information from the database
- ❖ The foundation of all queries is a base SELECT statement
- ❖ A properly written SELECT statement will always produce a result in the form of one or more rows of output
- ❖ The SELECT statement chooses (selects) rows from one or more tables according to specific criteria.

Syntax Conventions

- ❖ Each select statement must follow precise syntactical and structural rules.
- ❖ The following is the *minimum* structure and syntax required for an SQL SELECT statement.

```
SELECT { * | select_list }

FROM table_name;
```

SQL Example 1 ALL Columns

```
SELECT *
FROM Employee;

EMPLOYEEID  SSN          LASTNAME        FIRSTNAME
67555        981789642  Simmons         Lester
33355        890536222  Boudreaux       Beverly
33344        890563287  Adams           Adam
more rows and columns will be displayed...
```

- This query selects rows from the *employee* table.
- The asterisk (*) tells the DBMS to select (display) **all columns** contained in the table “employee”.
- Some rows and columns have been deleted from the display above for readability.

SQL Example 1 ALL Columns

The following SELECT statement produces an identical output.

```
SELECT EmployeeID, SSN, Lastname, FirstName,  
MiddleName, DepartmentNumber, Office, DateHired, Title,  
WorkPhone, PhoneExtension, LicensureNumber, Salary,  
WageRate, ParkingSpace, Gender, SupervisorID  
  
FROM Employee;
```

Selecting Specific Columns

- ❖ Type the exact, complete column name.
- ❖ Separate each column name with a comma (,).
- ❖ Specify the name of the table after the FROM clause.
- ❖ Terminate the query with a semi-colon (;).

```
/* SQL Example */  
SELECT DepartmentNumber, DepartmentName  
FROM Department;
```

Indenting SQL Code

- ❖ A query will be processed regardless of whether you type the entire query on one line or several.
- ❖ There are no rules about how many words can be put on a line or where to break a line.
- ❖ Indenting is not required, but enhances readability.
- ❖ Generally programmers indent code that extends a statement or statement clause 4 spaces.

Indenting SQL Code

- ❖ The following keywords are your signal to start a new line.
 - ❖ SELECT
 - ❖ FROM
 - ❖ WHERE
 - ❖ GROUP BY
 - ❖ HAVING
 - ❖ ORDER BY

Aggregate Functions

Week 3 Chapter 6 Video 1
COUNT
Dr. Anne Powell

Agenda

1. Aggregate Function Rules – COUNT
2. Aggregate Function Rules – the rest
3. GROUP BY command
4. HAVING command
5. Examples

Learning Objectives

- ❖ Write queries with aggregate functions: COUNT, AVG, SUM, MAX, and MIN.
- ❖ Use the GROUP BY clause to answer complex managerial questions.
- ❖ Nest aggregate functions.
- ❖ Use the GROUP BY clause with NULL values.
- ❖ Use the GROUP BY clause with the WHERE and ORDER BY clauses.
- ❖ Use the HAVING clause to filter out rows from a result table.

AGGREGATE ROW FUNCTIONS

- Aggregate Row functions give the user the ability to answer business questions such as:
 - What is the average salary of an employee in the company?
 - What were the total salaries for a particular year?
 - What are the maximum and minimum salaries in the Computer's Department?

AGGREGATE ROW FUNCTION LIST

- List of common aggregate functions

Function Syntax	Function Use
SUM ([ALL DISTINCT] expression)	The total of the (distinct) values in a numeric column/expression.
AVG([ALL DISTINCT] expression)	The average of the (distinct) values in a numeric column/expression.
COUNT([ALL DISTINCT] expression)	The number of (distinct) non-NULL values in a column/expression.
COUNT(*)	The number of (distinct) non-NULL values in a column/expression.
MAX(expression)	The highest value in a column/expression.
MIN(expression)	The lowest value in a column/expression.

Aggregate Function Rules

- ❖ Can be used in the SELECT clause
- ❖ Can be used in the HAVING clause
- ❖ Cannot be used in the WHERE clause

Example Aggregate Function Query – Error in Use of a WHERE Clause

- ❖ The query on the next slide erroneously uses an aggregate function in a WHERE clause.
- ❖ A WHERE clause includes or excludes rows from a result table based on user-defined criteria.
- ❖ Aggregate functions act upon rows that satisfy WHERE clause criteria – since the WHERE clause executes *before* the aggregate function takes effect, you cannot include an aggregate function in a WHERE clause.

Example Aggregate Function Query – Error in Use of a WHERE Clause

```
/* SQL Example 6.2: Who are the  
employees who make the lowest  
salary in the entire company? */
```

```
SELECT *
```

```
FROM Employee
```

```
WHERE Salary = Min(Salary);
```

*ERROR at line 3: ORA-00934: group
function is not allowed here.*

COUNT()

- ❖ If a manager needs know how many employees work in the organization, COUNT(*) can be used to produce this information.
- ❖ The COUNT(*) function counts all rows in a table, will include NULL values.
- ❖ The wild card asterisk (*) is used as the parameter in the function.
- ❖ COUNT(column name) does almost the same thing. The difference is that you may define a specific column to be counted. Excludes NULL values.

```
/* SQL Example 6.1b */  
SELECT COUNT(*) "Number of Employees"  
FROM Employee;
```

Number of Employees

COUNT()

- ❖ The result table for the COUNT(*) function is a single *scalar* value.
- ❖ Normally the result table has a column heading that corresponds to the name of the aggregate function specified in the SELECT clause.
- ❖ The output column can be assigned a more meaningful column name as is shown in the query .

COUNT()

- ❖ COUNT(*) is used to count all the rows in a table.
- ❖ COUNT(column name) does almost the same thing. The difference is that you may define a specific column to be counted.
- ❖ When column name is specified in the COUNT function, rows containing a NULL value in the specified column are omitted.
- ❖ A NULL value stands for “unknown” or “unknowable” and must not be confused with a blank or zero.

Count() – Null Values

- ❖ The *SupervisorID* column has one row with a NULL value.

```
/* SQL Example 6.10 */
```

```
SELECT COUNT(SupervisorID) "Number Supervised Employees"  
FROM Employee;
```

Number Supervised Employees

COUNT ()

- In contrast the count(*) will count each row regardless of NULL values.

```
/* SQL Example 6.11 */  
SELECT COUNT(*) "Number of Employees"  
FROM Employee;  
  
Number of Employees  
-----
```

MULTI-TABLE QUERIES

Week 4 Chapter 7 Video 1
JOINS
Dr. Anne Powell

Agenda

- ❖ Foreign Keys
- ❖ Aliases
- ❖ JOINS – using FROM and using WHERE

JOINS

- ❖ We will begin our study of JOIN operations by focusing on the relationship between the *employee* and *department* tables represented by the common *DepartmentNumber* values.
- ❖ Key to joining: Columns in different tables have shared value domains.
- ❖ Key to referential integrity: Values in one table's columns are used to validate values in another table's columns.

JOINS

- ❖ A JOIN must be used when data in the final result comes from multiple tables.
- ❖ A JOIN selects data from a table based on a search condition involving one or more columns from a different table.
- ❖ What do you need?
 - ❖ Two columns that have the same data type (typically the foreign key and primary key)

JOINS

- ❖ This is a single-table query. The Dept number column values may lack meaning to managers.

```
/* SQL Example 7.1 */  
SELECT LastName "Last Name", FirstName "First Name",  
       DepartmentNumber "Dept"  
FROM Employee  
ORDER BY LastName, FirstName;
```

Last Name	First Name	Dept
Adams	Adam	8
Barlow	William	3
Becker	Robert	3
<i>more rows will be displayed . . .</i>		

JOINS

- ❖ A large organization can have dozens or even hundreds of departments. Thus, the numbers displayed in the department column shown above may not be very meaningful.
- ❖ Suppose you want the department names instead of the department numbers to be listed – the department names are stored in the *department* table.
- ❖ Hence we need to join the *employee* and the *department* tables to produce the required results
- ❖ A JOIN can be coded through either the **FROM** or **WHERE** clause.

JOINS

- ❖ What is different?
 - ❖ Since two fields you are joining are likely named the same thing (i.e., your FK is named the same as your PK), you must prefix your fieldname with the relation name (e.g. Employee.DepartmentNumber and Department.DepartmentNumber)
- ❖ SHORTCUT: Aliases
 - ❖ In the From statement, you can define an alias for a table
 - ❖ FROM Employee E, Department D
 - ❖ So that when you join the PK and FK, you can refer to them as E.departmentNumber and D.departmentNumber

Basic JOIN

- ❖ JOINS can be specified in the WHERE clause
 - ❖ WHERE E.departmentNumber = D.departmentNumber
- ❖ Joins can be specified in the FROM clause
 - ❖ FROM Employee E JOIN Department D
ON (E.departmentNumber = D.departmentNumber)

NOTE: Doesn't matter which table/alias you put first, but typically write left to right based on your drawing.

Query using a WHERE Clause JOIN

- ❖ Same query as previous slide with added attributes.

```
/* SQL Example 7.3 Same query with alias names */

SELECT LastName "Last Name", FirstName "First Name",
       DepartmentName "Department Name"

FROM Employee e, Department d

WHERE e.DepartmentNumber = d.DepartmentNumber

ORDER BY LastName, FirstName;
```

Query using a FROM Clause JOIN

```
/* SQL Example 7.4 - Query with alias names */
SELECT LastName "Last Name", FirstName "First Name",
       DepartmentName "Department Name"
FROM Employee e JOIN Department d
    ON (e.DepartmentNumber = d.DepartmentNumber)
ORDER BY LastName, FirstName;
```

Last Name	First Name	Department Name
Adams	Adam	Admin/Labs
Barlow	William	Emergency-Surgical
Becker	Robert	Emergency-Surgical
<i>more rows will be displayed . . .</i>		

How JOINS Are Processed

- ❖ This shows two tables simply named *Table_1* and *Table_2*.
- ❖ Each table has a single column named *Col_1*. Each table also has three rows with simple alphabetic values stored in the *Col_1* column.

Table_1

COL_1
a
b
c

Table_2

COL_1
a
b
c

How JOINS Are Processed

- ❖ Joining produces a Cartesian product—all possible row combinations for the two tables.

```
SELECT *
FROM table_1, table_2;
COL_1      COL_1
-----  -----
      a          a
      b          a
      c          a
      a          b
      b          b
      c          b
      a          c
      b          c
      c          c
```

How JOINS Are Processed

- ❖ The first row of *table_1* table was joined with every row in *table_2*.
- ❖ A Cartesian product may not be useful and could be misleading – the JOIN shown here links rows that are related.
- ❖ The JOIN filters out rows that are not related.

```
/* SQL Example 6.6 */  
SELECT *  
FROM Table_1 t1 JOIN Table_2 t2  
    ON t1.col_1 = t2.col_1;  
COL_1 COL_1  
----- -----  
a      a  
b      b  
c      c
```

JOIN Operation Rules

- ❖ Starts with a SELECT clause, can use (*) for all fields
- ❖ When an (*) is used, the column order of the result table is based on the order in which tables are listed in the FROM clause.
- ❖ Use the FROM clause to indicate the tables that will be used
 - ❖ Order is irrelevant
 - ❖ When column names are ambiguous you must use aliases

FROM JOIN with a WHERE

```
/* Join conditions on FROM using WHERE */

SELECT LastName "Last Name", FirstName "First Name",
       DepartmentName "Department Name"

FROM Employee e JOIN Department d

ON (e.DepartmentNumber = d.DepartmentNumber)

WHERE e.DepartmentNumber = 8

ORDER BY LastName, FirstName;
```

Last Name	First Name	Department Name
Adams	Adam	Admin/Labs
Boudreaux	Beverly	Admin/Labs
Clinton	William	Admin/Labs
Simmons	Lester	Admin/Labs
Thornton	Billy	Admin/Labs

Example: Same Query with WHERE Clause JOIN

```
/* SQL Example 7.8 */

/* Join conditions in the WHERE clause */

SELECT LastName "Last Name", FirstName "First Name",
       DepartmentName "Department Name"
FROM Employee e, Department d
WHERE e.DepartmentNumber = d.DepartmentNumber AND
      e.DepartmentNumber = 8
ORDER BY LastName, FirstName;
```

Extra Info: Column Names – Rules to Remember

- ❖ The alias name letters are completely arbitrary – pick alias names that mean something to you as the programmer.
- ❖ The important points to learn are:
 - ❖ The alias must follow a table name.
 - ❖ Use a space to separate a table name and its alias.
 - ❖ The alias must be unique within the SELECT statement.
- ❖ If the column names are not identical you are not required to qualify them, although you still might want to for documentation purposes

Extra Info: Additional WHERE Clause Options

- ❖ All selection criteria such as the IN operator can be used to add power to queries can be used in JOIN queries.
- ❖ This retrieves employees based on department numbers in either department 3 (Emergency-Surgical) or 7 (Pharmacy Department).

```
/* SQL Example 7.13 - Join condition in the WHERE clause */
SELECT LastName "Last Name", FirstName "First Name",
       DepartmentName "Department Name"
FROM Employee e, Department d
WHERE e.DepartmentNumber = d.DepartmentNumber AND
      d.DepartmentNumber IN (3, 7)
ORDER BY LastName;
```

Last Name	First Name	Department Name
Barlow	William	Emergency-Surgical
Becker	Robert	Emergency-Surgical
Boudreaux	Betty	Pharmacy Department

More rows will display . . .

Subqueries

CMIS 563

Week 5 Chapter 8 Video 1
Subquery Rules
Dr. Anne Powell

Agenda

- ❖ 1. Explanation of subqueries / Subquery Rules
- ❖ 2. Using IN, ORDER BY, and comparison operators
- ❖ 3. Nest subqueries at multiple levels / Using ANY and ALL keywords
- ❖ 4. Correlated subqueries / EXISTS operator
- ❖ 5. In-class examples

SUBQUERY

- ❖ A *subquery* is a query within a query.
- ❖ Subqueries enable you to write queries that select data rows for criteria that are actually developed while the query is executing at *run time*.

Subqueries

- ❖ Oracle allows a maximum nesting of 255 subquery levels in a WHERE clause. We will NOT do 255 subquery levels! Three at most!
- ❖ The practice of nesting one SELECT statement inside another is where Structured came from in SQL.

Example 8.1

- ❖ Here the WHERE clause criteria are known at design time – management wants high salary employees from departments 3 and 6.

```
/* SQL Example 8.1 */  
SELECT LastName "Last Name", FirstName "First Name",  
       DepartmentNumber "Dept", Salary "Salary"  
FROM Employee  
WHERE Salary >= 20000 AND DepartmentNumber IN (3, 6);
```

Last Name	First Name	Dept	Salary
Becker	Robert	3	\$23,545
Jones	Quincey	3	\$30,550
Barlow	William	3	\$27,500
Smith	Susan	3	\$32,500
Becker	Roberta	6	\$23,000

more rows are displayed . . .

Example: Criteria values unknown

- ❖ BUT, what if you don't know what the criteria values are? This is where the subquery becomes valuable.
- ❖ Report needed: List the names of all employees that earn a salary equal to the minimum salary amount paid within the hospital.
- ❖ You do not know the minimum salary – which can keep changing

```
/* SQL Example Wrong */  
SELECT LastName "Last Name", FirstName  
    "First Name", Salary "Salary"  
FROM Employee  
WHERE Salary = MIN(Salary)
```

- ❖ WHERE Salary = MIN(Salary)
 - *
- ERROR at line 4:
ORA-00934: group function is not allowed here

Example 8.2

- ❖ Report need: List the names of all employees that earn a salary equal to the minimum salary amount paid within the hospital.
- ❖ We can find this information by using a 2-step approach. First we will determine the minimum salary:

Step 1:

```
/* SQL Example 8.2 */  
  
COLUMN "Min Salary" FORMAT $999,999;  
  
SELECT MIN(Salary) "Min Salary"  
  
FROM Employee;
```

Min Salary

\$2,200

Step 2

- ❖ Now you could substitute the minimum salary into another query.

```
/* SQL Example 2-step process */  
SELECT LastName "Last Name", FirstName  
    "First Name", Salary "Salary"  
FROM Employee  
WHERE Salary = 2200;
```

Last Name	First Name	Salary
Simmons	Leslie	\$2,200
Young	Yvonne	\$2,200

Example 8.3

- ❖ Better approach – use a subquery.

```
/* SQL Example 8.3 */  
SELECT LastName "Last Name", FirstName  
    "First Name", Salary "Salary"  
FROM Employee  
WHERE Salary =  
    (SELECT MIN(Salary)  
     FROM Employee);
```

Last Name	First Name	Salary
Simmons	Leslie	\$2,200
Young	Yvonne	\$2,200

Rules

- ❖ A subquery is always enclosed in parentheses.
- ❖ The SELECT clause of a subquery must contain only one expression, only one aggregate function, or only one column name.
- ❖ The value(s) returned by a subquery must be *join compatible* with the WHERE clause of the outer query.
- ❖ The ORDER BY clause cannot be used in writing the subquery part of a query – it **can** be used for the outer query.
- ❖ Subqueries can be nested inside both the WHERE and HAVING clauses of an outer SELECT, or inside another subquery.

Join Compatible Data Types

Rules

- ❖ Values returned by a subquery, for example, EmployeeID values must have a shared domain of values with the outer query condition.
- ❖ The data type of the returned column value(s) must be *join compatible*.
- ❖ Join compatible data types are data types that the Oracle Server will convert automatically when matching data in criteria conditions. But, realize, at least in this class, examples will show the returned column attribute name is the same as the attribute name in the outer query.

Rules Cont'd

- ❖ Oracle does not make comparisons based on column names.
- ❖ Columns from two tables that are being compared may have different names as long as they have a shared domain and the same data type or convertible data types.

Column Name	Data Type
StudentNumber	CHAR
StudentID	CHAR

Additional Subquery Rules

- ❖ The DISTINCT keyword cannot be used in subqueries that include a GROUP BY clause.
- ❖ Subqueries cannot manipulate their results internally. This means that a subquery cannot include the ORDER BY clause, the COMPUTE clause, or the INTO keyword.
- ❖ Columns of a result table can only include columns from a table named in the FROM clause of the outer query—if a table name appears only in a subquery, then the result table cannot contain columns from that table.

Additional Functions

CMIS 563

Week 6 Chapter 10 Video 1
CHAR functions
Dr. Anne Powell

Agenda

- ❖ 1. CHAR functions
- ❖ 2. Mathematical functions
- ❖ 3. Conversion / DATE functions / DECODE

Learning Objectives

- ❖ Use character functions to manipulate CHAR type data.
- ❖ Use mathematical functions to manipulate NUMBER type data.
- ❖ Use conversion functions to convert data from one data type to another data type.
- ❖ Use date functions to manipulate DATE type data.
- ❖ Use the DECODE function to complete value substitutions.

General notation

- Functions are formally defined by using the general notation shown below.
 - ❖ FUNCTION (argument₁, [option])
- The function name is given in capital letters.
- The word *argument₁* is a placeholder that may be filled by either a string of characters enclosed in single-quote marks, a column name, or numeric value.
- As was the case with aggregate functions, each function has a single set of parentheses, and all values and options are enclosed by these parentheses.
- The optional clauses will vary among the different functions.

CHARACTER (String) Functions

FUNCTION	USE/DEFINITION
CONCAT	Concatenates two substrings to return a single string. This function is equivalent to the concatenation operator ().
INITCAP	Capitalizes the first letter of a string of characters.
INSTR	Searches a character string for a character string subset and returns the start position and/or occurrence of the substring.
LENGTH	Returns a numeric value equivalent to the number of characters in a string of characters.
LOWER	Returns a character value that is all lower case.
LTRIM	Trims specified characters from the left end of a string.
RTRIM	Trims specified characters from the right end of a string.
TRIM	Trims characters from both ends of a string.
SUBSTR	Returns a string of specified <i>length</i> from a larger character string beginning at a specified character <i>position</i> .
UPPER	Returns a character value that is all upper case.

UPPER, LOWER, and INITCAP Functions

- The UPPER, LOWER, and INITCAP functions alter the appearance of information in a result table.
- The UPPER function converts data stored in a character column to upper case letters.
- The LOWER function converts data stored in a character column to lower case letters.
- The INITCAP function capitalizes the first letter of a string of characters.
- The general form of these functions is:
 - ❖ `LOWER(CharacterString)`
 - ❖ `UPPER(CharacterString)`
 - ❖ `INITCAP(CharacterString)`

Example 10.1 – LOWER, UPPER, INITCAP

```
/* SQL Example 10.1 */
SELECT LOWER(Gender) "Gender", UPPER(LastName)
"Last Name", Office "Office-Caps",
INITCAP(LOWER(Office)) "Office-InitCap"
FROM Employee;
```

Gender	Last Name	Office-Caps	Office-InitCap
m	SIMMONS	SW4801	Sw4801
f	BOUDREAUX	NW0105	Nw0105
m	ADAMS	NW0105	Nw0105

more rows will be displayed...

LENGTH Function

- ❖ The general form of the LENGTH function is:
LENGTH (CharacterString)
- ❖ The function returns a numeric value equivalent to the number of characters comprised by the specified *CharacterString*.
- ❖ Usually used in conjunction with other functions for tasks such as determining how much space needs to be allocated for a column of output on a report.

Example 10.2 – LENGTH Function

```
/* SQL Example 10.2 */  
COLUMN "City" FORMAT A15;  
COLUMN "Length" FORMAT 999999;  
SELECT DISTINCT City "City", LENGTH(City) "Length"  
FROM Patient;
```

City	Length
Alton	5
Collinsville	12
Edwardsville	12
O Fallon	8

SUBSTR Function and Concatenation

- ❖ SUBSTR function can extract a substring from a string of characters.
- ❖ The general format of the function is:

```
SUBSTR(CharacterString, StartPosition  
       [, NumberOfCharacters])
```

- ❖ *CharacterString* parameter – the string value from which you wish to extract characters.
- ❖ *StartPosition* parameter – specifies the position within the string with which to begin the extraction.
- ❖ *NumberOfCharacters* – optional parameter that specifies how many characters to extract. When this parameter is not specified, the extraction automatically continues to the last character in the string.

Example 10.3 – SUBSTR Function

```
/* SQL Example 10.3 */  
SELECT LastName "Last Name", FirstName "First Name",  
SUBSTR(SSN, 6) "Last 4 SSN"  
FROM Employee  
WHERE DepartmentNumber = 3;
```

Last Name	First Name	Last 4 SSN
Sumner	Elizabeth	3308
Becker	Robert	9991
Jones	Quincey	9993
Barlow	William	9002
Smith	Susan	5540

SUBSTR Function and Concatenation

- The SUBSTR function can be combined with the *concatenation operator* (||) – two vertical lines.
- This enables you to concatenate substrings in order to achieve special formatted output.
- The SELECT statement shown in the next slide formats the employee social security numbers in the result table.
- The concatenation operator is also used to format each employee name (last and first name) for display as a single column.

Example 10.4 - Concatenation

```
/* SQL Example 10.4 */
SELECT LastName || ', ' || FirstName "Employee Name",
SUBSTR(SSN,1,3) || '-' || SUBSTR(SSN,4,2) || '-' ||
SUBSTR(SSN,6,4) "SSN"
FROM Employee
WHERE DepartmentNumber = 3;
```

Employee Name	SSN
Sumner, Elizabeth	216-22-3308
Becker, Robert	347-88-9991
Jones, Quincey	654-33-9993
Barlow, William	787-24-9002
Smith, Susan	548-86-5540

CONCAT Function

- ❖ The CONCAT function can be used instead of the concatenation operator—they produce exactly the same end result.
- ❖ The general format of the function is:

CONCAT(CharacterString1, CharacterString2)

- ❖ When concatenating more than two strings, you must nest CONCAT functions.
- ❖ SQL Example 10.5 uses two nested CONCAT function to combine the employee last name, a comma and blank space, and the employee first name for display in the result table as a single column.

Example 10.5 – CONCAT Function

```
/* SQL Example 10.5 */  
SELECT CONCAT(CONCAT(Concat(LastName, ', '), FirstName)  
    "Employee Name",  
    CONCAT(CONCAT(CONCAT(SUBSTR(SSN, 1, 3), '-'),  
        SUBSTR(SSN, 4, 2)), '-'), SUBSTR(SSN, 6, 4)) "SSN"  
FROM Employee  
WHERE DepartmentNumber = 3;
```

Employee Name	SSN
Sumner, Elizabeth	216-22-3308
Becker, Robert	347-88-9991
Jones, Quincey	654-33-9993
Barlow, William	787-24-9002
Smith, Susan	548-86-5540

Example 10.5 - Explained

- ❖ The first CONCAT function concatenates the values of the *LastName* column with a comma and space. This creates an initial employee name string value for each employee with output as follows:

Sumner,

Becker,

Jones,

Barlow,

Smith,

Example 10.5 - Explained

- ❖ The second CONCAT function concatenates the initial employee name with the *FirstName* column values. This creates the complete string value displayed under the Employee Name column heading as follows:

Sumner, Elizabeth

Becker, Robert

Jones, Quincey

Barlow, William

Smith, Susan

- ❖ Can you decipher the nested CONCAT functions that combine with SUBSTR functions to format the SSN output?

SQL*Plus Reports

CMIS 563

Week 7 Chapter 9 Video 1
Report Formatting
Dr. Anne Powell

Agenda

- ❖ 1. Report Formatting
- ❖ 2. Control Break Reports
- ❖ 3. Master Detail Reports
- ❖ 4. Interactive Programs

Learning Objectives

- ❖ Create a SQL*Plus program command file and view command file settings.
- ❖ Select data to display in reports.
- ❖ Format all aspects of report layout.
- ❖ Create a control break report including clearing breaks and using the BREAK and COMPUTE commands.
- ❖ Use the SPOOL command to produce report listing files.
- ❖ Create Master-Detail reports including the use of variables in report titles and footers with the COLUMN command NEW_VALUE clause.
- ❖ Use the ACCEPT, PROMPT, and PAUSE commands for interactive program execution with substitution variables.
- ❖ Define user variables for interactive reporting and pass parameter values through the START command.

SQL*Plus Program Command File

- Focus on creating files to store SQL*Plus commands – a command file or SQL program.
- File name extension is *.sql*.
- Commands can specify report headings, report footers, report titles, page numbers, columns, page breaks, and other common report features that managers need.
- When you exit SQL*Plus, all of the information about a report's features are stored in the file so nothing is lost.

Example 9.1

```
/* SQL Example 9.1 */
-- Program: ch9-1.sql
-- Programmer: apowell; today's date
-- Description: List employee project work history
TTITLE 'Project Information Details'
BTITLE SKIP 1 CENTER 'Not for external dissemination.'
REPHEADER 'Project Report #1 - prepared by A. Powell'
SKIP 2
REPFOOTER SKIP 3 '- Last Page of Report -'
SET LINESIZE 55
SET PAGESIZE 24
SET NEWPAGE 1
```

Example 9.1 Cont'd

- ❖ COLUMN "Employee ID" FORMAT A11
- ❖ COLUMN "Hours Worked" FORMAT 999.99
- ❖
- ❖ SELECT EmployeeID "Employee ID",
ProjectNumber "Project #", HoursWorked "Hours
Worked"
- ❖ FROM ProjectAssignment
- ❖ ORDER BY EmployeeID, ProjectNumber;
- Pages #1 and #2 of the report produced by the program
are given on the next two slides.

Project Information Details

Project Report #1 - prepared by A. Powell

Employee ID	Project #	Hours Worked
-------------	-----------	--------------

01885	3	10.20
23100	4	10.30
23100	7	
23232	1	14.20
23232	2	10.60
23232	8	
33344	4	5.10
33344	5	9.50
33344	8	23.00
33358	5	41.20

< some report lines deleted here >

Mon Aug 9

page 2

Project Information Details

Employee ID	Project #	Hours Worked
-------------	-----------	--------------

-----	-----	-----
-------	-------	-------

67555	5	35.40
67555	7	12.20
67555	8	24.10
88505	4	34.50
88777	3	30.80

- Last Page of Report -

Not for external dissemination.

Remarks

- Optional remarks are typically entered at the beginning of a command file program that identify the filename, programmer name, date of program creation, brief description of program.
- Symbols used to enter remarks:
 - ❖ /* Remark is enclosed here */
 - ❖ -- Remark with two dash lines
- Remarks may include a list of modifications made by programmer name, date and description here.
- Remarks and blank lines are used throughout a program to enhance the understandability and readability of programming code.

Top and Bottom Titles

- Titles and footers on reports enhance the meaning of reports for managerial system users.
- Reports are rarely disseminated to managers without appropriate title and footers.
- SQL*Plus supports the programming of four types of titles and footers:
 1. Top title,
 2. Bottom title,
 3. Report header and
 4. Report footer.

Top and Bottom Titles

- The TTITLE command (short for top title) prints a title on *each page* of a report.
 - When a simple TTITLE command like the one shown below is used, the report will automatically display the report date and page number.
- ❖ TTITLE 'Project Information Details'

Top and Bottom Titles

- The TTITLE command can be entered interactively at the SQL> prompt.
- The first TTITLE command shown below turns the report title off.
- The second TTITLE command changes the report title interactively when followed by a slash (/) command.

```
TTITLE OFF
```

```
TTITLE 'Project and Employee Details'
```

```
/
```

Top and Bottom Titles

- The BTITLE command prints a bottom title with the specified information at the bottom of *each page* of a report.
- For example, your organization may want each page of a report marked as not for external dissemination as is shown in the BTITLE command here.

```
BTITLE SKIP 1 CENTER 'Not for external  
dissemination.'
```

Top and Bottom Titles

- The SKIP clause is optional.
- SKIP 1 inserts one blank line into the report.
- You can specify the number of lines to skip. If the SKIP option is specified prior to the bottom title, as shown here, then one line is skipped prior to printing the bottom title.

```
BTITLE SKIP 1 CENTER 'Not for  
external dissemination.'
```

Top and Bottom Titles

- Top and Bottom Titles can be positioned with the keywords CENTER, RIGHT, and LEFT.
- An example multi-lined TTITLE command is shown below.
- The dash (-) at the end of a line continues the command to the next line.
- The *sql.pno* entry is a predefined SQL variable that can be used to display the current page number for a report.
- When a complex TTITLE command is used, Oracle does not automatically print the date and page number information as was done earlier .

```
TTITLE LEFT DateVar -
        RIGHT 'Page: ' FORMAT 99 sql.pno SKIP 1 -
        CENTER 'Project and Employee Details'
```

Report Headers and Footers

- A report header prints to the top of the *first page* of a report.
- This REPHEADER command uses the SKIP 2 option to insert two blank lines immediately after the report header is printed.
- Notice that the report header prints after the top title line.

```
REPHEADER 'Project Report #1 - prepared by  
A. Powell' SKIP 2
```

Report Headers and Footers

- Report footers print to the bottom of the *last page* of a report.
- Here the SKIP 3 option provides for three skipped blank lines prior to printing the report footer.
- Note that the report footer prints prior to the bottom title line.

```
REPFOOTER SKIP 3 '-- Last Page of Report --'
```

- The OFF option also applies to report headers and footers, and will turn the report header and/or footer off.

```
REPHEADER OFF
```

```
REPFOOTER OFF
```

Setting the Line and Page Size

- SET LINESIZE specifies the size of an output line in characters. Depending on font size, a printer may be able to handle 79 characters.
- This command sets a line size of 55 characters.

```
SET LINESIZE 55
```

- SET PAGESIZE specifies the number of lines to be printed per page.
- A typical setting is 50 to 55 lines of output per page for 10-point or 12-point printer fonts.
- This command sets the page size to 50 lines.

```
SET PAGESIZE 50
```

Setting the Line and Page Size

- SET NEWPAGE specifies the number of blank lines to print before the top title line of a report, that is, the line that displays the report date and page number.
- This is useful for aligning reports produced by various types of printers.
- SET NEWPAGE does not affect the PAGESIZE value.
- This command specifies 6 blank lines at the top of *each page*. If the page size is set to 55, this will leave 49 lines for displaying output.

```
SET NEWPAGE 6
```

Output to the Computer Monitor Screen

- When you are testing a SQL program that will be produce a printed report, it is sometimes useful to specify values for the LINESIZE, PAGESIZE, and NEWPAGE values so that report output will fit on a computer monitor screen.
- Typical values for screen output are shown below.

```
SET LINESIZE 79
```

```
SET PAGESIZE 24
```

```
SET NEWPAGE 0
```

Output to the Computer Monitor Screen

- Use this series of SET PAUSE commands to cause computer monitor screen output to pause between pages so that you can review the report.



```
SET PAUSE 'More . . .'
```

```
SET PAUSE ON
```

```
SET PAUSE OFF
```

SELECT Statement

- ❖ A SELECT statement supplies the information to display in report detail lines.
- ❖ This SELECT statement sorts the output rows by employee identifier (*EmployeeID*), then by project number (*ProjectNumber*).

```
SELECT EmployeeID, ProjectNumber, HoursWorked  
FROM ProjectAssignment  
ORDER BY EmployeeID, ProjectNumber;
```

ERD Basics

Dr. Anne Powell
CMIS563

Let's talk databases

- ❖ Why do we use them? Why not excel?

Last Name	First Name	Email	Adviser Last Name	AdviserEmail
Andrews	Matthew	Matthew Andrews@ourcampus.edu	Baker	Linda Baker@ourcampus.edu
Brisbon	Lisa	Lisa Brisbon@ourcampus.edu	Valdez	Richard Valdez@ourcampus.edu
Fischer	Douglas	Douglas Fischer@ourcampus.edu	Backer	Linda Baker@ourcampus.edu
Hwang	Terry	Terry Hwang@ourcampus.edu	Taing	Susan Tang@ourcampus.edu
Lai	Tzu	Tzu Lai@ourcampus.edu	Valdez	Richard Valdez@ourcampus.edu
Marino	Chip	Chip Marino@ourcampus.edu	Tran	Ken Tran@ourcampus.edu
Thompson	James	James Thompson@ourcampus.edu	Taing	Susan Taing@courcampus.edu

The Entity-Relationship Diagram

- ❖ Picture of the people, places, objects, things, events, or concepts, their characteristics and relationships, for an organization or business.
- ❖ Visual representation
- ❖ Communication tool
- ❖ Independent of technology
- ❖ Understandable representation of organization data

Entities

- ❖ Something of importance to a user that needs to be represented in a database
- ❖ One theme or topic
- ❖ Restricted to things that can be represented by a single table

The Components of an E-R Diagram...

❖ Entities

- ❖ Something of importance to a user that needs to collect data about
- ❖ One theme or topic (noun)

Employee

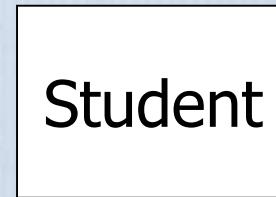
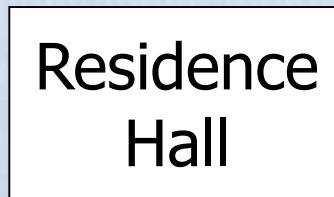
❖ Relationship

- ❖ Association between entities (verbs)
- ❖ Directional
- ❖ Employee Works in Department

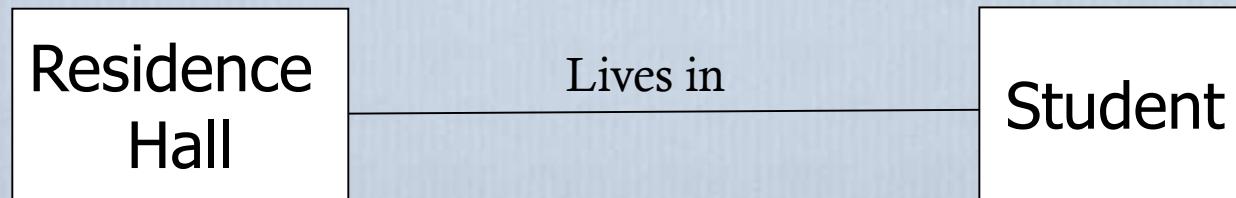
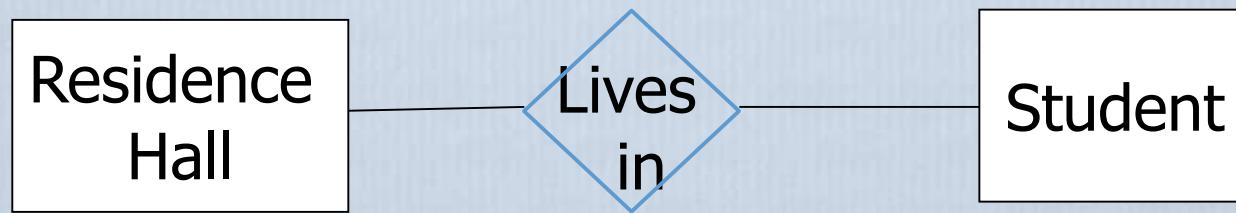
Works

Notation

- ❖ Entities



- ❖ Relationships



Relation

- ❖ A **relation** is a two-dimensional table that has specific characteristics.
- ❖ The table dimensions, like a matrix, consist of rows and columns.

Terminology

- ❖ Synonyms
 - ❖ Each column contains words that are synonyms

Relation		Attribute
Entity	Entity Instance	Attribute
Table	Row	Column
File	Record	Field

- ❖ An entity is a table is a file.
- ❖ An entity instance is a row is a record.
- ❖ An attribute is a column is a field.

Characteristics of a Relation/Entity

- ❖ Rows contain data about an entity.
- ❖ Columns contain data about attributes of the entity.
- ❖ Cells of the table hold a single value.
- ❖ All entries in a column are of the same kind.
- ❖ Each column has a unique name.
- ❖ The order of the columns is unimportant.
- ❖ The order of the rows is unimportant.
- ❖ No two rows may be identical.

Sample relation/entity

EmployeeNumber	FirstName	LastName
100	Mary	Abernathy
101	Jerry	Cadley
104	Alex	Copley
107	Megan	Jackson

And one way this might be shown:

EMPLOYEE(EmployeeNumber, FirstName, LastName)

A non-relation example

Cells of the table hold multiple values

EmployeeNumber	Phone	LastName
100	335-6421, 454-9744	Abernathy
101	215-7789	Cadley
104	610-9850	Copley
107	299-9090	Jackson

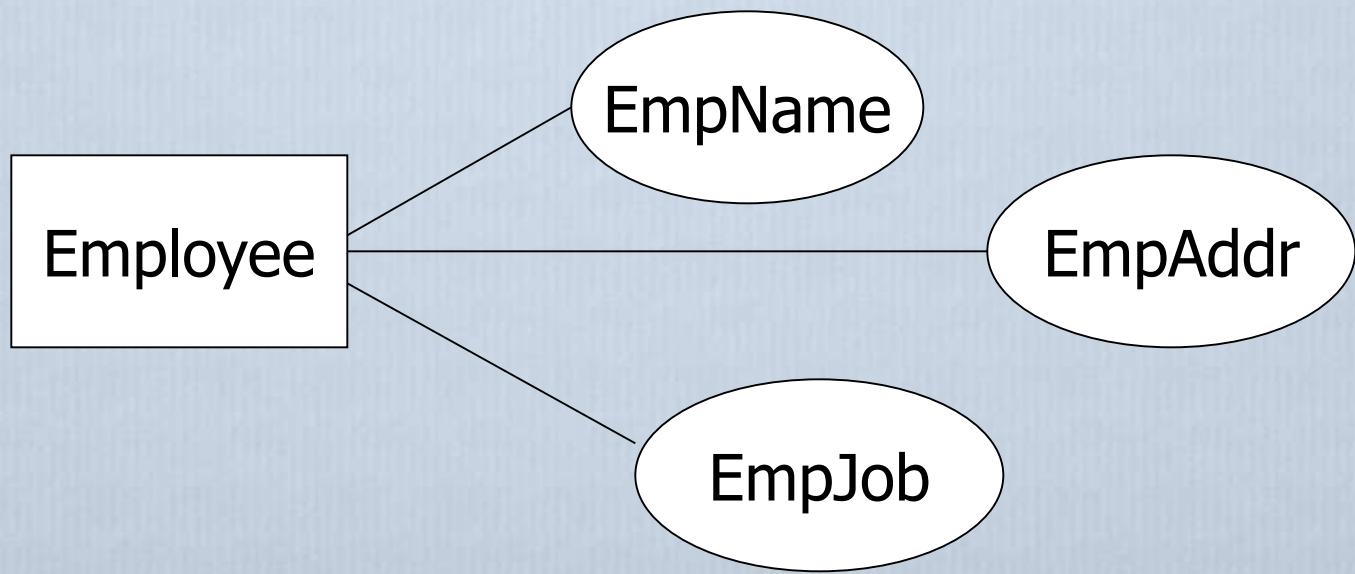
A non-relational table

No two rows may be identical

EmployeeNumber	Phone	LastName
100	335-6421	Abernathy
101	215-7789	Cadley
104	610-9850	Copley
100	335-6421	Abernathy
107	299-9090	Jackson

And there's more

- ❖ Attributes
 - ❖ Properties or characteristics of entities
 - ❖ Actual data items we collect

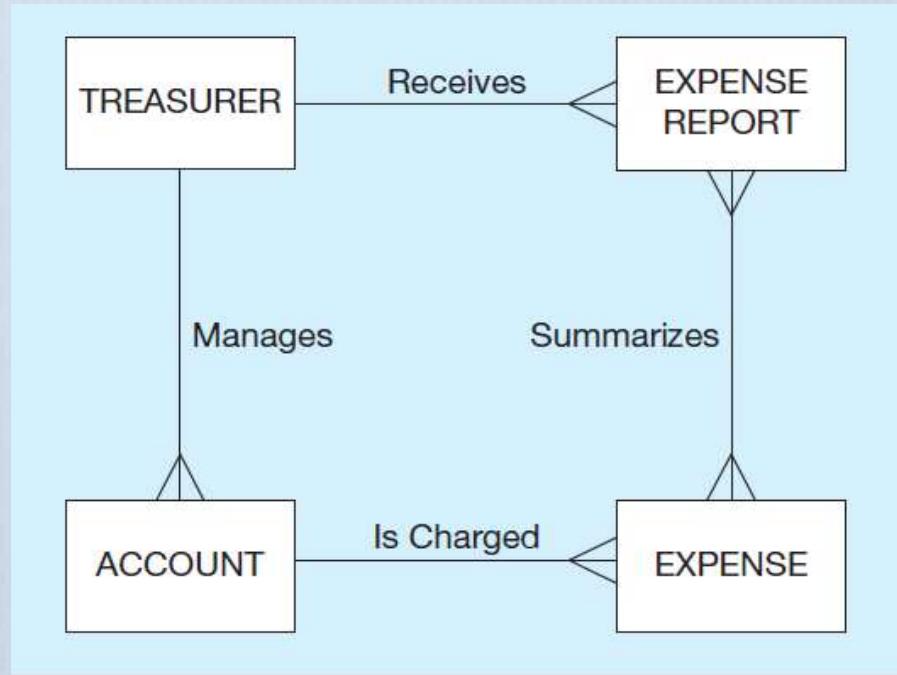


An entity . . .

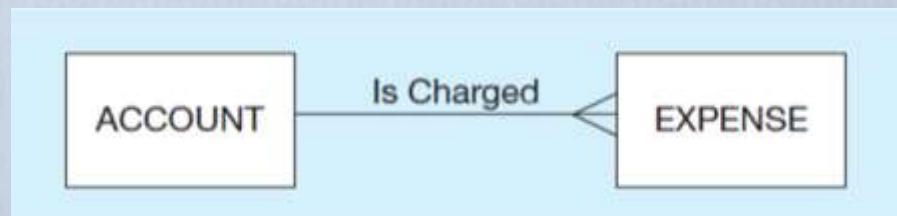
- ❖ **Should Be:**
 - ❖ An object that will have many instances in the database
 - ❖ An object that will be composed of multiple attributes
 - ❖ An object that we are trying to model
- ❖ **Should Not Be:**
 - ❖ A user of the database system
 - ❖ An output of the database system (e.g., a report)

Example of Inappropriate Entities

(a) System user (Treasurer) and output (Expense Report) shown as entities



b) E-R diagram with only the necessary entities



Multivalued Attributes ...

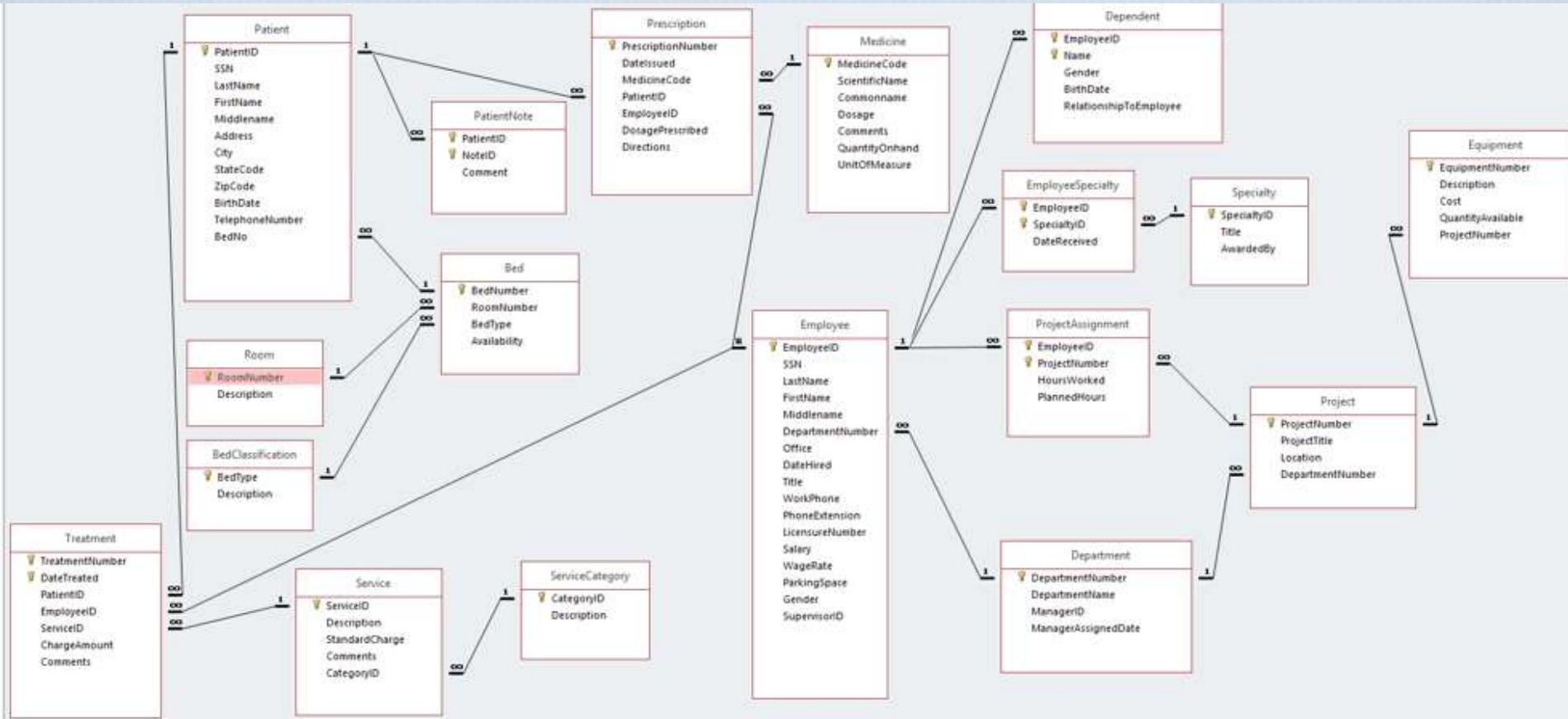
... are bad!!!

- ❖ Attributes that can have multiple values for an entity instance (i.e., a “repeating group”)
- ❖ Should be re-modeled as a separate entity that has a relationship to the entity type from which it was removed (i.e., a “repeating group” is implemented as a separate table)

Multi-value attributes

EMPLOYEE	
PK	EmpID
	LastName
	FirstName
	Children
	Birthdates

EmpID	LastName	FirstName	Children	Birthdates
6141	Powell	Anne	Dan, Mike, John	11/10/2008, 12/07/2012, 04/19/2014
6142	Hannah	Katie	Ike, Isabelle	01/28/2009, 01/31/2011
6143	Thrall	Daniel	Will, Margaret, Edie	08/06/2009, 02/13/2013, 10/21/2017



SQL for Analytics

CMIS 563

Dr. Anne Powell
Module 1 Video 1
SQL Basics

Agenda – Module 1

- ❖ *Video 1: Introduction to SQL*
- ❖ Video 2: CREATE a new table in SQL;
Get signed into SQL via Putty and WinSCP
- ❖ Video 3: Add constraints to fields in tables
- ❖ Video 4: Additional basics in DDL
- ❖ Video 5: Relating (joining) tables in SQL using Foreign Keys
- ❖ Video 6: Populating tables with data

SQL

- ❖ SQL (pronounced ‘Sequel’ or simply S-Q-L)
 - ❖ computer programming language developed especially for querying relational databases using a non-procedural approach.
- ❖ *Nonprocedural*
 - ❖ Extract information by simply telling the system what information is needed without telling it how to perform the data retrieval. The RDBMS parses (converts) the SQL commands and completes the task.

SQL

- ❖ Extracting information from the database by using SQL is termed *querying* the database.
- ❖ SQL is a language that is fairly simple to learn, in terms of writing queries but it has considerable complexity because it is a very powerful language.

SQL

- ❖ SQL
 - ❖ A comprehensive database language
 - ❖ Standards
 - ❖ American National Standards Institute (ANSI)
 - ❖ International Standards Organization (ISO)
- ❖ Most widely used database language
- ❖ Commands for both manipulation and definition of databases.

SQL

- ❖ SQL is used by Oracle for all interaction with the database.
- ❖ Two types of data are stored within a database
 - ❖ System data: Data the database needs to manage user data and to manage itself. This is also termed *metadata, or the data about data*.
 - ❖ User data: Data that must be stored by an organization.

Command Type	Actions	Acts On
Data Definition Language (DDL)	Create and define objects in the database	System data
Data Manipulation Language (DML)	Manipulate data in the database	User data

Relational Operations

- ❖ DDL include (among others):
 - ❖ CREATE TABLE
 - ❖ CREATE INDEX
 - ❖ ALTER TABLE
 - ❖ CREATE USER
- ❖ DML operations include:
 - ❖ SELECT – primary command to write queries to extract database information from tables
 - ❖ INSERT – command to insert new data rows in a table
 - ❖ UPDATE – command to alter data stored in existing data rows
 - ❖ DELETE – command to remove data rows from a table

SQL

- ❖ Free format language: no particular spacing rules that must be followed
- ❖ The Semi-Colon:
 - ❖ Every SQL statement ends with a semi-colon no matter how many lines of code
 - ❖ This tells Oracle to execute the code
- ❖ Keywords
 - ❖ Have a predefined meaning in SQL.
 - ❖ May be entered in all upper or all lower case letters.
 - ❖ Some can be abbreviated (DESC or DESCRIBE)

SQL Naming Conventions

- ❖ Identifiers: names given to database objects such as tables, columns, indexes, and other objects as well as the database itself.
- ❖ Rules:
 - ❖ Between 1 and 30 characters.
 - ❖ The first character must be either alphabetic (a-z, A-Z) or the @ symbol.
 - ❖ After the first character, digits, letters, or the symbols \$,#, or _(underscore) may be used.
 - ❖ No embedded spaces are allowed in identifiers.
 - ❖ SQL keywords cannot be used as an identifier.
 - ❖ For example, you cannot name a table called ORDER.

Questions?



Single Table Queries

Week 2 Chapter 4 Video 2
Formatting your SQL Query
Dr. Anne Powell

Agenda

- ❖ 1. Basic SELECT commands
- ❖ **2. Column Formatting**
- ❖ 3. Common Errors
- ❖ 4. DISTINCT – WHERE commands
- ❖ 5. ORDER BY command
- ❖ 6. In-class practice

Column-Format Command

- ❖ Use this command to format the output display of column data by resizing the width of columns where output will not fit within a column and wraps around, or by specifying a formatting template.

Without Formatting

```
SQL> SELECT SSN, LastName, FirstName, DateHIred, SupervisorID
  2  FROM Employee;

SSN      LASTNAME
----- -----
FIRSTNAME                               DATEHIRED SUPER
----- -----
345670136 Parker
Albers

981789642 Simmons
Lester                                03-MAR-98

890536222 Boudreaux
Beverly                               15-OCT-01 67555

SSN      LASTNAME
----- -----
FIRSTNAME                               DATEHIRED SUPER
----- -----
890563287 Adams
Adam                                  29-JAN-85 33355

834576129 Thornton
Billy                                 14-APR-00 33355

457890233 Clinton
William                               05-JAN-01 33355
```

Column-Format Example #1 – Preventing Wrap Around Lines

```
/* SQL Example */
COLUMN SSN          FORMAT A9;
COLUMN LastName     FORMAT A11;
COLUMN FirstName    FORMAT A11;
COLUMN SupervisorID FORMAT A12;
SELECT SSN, LastName, FirstName, DateHired,
       SupervisorID
FROM Employee;
```

SSN	LASTNAME	FIRSTNAME	DATEHIRED	SUPERVISORID
981789642	Simmons	Lester	03-MAR-98	
890536222	Boudreaux	Beverly	15-OCT-01	67555
890563287	Adams	Adam	29-JAN-85	33355
<i>more rows will be displayed...</i>				

With Formatting

```
SQL> /* SQL Example */
SQL> COLUMN SSN          FORMAT A9;
SQL> COLUMN LastName     FORMAT A11;
SQL> COLUMN FirstName    FORMAT A11;
SQL> COLUMN SupervisorID FORMAT A12;
SQL> SELECT SSN, LastName, FirstName, DateHired, SupervisorID
  2 FROM Employee;
```

SSN	LASTNAME	FIRSTNAME	DATEHIRED	SUPERVISORID
981789642	Simmons	Lester	03-MAR-98	
890536222	Boudreaux	Beverly	15-OCT-01	67555
890563287	Adams	Adam	29-JAN-85	33355
834576129	Thornton	Billy	14-APR-00	33355
457890233	Clinton	William	05-JAN-01	33355
310223232	Eakin	Maxwell	06-JAN-98	67555
215243964	Bock	Douglas	11-AUG-87	23232
316223244	Webber	Eugene	06-FEB-95	67555
261223803	Bordoloi	Bijoy	23-AUG-99	23244
664865650	Smith	Alyssa	10-JUN-99	23244

- ❖ More rows would follow ...

Column-Format Example #2 – Formatting Numeric Columns

```
/* SQL Example */

COLUMN Salary FORMAT 99999.99;

SELECT Salary
FROM Employee;

      SALARY
-----
22000.00
17520.00
5500.00

more rows will be displayed..
```

Column-Format Example #3 – Formatting with a Currency Symbol

```
/* SQL Example */  
COLUMN Salary FORMAT $99,999.99;  
SELECT Salary  
FROM Employee;  
      SALARY
```

\$22,000.00
\$17,520.00
\$5,500.00

more rows will be displayed...

Aggregate Functions

Week 3 Chapter 6 Video 2
AVG, SUM, MIN, MAX
Dr. Anne Powell

Agenda

1. Aggregate Function Rules – COUNT
2. Aggregate Function Rules –
SUM, AVG, MIN, MAX
3. GROUP BY command
4. HAVING command
5. Examples

AGGREGATE ROW FUNCTIONS

- Aggregate Row functions give the user the ability to answer business questions such as:
 - What is the average salary of an employee in the company?
 - What were the total salaries for a particular year?
 - What are the maximum and minimum salaries in the CMIS Department?

AGGREGATE ROW FUNCTION LIST

- List of common aggregate functions

Function Syntax	Function Use
SUM ([ALL DISTINCT] expression)	The total of the (distinct) values in a numeric column/expression.
AVG([ALL DISTINCT] expression)	The average of the (distinct) values in a numeric column/expression.
COUNT([ALL DISTINCT] expression)	The number of (distinct) non-NULL values in a column/expression.
COUNT(*)	The number of (distinct) non-NULL values in a column/expression.
MAX(expression)	The highest value in a column/expression.
MIN(expression)	The lowest value in a column/expression.

Aggregate Function Rules

- ❖ Can be used in the SELECT clause
- ❖ Can be used in the HAVING clause
- ❖ Cannot be used in the WHERE clause

Using the AVG Function

- AVG function is used to compute the average value for the *Salary* column in the *employee* table.
- For example, the following query returns the average of the employee salaries.

```
/* SQL Example 6.3 */  
COLUMN "Average Employee Salary" FORMAT $999,999;  
SELECT AVG(Salary) "Average Employee Salary"  
FROM Employee;
```

Average Employee Salary

\$15,694

AVG with DISTINCT Clause Example

- What is the average salary offered to employees?
- This question asks you to incorporate the concept of computing the average of the distinct salaries paid by the organization.
- The same query with the DISTINCT keyword in the aggregate function returns a different average.

```
/* SQL Example 6.4 */  
SELECT AVG(DISTINCT Salary) "Average Employee Salary"  
FROM Employee;  
  
Average Employee Salary  
-----  
$16,336
```

Using the SUM Function

- Use the SUM function to compute a total value.
- This SELECT statement returns the total of the *Salary* column from the *employee* table.

```
/* SQL Example 6.5 */  
COLUMN "Total Salary" FORMAT $999,999;  
  
SELECT SUM(Salary) "Total Salary"  
FROM Employee;
```

Total Salary

\$345,265

More SUM Examples

- If management is preparing a budget for various departments, you may be asked to write a query to compute the total salary for different departments.
- This query computes the total Salary for employees assigned to department #8.

```
/* SQL Example 6.6 */  
COLUMN "Total Salary Dept 8" FORMAT $999,999;  
SELECT SUM(Salary) "Total Salary Dept 8"  
FROM Employee  
WHERE DepartmentNumber = 8;
```

Total Salary Dept 8

\$45,020

MIN and MAX Functions

- The MIN function returns the lowest value stored in a data column.
- The MAX function returns the largest value stored in a data column.
- Unlike SUM and AVG, the MIN and MAX functions work with numeric, character, **and** date data columns.

MIN and MAX Functions Example

- A query that uses the MIN function to find the lowest value stored in the *LastName* column of the *employee* table.
- This is analogous to determine which employee's last name comes first in the alphabet.
- Conversely, MAX() will return the employee row where last name comes last (highest) in the alphabet.

```
/* SQL Example 6.8 */  
SELECT MIN(LastName), MAX(LastName)  
FROM Employee;  
  
----- -----  
MIN (LASTNAME)    MAX (LASTNAME)
```

Adams

Zumwalt

MIN and MAX – Numeric Data

- ❖ This query returns the highest and lowest salaries from the *employee* table.
- ❖ Later you will learn how to attach names to the queries because the salary figures by themselves are not very useful.

```
/* SQL Example 6.9 */

COLUMN "Highest Salary" FORMAT $999,999

COLUMN "Lowest Salary" FORMAT $999,999

SELECT MAX(Salary) "Highest Salary", MIN(Salary) "Lowest Salary"
FROM Employee;
```

Highest Salary Lowest Salary

\$32,500 \$2,200

MIN and MAX – Date Data

- ❖ This query returns the earliest value for DateHired (longest employee) and latest value for DateHired (newest employee) from the *employee* table.
- ❖ Notice that the MIN and MAX functions here appear to work backwards—they don't, they just look that way.

```
/* SQL Example 6.9b */  
COLUMN "Oldest Hire" FORMAT A11;  
COLUMN "Newest Hire" FORMAT A11;  
SELECT MAX(DateHired) "Newest Hire",  
       MIN(DateHired) "Oldest Hire"  
FROM Employee;
```

Newest Hire	Oldest Hire
15-OCT-01	14-DEC-79

MULTI-TABLE QUERIES

Week 4 Chapter 7 Video 2
Complex JOINs
Dr. Anne Powell

Agenda

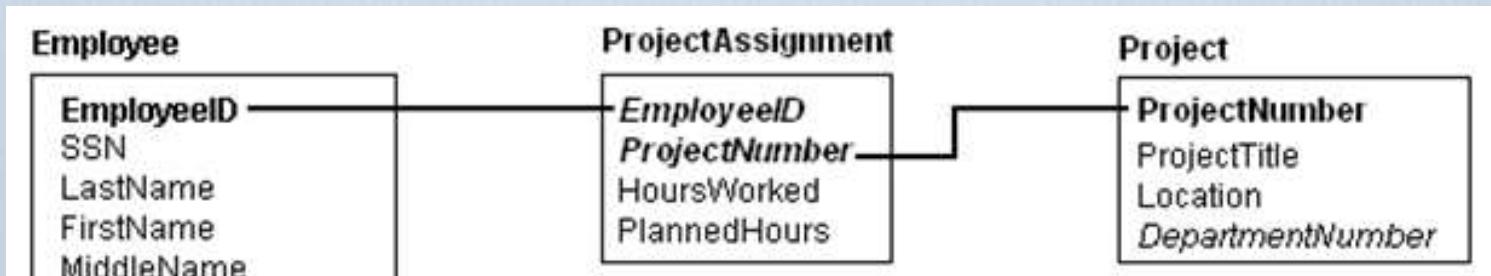
- ❖ Complex Joins (more than 2 tables)
- ❖ In-Class practice

Complex Joins of more than 2 tables

- ❖ While the examples given thus far have joined rows from two tables, you can specify up to 16 tables in a JOIN operation.
- ❖ The more tables that are included in a JOIN operation, the longer the query will take to process, especially when the tables are large with millions of rows per table.

Complex JOIN

- ❖ What about those associate entities that connect to two tables at the same time?



- ▶ FROM Employee e JOIN ProjectAssignment a
ON (e.EmployeeID = a.EmployeeID) JOIN Project p
ON (a.ProjectNumber = p.ProjectNumber)

Or:

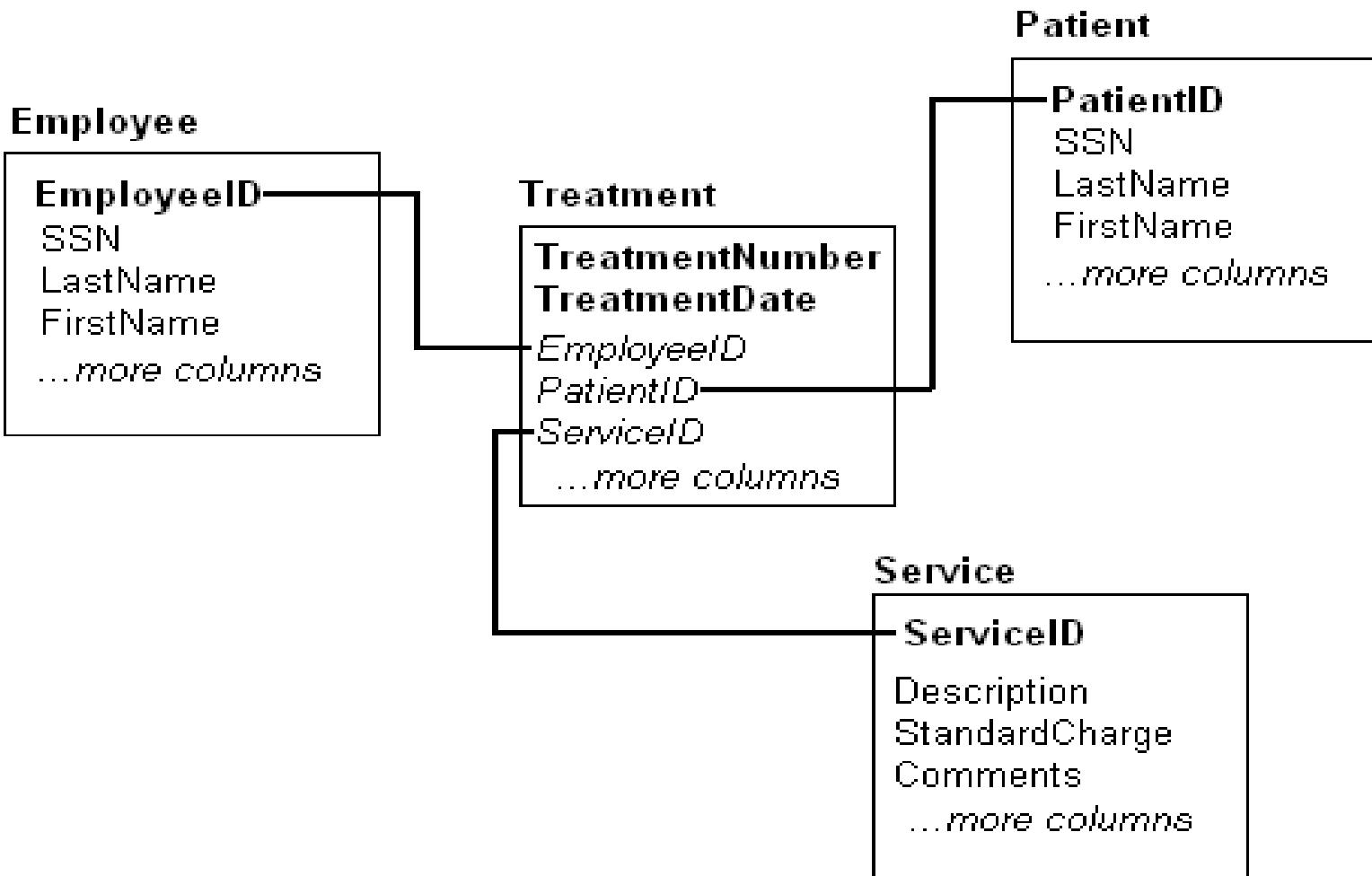
- ▶ FROM Employee e, ProjectAssignment a, Project p
WHERE e.employeeID = a.employeeID AND a.projectNumber =
p.projectNumber

Joining Three Tables

- ❖ The SELECT statement to join the tables depicted in the figure is shown here.

```
/* SQL Example 7.16 - Join conditions in the FROM clause */
COLUMN "Raised Salary" FORMAT $999,999;
SELECT LastName "Last Name", FirstName "First Name",
       1.10*Salary "Raised Salary", p.ProjectTitle "Project"
FROM Employee e JOIN ProjectAssignment a
      ON (e.EmployeeID = a.EmployeeID) JOIN Project p
      ON (a.ProjectNumber = p.ProjectNumber)
WHERE p.ProjectTitle LIKE 'Child Care Center';
Last Name          First Name          Raised Salary Project
-----  -----  -----
Eakin              Maxwell           $16,500 Child Care Center
Adams              Adam               $6,050 Child Care Center
Simmons            Lester             $24,200 Child Care Center
```

Joining Four Tables



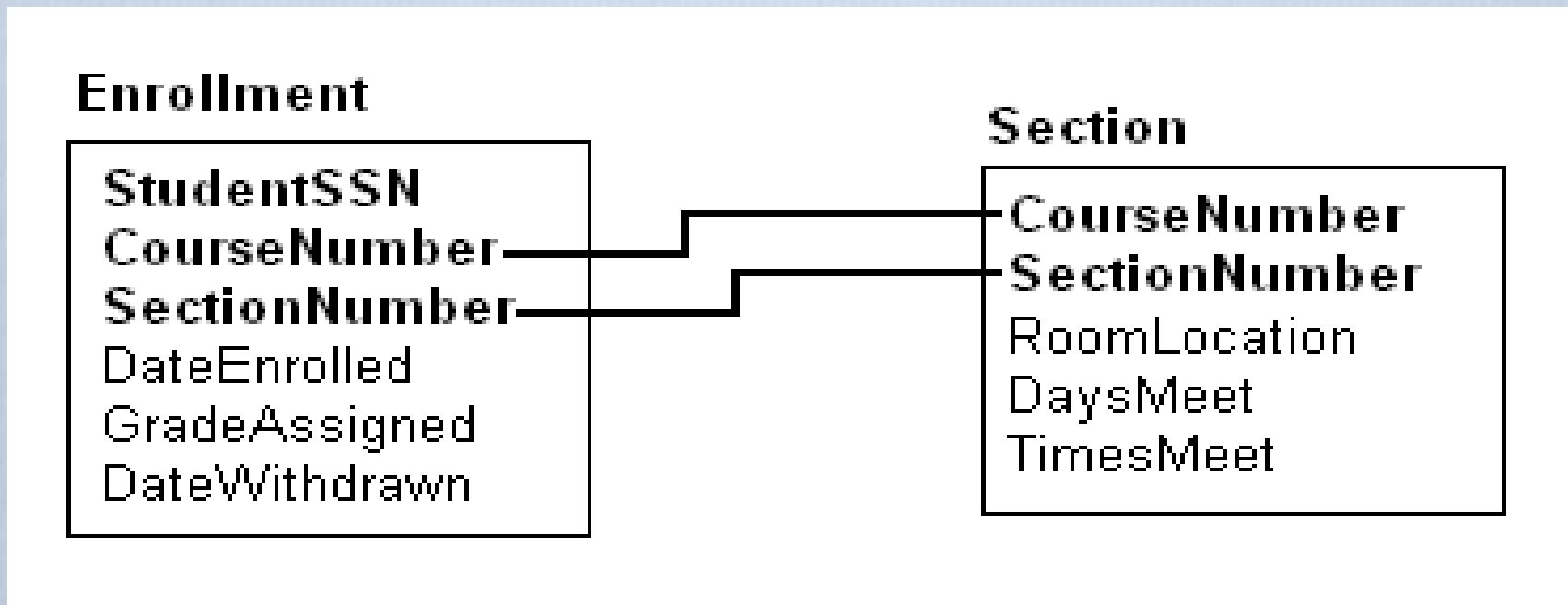
Joining Four Tables using FROM

```
/* SQL Example 7.18 - Join conditions in the FROM clause */
SELECT p.LastName || ', ' || p.FirstName "Patient Name",
       s.Description "Treatment",
       e.LastName "Med Employee"
FROM Patient p JOIN Treatment t ON (p.PatientID = t.PatientID)
                JOIN Employee e ON (e.EmployeeID = t.EmployeeID)
                JOIN Service s ON (s.ServiceID = t.ServiceID)
WHERE e.LastName = 'Quattromani'
ORDER BY p.LastName, p.FirstName;
```

Patient Name	Treatment	Med Employee
Ridgeway, Ricardo	EKG/Interp	Quattromani
Ridgeway, Ricardo	Therapeutic Inj	Quattromani

Joining Tables by Using Two Columns

- ❖ The diagram depicts the relationship at a university where students enroll in course sections.



Joining Tables by Using Two Columns

- ❖ The SELECT statement that accomplishes the JOIN based on two columns is shown below.
- ❖ This situation arises when the related tables have *composite primary key* columns.

```
/* SQL Example 7.19 */  
/* Join conditions in the WHERE clause */  
SELECT s.CourseNumber "Course", e.StudentSSN "Student SSN",  
      s.RoomLocation "Room"  
FROM Enrollment e, Section s  
WHERE e.CourseNumber = s.CourseNumber AND  
      e.SectionNumber = s.SectionNumber;  
  
/* Join conditions in the FROM clause */  
SELECT s.CourseNumber "Course", e.StudentSSN "Student SSN",  
      s.RoomLocation "Room"  
FROM Enrollment e JOIN Section s  
ON (e.CourseNumber = s.CourseNumber) AND  
  (e.SectionNumber = s.SectionNumber);
```

Subqueries

CMIS 563

Week 5 Chapter 8 Video 2
Using IN, Comparison operators, ORDER BY
Dr. Anne Powell

Agenda

- ❖ 1. Explanation of subqueries / Subquery Rules
- ❖ **2. Using IN, comparison operators, and ORDER BY**
- ❖ 3. Nest subqueries at multiple levels / Using ANY and ALL keywords
- ❖ 4. Correlated subqueries / EXISTS operator
- ❖ 5. In-class examples

Subquery Types

1. Subqueries that operate on lists by use of the **IN** operator.
 - ❖ These subqueries can return a group of values.
 - ❖ The values must be from a single column of a table.
2. Subqueries that use a comparison operator (=, <, >, <>) – these subqueries must return only a single, *scalar* value.

In Video 3:

3. Subqueries that operate on lists by use of a comparison operator modified by the **ANY** or **ALL** optional keywords.

In Video 4:

4. *Correlational Subqueries* including those that use the **EXISTS** operator to test the *existence* of data rows satisfying specified criteria.

Example

This query lists employees by name that have dependents.

```
/* SQL Example */
SELECT LastName "Last Name", FirstName "First Name"
FROM Employee
WHERE EmployeeID IN
    (SELECT EmployeeID
     FROM Dependent);
```

Last Name	First Name
Bock	Douglas
Bordoloi	Bijoy
Boudreaux	Beverly
Simmons	Lester

Example 8.4

- ❖ Subquery returns *EmployeeID* from the *dependent* table.

```
(SELECT EmployeeID  
      FROM Dependent);
```

- ❖ Outer query selects employees where *EmployeeID* is found in the list produced by the subquery.

```
SELECT LastName "Last Name", FirstName "First Name"  
FROM Employee  
WHERE EmployeeID IN
```

The only difference in the use of the IN operator with subqueries is that the list does not consist of hard-coded values! (Because we don't know those values, so we must use a subquery).

Wait, what about a join?

- ❖ Couldn't we get the same result by using a JOIN?
 - ❖ Yes, we can.
- ❖ So, when do we use a subquery and when do we use a JOIN?...

General rules of thumb

- ❖ Subquery: use when the result displays columns from a single table
- ❖ Join query: use when the result displays columns from two or more tables
- ❖ Join query: may perform better when the existence of values must be checked with the EXISTS operator

Example 8.6

- ❖ Management needs a listing of employees that have male dependents, but not a listing of the actual dependents themselves.
- ❖ Write a subquery that retrieves *EmployeeID* column values from the *dependent* table where the dependent's gender is male.
- ❖ Write an outer query lists employees with an *EmployeeID* column value that is found in the listing produced by the subquery.
- ❖ The next slide shows the query.

Example 8.6 Cont'd

```
/* SQL Example 8.6 */  
SELECT LastName "Last Name", FirstName "First Name"  
FROM Employee  
WHERE EmployeeID IN  
(SELECT EmployeeID  
  FROM Dependent  
 WHERE Gender = 'M') ;
```

Last Name	First Name
Bock	Douglas
Boudreaux	Beverly
Simmons	Lester

SUBQUERIES AND THE IN Operator

- ❖ Conceptually, SQL Example 8.6 is evaluated in two steps.
- ❖ First, the inner query returns the identification numbers of those employees that have male dependents as shown here.

```
/* SQL Example 8.7 */  
COLUMN EmployeeID FORMAT A10;  
SELECT EmployeeID  
FROM Dependent  
WHERE Gender = 'M';
```

EMPLOYEEID

67555
33355
01885

SUBQUERIES AND THE IN Operator

- ❖ Next, these *EmployeeID* values are substituted into the outer query as the listing that is the object of the **IN** operator. So, from a conceptual perspective, the outer query now looks like the following.

```
/* SQL Example 8.8 */  
SELECT LastName "Last Name", FirstName "First Name"  
FROM Employee  
WHERE EmployeeID IN ('67555', '33355', '01885');
```

Last Name	First Name
Bock	Douglas
Boudreaux	Beverly
Simmons	Lester

In or not in

- ❖ The result of a subquery introduced with IN (or with NOT IN) is a list of zero or more values.
- ❖ After the subquery returns results, the outer query makes use of them.

Scenario example

The following query finds all the room numbers with an emergency room bed

```
SELECT RoomNumber  
FROM Bed  
WHERE bedtype IN  
(SELECT BedType  
FROM Bedclassification  
WHERE description like '%Emergency%');
```

The NOT IN Operator

- ❖ Like the IN operator, the NOT IN operator can take the result of a subquery as the operator object.
- ❖ Hospital management requires a listing of employees who do *not* have any dependents.

```
/* SQL Example 8.10 */  
SELECT LastName "Last Name", FirstName "First Name"  
FROM Employee  
WHERE EmployeeID NOT IN  
(SELECT EmployeeID  
FROM Dependent);
```

Last Name	First Name
Sumner	Elizabeth
Eakin	Maxwell
Webber	Eugene
more rows are displayed . .	

Comparison operators

Operators	Meanings
=	Equal to
<	Less than
>	Greater than
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to
<>	Not equal to
!>	Not greater than
!<	Not less than

Using Comparison Operators

- ❖ The general form of the **WHERE** clause with a comparison operator is similar to that used thus far in the text.

WHERE <expression> <comparison_operator> (subquery)

- ❖ A subquery with a comparison operator can only return a single or *scalar* value.
- ❖ This is also termed a *scalar subquery* because a single column of a single row is returned by the subquery.
- ❖ If a subquery returns more than one value, the Oracle Server will generate the “ ORA-01427: *single-row subquery returns more than one row* ” error message, and the query will fail to execute.

Comparison operators

- ❖ When using a subquery with a **comparison operator** the subquery can only return a **single/scalar value**

```
/* SQL Example */
SELECT EmployeeID
FROM Employee
WHERE Salary >
      (SELECT Salary
       FROM Employee
       WHERE Salary > 20000);
```

ERROR at line 4:
ORA-01427: single-row
subquery returns more than
one row

```
/* SQL Example */
SELECT LastName "Last Name", FirstName
      "First Name", Salary "Salary"
   FROM Employee
 WHERE Salary >
      (SELECT AVG(Salary)
       FROM Employee);
Last Name          First Name
Salary
-----
-
Simmons           Lester
$22,000
Boudreaux         Beverly
$17,520
Bock              Douglas
$16,250
more rows are displayed . . .
```

Aggregate Functions in Subqueries

- ❖ The aggregate functions (AVG, SUM, MAX, MIN, and COUNT) always return a *scalar* result table.
- ❖ A subquery with an aggregate function as the object of a comparison operator will always execute provided you have formulated the query properly.

ORDER BY

- ❖ The ORDER BY clause must go with the OUTER most query in subqueries. Note parentheses placement and ORDER BY placement in these examples:

Example 1:

```
SELECT LastName "Last Name", FirstName "First Name"  
FROM Employee  
WHERE EmployeeID IN  
  (SELECT EmployeeID  
   FROM Dependent)  
ORDER BY firstName;
```

ORDER BY examples

Example 2:

```
SELECT LastName "Last Name", FirstName "First Name"  
FROM Employee  
WHERE EmployeeID IN  
(SELECT EmployeeID  
FROM Dependent  
WHERE Gender = 'M')  
ORDER BY lastName;
```

Example 3:

```
SELECT RoomNumber  
FROM Bed  
WHERE bedtype IN  
(SELECT BedType  
FROM Bedclassification  
WHERE description like '%Emergency%')  
ORDER BY RoomNumber;
```

NOTE: ORDER BY attribute can be any attribute in the FROM entity in the OUTER query.

Additional Functions

CMIS 563

Week 6 Chapter 10 Video 2
Mathematical Functions
Dr. Anne Powell

Agenda

- ❖ 1. CHAR functions
- ❖ **2. Mathematical functions**
- ❖ 3. Conversion / DATE functions / DECODE

Learning Objectives

- ❖ Use character functions to manipulate CHAR type data.
- ❖ **Use mathematical functions to manipulate NUMBER type data.**
- ❖ Use conversion functions to convert data from one data type to another data type.
- ❖ Use date functions to manipulate DATE type data.
- ❖ Use the DECODE function to complete value substitutions.

Mathematical Functions (Single-Value Functions)

- ❖ Functions that manipulate NUMBER columns and data.
- ❖ Can be combined with the arithmetic operator symbols for addition, subtraction, multiplication, and division (+ - * /) to develop complex expressions.
- ❖ Values returned are generally accurate to 38 decimal digits.

Transcendental Functions

- ❖ The transcendental functions include ACOS, ASIN, ATAN, ATAN2, COS, COSH, EXP, LN, LOG, SIN, SINH, TAN, and TANH.
- ❖ The query shown below demonstrates how to generate values for selected transcendental functions from the *dual* pseudo table – the *dual* table exists in every Oracle database.

```
/* SQL Example 10.11 */
SELECT COS(0.5), EXP(1), LN(0.5), LOG(10,0.5)
FROM Dual;
COS(0.5)      EXP(1)      LN(0.5)  LOG(10,0.5)
----- ----- ----- -----
.877582562  2.71828183 -.69314718     -.30103
```

NVL Function for NULL Value Substitution

- NVL is a substitution function – allows you to substitute a specified value where the stored value in a row is NULL.
- Use to substitute "reasonable guess" or "average value" where a NULL value exists when values are unknown in a table.
- Use to highlight the absence of a value by substituting another value, such as zero for the NULL value.
- The general format of the NVL function is:
 - ❖ NVL (Value1, Value2)
- The NVL function works with character, date and other data types as well as numbers.

NVL Function for NULL Value Substitution

- If *Value1* is NULL, NVL returns *Value2*; otherwise, NVL returns *Value1*.
- The following query will produce the result by listing a value of 0 where *HoursWorked* is NULL.

```
/* SQL Example 10.13 */

SELECT EmployeeID "Employee ID", ProjectNumber "Project",
       NVL(HoursWorked, 0) "Hours"
  FROM ProjectAssignment
 WHERE ProjectNumber IN (1, 6, 7);

Employee ID      Project      Hours
-----  -----  -----
 23232            1        14.2
 66425            6        0.0
 23100            7        0.0
 66532            7        14.8
 67555            7       12.2
```

ABS Function

- The absolute value is a mathematical measure of magnitude.
- The general format of the ABS function is:
 - ❖ $\text{ABS}(\text{value})$
- Oracle provides the ABS function for use in computing the absolute value of a number or numeric expression.
- For example, suppose that the senior project manager has established 20 hours as the desired standard for working on assigned projects.
- The manager may wish to know which employees have deviated significantly from this standard, either by not working enough (less than 10 hours) or by exceeding expectations (more than 30 hours).

Example 10.14 – ABS Function

```
/* SQL Example 10.14 */
SELECT EmployeeID "Employee ID", HoursWorked "Hours",
       ABS(HoursWorked - 20) "Difference"
FROM ProjectAssignment
WHERE ABS(HoursWorked - 20) >= 10
ORDER BY ABS(HoursWorked - 20);
```

Employee ID	Hours	Difference
33344	9.5	10.50
88777	30.8	10.80
88505	34.5	14.50
33344	5.1	14.90
67555	35.4	15.40
33358	41.2	21.20

6 rows selected.

POWER and SQRT Functions

- The general format for these functions is:

```
POWER (NumericValue1, NumericValue2)  
SQRT (NumericValue))
```

- The POWER function raises *NumericValue1* to a specified positive exponent, *NumericValue2*.
- The SQRT function computes the square root of a *NumericValue*, expression, or NUMBER column value.

```
/* SQL Example 10.15 */  
SELECT POWER(10, 3), POWER(25, 0.5), SQRT(25)  
FROM Dual;  
POWER(10,3)  POWER(25,0.5)      SQRT(25)
```

ROUND and TRUNC Functions

- The general format of these functions is:

ROUND (NumericValue1, IntegerValue)

TRUNC (NumericValue1, IntegerValue)

- The ROUND function rounds *NumericValue1* to the specified number of digits of precision, an integer value shown in the formal definition as *IntegerValue*.
- The TRUNC function truncates digits from a number.

Example 10.16 – ROUND and TRUNC

```
/* SQL Example 10.16 */
SELECT EmployeeID "EmployeeID", HoursWorked "Hours",
       ProjectNumber "Project", ROUND(HoursWorked,0) "Rounded",
       TRUNC(HoursWorked,0) "Truncated"
FROM ProjectAssignment
ORDER BY EmployeeID;
```

EmployeeID	Hours	Project	Rounded	Truncated
01885	10.2	3	10	10
23100	10.3	4	10	10
23100		7		
23232	14.2	1	14	14
23232	10.6	2	11	10
23232		8		
33344	5.1	4	5	5
33344	9.5	5	10	9

more rows will be displayed...

SQL*Plus Reports

CMIS 563

Week 7 Chapter 9 Video 2
Control Break Reports
Dr. Anne Powell

Agenda

- ❖ 1. Report Formatting
- ❖ **2. Control Break Reports**
- ❖ 3. Master Detail Reports
- ❖ 4. Interactive Programs

Learning Objectives

- ❖ Create a SQL*Plus program command file and view command file settings.
- ❖ Select data to display in reports.
- ❖ Format all aspects of report layout.
- ❖ **Create a control break report including clearing breaks and using the BREAK and COMPUTE commands.**
- ❖ Use the SPOOL command to produce report listing files.
- ❖ Create Master-Detail reports including the use of variables in report titles and footers with the COLUMN command NEW_VALUE clause.
- ❖ Use the ACCEPT, PROMPT, and PAUSE commands for interactive program execution with substitution variables.
- ❖ Define user variables for interactive reporting and pass parameter values through the START command.

CONTROL BREAK REPORTS

- A control break report organizes information into meaningful groups.
- It may be desirable to organize a report into groups according to each employee's *EmployeeID* identifier. The modified example program is listed on the next set of slides produces a control break report.
- Additionally, the decision was made to remove the report header and report footer.

Example 9.2 – Control Break Report

```
/* SQL Example 9.2 */
-- Program: ch9-2.sql
-- Programmer: apowell; today's date
-- Description: A control break report
TTITLE 'Project Information by Employee'
BTITLE SKIP 1 CENTER 'Not for external dissemination.'
REPHEADER 'Project Report #2 - prepared by A. Powell' SKIP 2
REPFOOTER SKIP 3 '- Last Page of Report -'

SET LINESIZE 55
SET PAGESIZE 24
SET NEWPAGE 1

COLUMN "Employee ID" FORMAT A11
COLUMN "Hours Worked" FORMAT 999.99
```

Example 9.2 – Control Break Report

```
CLEAR BREAKS
```

```
BREAK ON "Employee ID" SKIP 2 ON REPORT
```

```
COMPUTE SUM OF "Hours Worked" ON "Employee ID"
```

```
COMPUTE SUM OF "Hours Worked" ON REPORT
```

```
SPOOL report9-2.lst
```

```
SELECT EmployeeID "Employee ID",
       ProjectNumber "Project #",
       HoursWorked "Hours Worked"
  FROM ProjectAssignment
 ORDER BY EmployeeID, ProjectNumber;
```

```
SPOOL OFF
```

The report produced by this program is shown on the next slide.

Project Information by Employee

Project Report #2 - prepared by A. Powell

Employee ID	Project #	Hours Worked
-------------	-----------	--------------

-----	-----	-----
-------	-------	-------

01885	3	10.20
-------	---	-------

*****	-----	-----
-------	-------	-------

sum		10.20
-----	--	-------

23100	4	10.30
-------	---	-------

	7	
--	---	--

*****	-----	-----
-------	-------	-------

sum		10.30
-----	--	-------

23232	1	14.20
-------	---	-------

	2	10.60
--	---	-------

	8	
--	---	--

Not for external dissemination.

The BREAK Command

- ❖ The BREAK command groups data rows for a control break report. The syntax of the BREAK command is:

```
BREAK ON (expression1, ON expression2, ...
           \row\page\report) ...

[SKIP n | [SKIP] PAGE]

[NODUPLICATES | DUPLICATES];
```

- ❖ The BREAK command can be used to break on an expression (such as a column or alias), row, page, report, or more than one of these at a time.

```
CLEAR BREAKS
```

```
BREAK ON "Employee ID" SKIP 2 ON REPORT
```

- ❖ The CLEAR BREAKS command clears any previously established breaks.

BREAK and ORDER BY

- ❖ The BREAK command works in conjunction with the ORDER BY clause of the SELECT statement.
- ❖ The ORDER BY clause in SQL Example 9.2 sorts output rows by *EmployeeID*, then by *ProjectNumber*.

```
ORDER BY EmployeeID, ProjectNumber;
```

- ❖ This sorts report data to make the report easier for managers to understand.
- ❖ A control break will still occur without an ORDER BY clause; however, the detail output rows will be in *natural order* by their occurrence within the *projectAssignment table*.

BREAK ON REPORT

- ❖ The ON REPORT clause of the BREAK command specifies that a break will also be enforced after all selected data rows have processed. The location of the ON REPORT clause is not important within the BREAK command. When the clause is specified, the report will always issue a final break at the end of the report. This break enables you to produce a final total of hours worked by all employees.
- ❖ The order of expressions or column names in the BREAK command is critical.
- ❖ The BREAK ON must be ordered from largest to smallest as is shown here.

BREAK ON Department ON Section ON EmployeeID

Output Style

- ❖ The style of output produced by SQL Example 9.2 is termed NONDUPLICATES, or **NODUP** because each group value (*EmployeeID*) is shown only once.
- ❖ This is the default BREAK output method.
- ❖ This style is easy for managers to read as it organizes data into groups.
- ❖ The NODUP default can be overwritten by specifying the keyword DUP with the BREAK command as is shown here. This will yield the purely relational, two-dimensional, matrix format for output.

```
BREAK ON "Employee ID" DUP SKIP 2
```

SKIP and PAGE Keywords

- ❖ To enhance the readability of a report, one or more blank rows can be inserted after each social security number grouping.
- ❖ SKIP inserts the blank rows. Our program specified to skip two lines prior to beginning the next report group.

```
BREAK ON "Employee ID" SKIP 2 ON REPORT
```

- ❖ Replace SKIP with PAGE to cause a *page eject* to occur after each grouping for printed reports.
- ❖ This will produce a report where a new page begins for each different *EmployeeID* grouping. Each group will be preceded by new column headings.

The COMPUTE Command

- A COMPUTE command computes subtotals and totals when used in conjunction with a BREAK command.
- Without a BREAK command, a COMPUTE command will not produce any results!
- When used with BREAK, a COMPUTE command displays values that are computed for the BREAK expression. The syntax of the COMPUTE command is shown here.

```
COMPUTE {group function} OF {column_name |  
    column_name_alias, . . . } ON  
{break column_name | ROW | PAGE |  
    REPORT} ;
```

Example Compute Commands

- ❖ Both of the COMPUTE commands shown here use the SUM aggregate function.
- ❖ The first COMPUTE command sums on the "Hours Worked" alias column name.
- ❖ The second COMPUTE command produces a report total for hours worked.

```
BREAK ON "Employee ID" SKIP 2 ON REPORT
COMPUTE SUM OF "Hours Worked" ON "Employee ID"
COMPUTE SUM OF "Hours Worked" ON REPORT
SELECT EmployeeID "Employee ID",
       ProjectNumber "Project #",
       HoursWorked "Hours Worked"
FROM ProjectAssignment
ORDER BY EmployeeID, ProjectNumber;
```

Example 9.2

- ❖ This is the final page of the report produced by the SQL Example 9.2 program – note the final total of hours worked.

Mon Oct 15

page 5

Project Information by Employee

Employee ID	Project #	Hours Worked
-------------	-----------	--------------

-----	-----	-----
-------	-------	-------

-----	-----	-----
-------	-------	-------

sum	306.90
-----	--------

- Last Page of Report -

Not for external dissemination.

Using Other Aggregate Functions

- ❖ You can specify other aggregate functions with the COMPUTE command. These include:
AVG, COUNT, MAXIMUM, MINIMUM, NUMBER,
STD (standard deviation), SUM, and VARIANCE.
- ❖ If management wants a report to display the average hours worked instead of the sum, then the AVG aggregate function would be used with the COMPUTE command as shown below.

```
COMPUTE AVG OF "Hours Worked" ON REPORT
```

Additional BREAK Command Details

- ❖ The BREAK command used thus far breaks on a column and on the report. You can also break on rows and pages.
- ❖ The BREAK ON ROW command can be used to change report spacing. The BREAK command shown below will insert a blank line between each row of the *projectAssignment* report.

```
BREAK ON ROW SKIP 1
```

- ❖ A column break and a row break can be used together. In conjunction, these two breaks create a double-spaced report that is still separated by column values. The command shown here will produce a double-spaced report that also breaks at the end of the report.

```
BREAK ON "Employee ID" SKIP 1 ON REPORT ON  
ROW SKIP 1
```

Viewing Current BREAK and COMPUTE Command Settings

- ❖ Only one BREAK command can be active at a time.
- ❖ You can interactively replace the current BREAK command by typing a new command at the SQL> prompt.
- ❖ To discover the active BREAK command type the command BREAK on a line by itself – SQL*Plus will display the break status.
- ❖ The default for the BREAK command is no duplicates (NODUP).

BREAK

<Oracle responds with this output>

```
break on report nodup
```

```
on EmployeeID skip 2 nodup
```

Viewing Current BREAK and COMPUTE Command Settings

- ❖ Unlike BREAK, the COMPUTE command is cumulative.
- ❖ While you are testing a program, you may accumulate quite a number of COMPUTE settings. You can display the current settings by simply typing the COMPUTE command at the SQL> prompt.

```
COMPUTE
```

```
COMPUTE sum LABEL 'sum' OF Hours Worked ON Employee ID  
COMPUTE avg LABEL 'avg' OF Hours Worked ON REPORT
```

- ❖ You can clear COMPUTE settings by typing CLEAR COMPUTE at the SQL> prompt or by placing the command within a program.
- ❖ When the command is used interactively, Oracle will respond as shown below.

```
CLEAR COMPUTE
```

computes cleared

Cardinality Basics

Dr. Anne Powell
CMIS563

More about relationships

- ❖ Degree
 - ❖ Number of entities involved
 - ❖ Typical
 - ❖ Unary (e.g. is married to)
 - ❖ Binary (most common)
 - ❖ The relationship “lives in” is of what degree? (as in Student “lives in” Residence Hall)

And another thing...

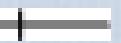
- ❖ Cardinality
 - ❖ Number of instances of one entity that are associated with another
 - ❖ Minimum and maximum – lower and upper bounds on the number of instances
 - ❖ Typically: 0, 1, many – deviations can be noted
 - ❖ Mandatory and optional

Cardinality

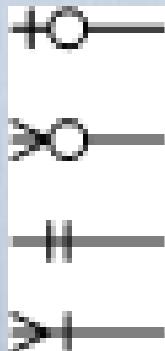
- ❖ Minimum

- ❖ Zero = may ()
 - ❖ One = must ()

- ❖ Maximum

- ❖ One ()
 - ❖ Many ()

- ❖ Possibilities:



Many-to-One



a one through many notation on one side of a relationship
and a one and only one on the other



a zero through many notation on one side of a relationship
and a one and only one on the other



a one through many notation on one side of a relationship
and a zero or one notation on the other



a zero through many notation on one side of a relationship
and a zero or one notation on the other

Many-to-Many



a zero through many on both sides of a relationship



a one through many on both sides of a relationship



a zero through many on one side and a one through many on the other

One-to-One

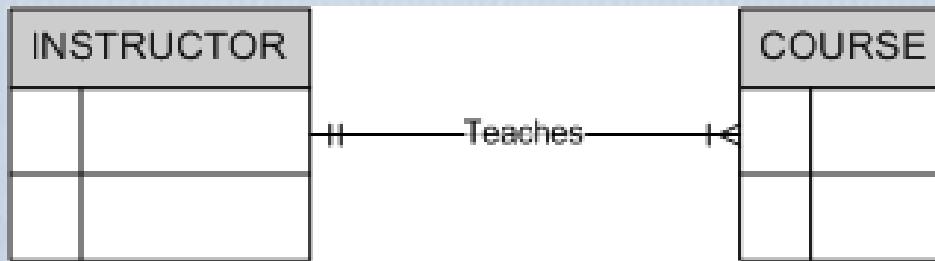
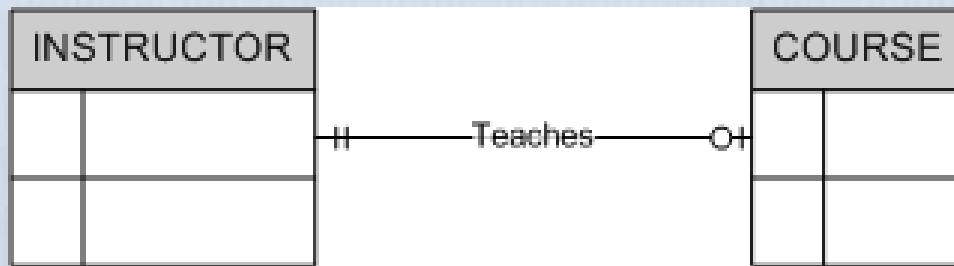


a one and only one notation on one side of a relationship
and a zero or one on the other

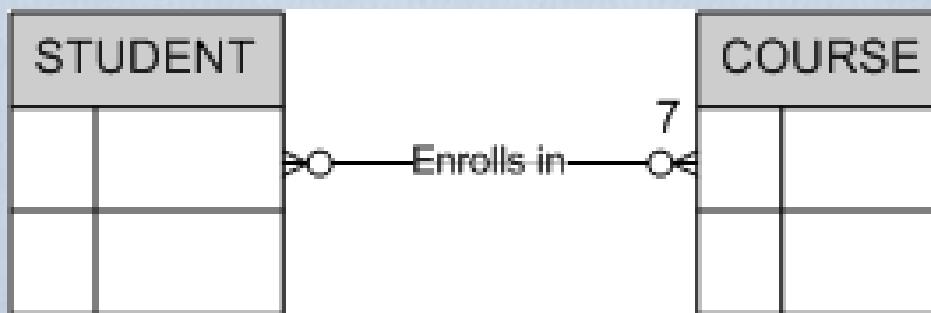
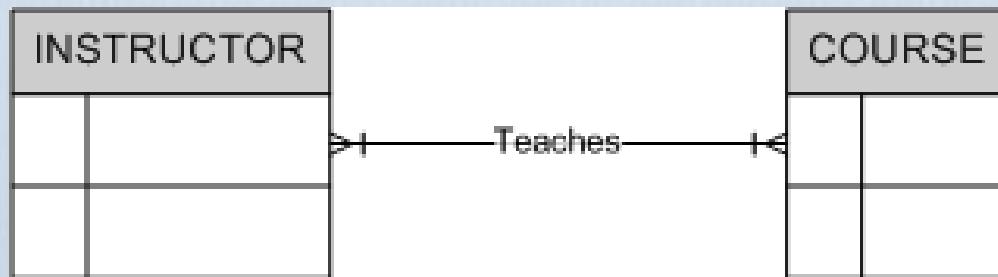


a one and only one notation on both sides

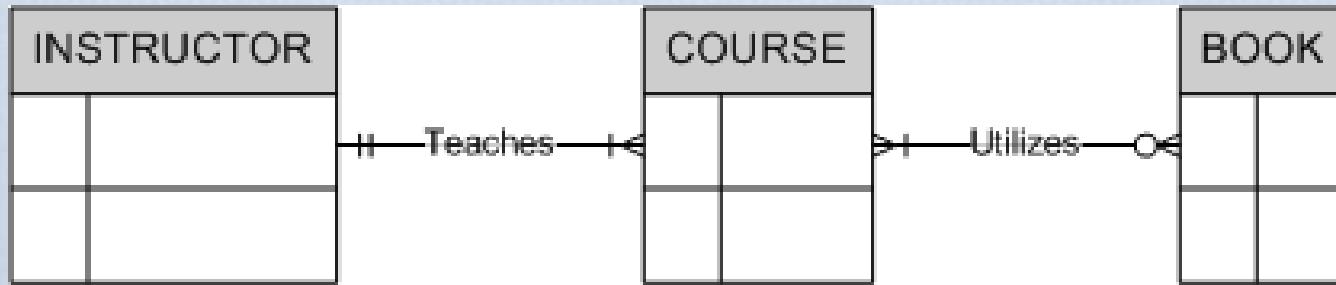
Reading ERDs



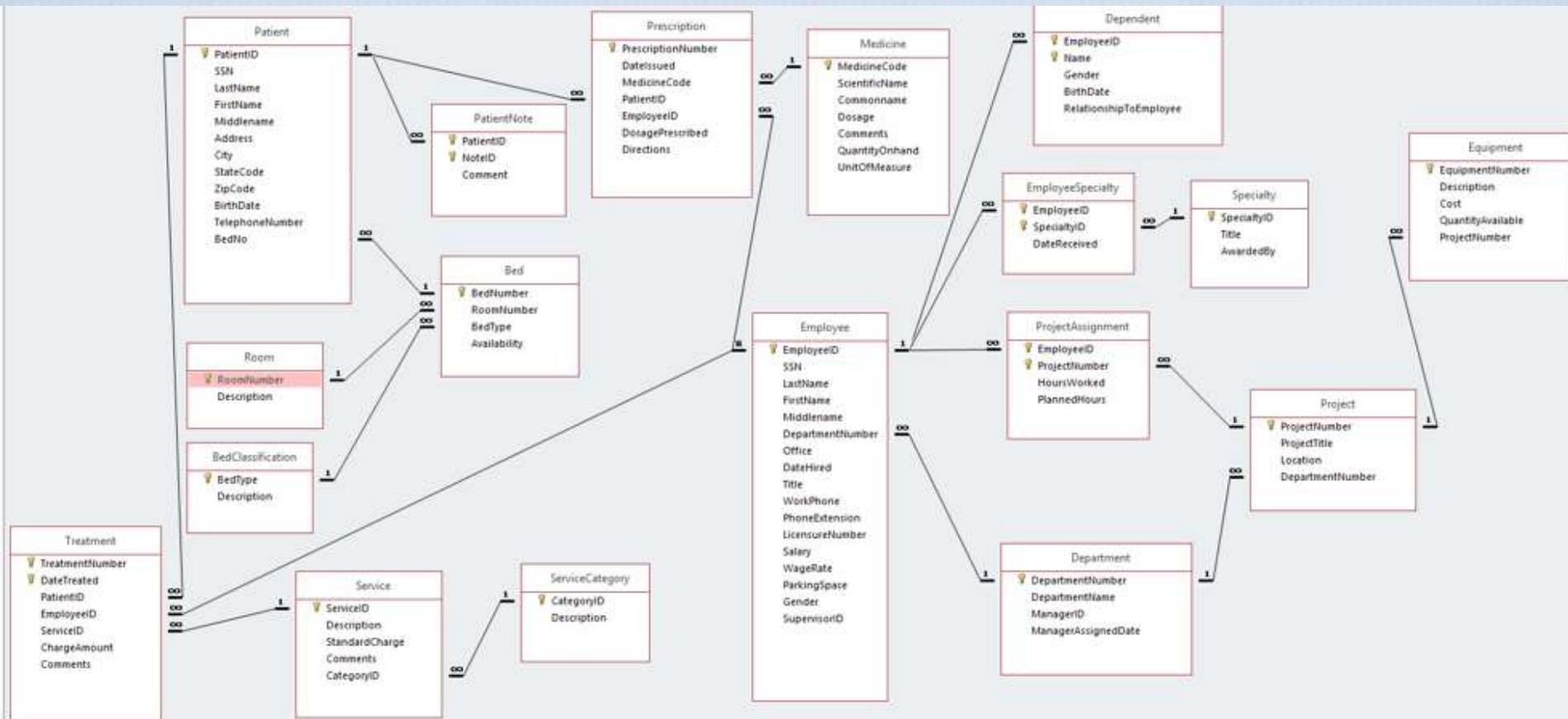
Reading ERDs (continued)



Binary Example



- ❖ What does this tell you?
- ❖ Read the relationships both directions.



Single Table Queries

Week 2 Chapter 4 Video 3
Common Errors
Dr. Anne Powell

Agenda

- ❖ 1. Basic SELECT commands
- ❖ 2. Column Formatting
- ❖ 3. Common Errors**
- ❖ 4. DISTINCT – WHERE commands
- ❖ 5. ORDER BY command
- ❖ 6. In-class practice

Common Errors

- ❖ There are syntactical rules that must be followed or Oracle gives an error message instead of the desired result table.
- ❖ Oracle communicates errors in SELECT statements by providing unique error numbers and accompanying error descriptions.

Common Errors – Invalid Column Name

- ❖ The SELECT statement in SQL Example 4.16 has the employee *SSN* column name spelled incorrectly.

```
SQL> /* SQL Example 4.16 */
SQL> SELECT Socsecno
      2 FROM Employee;
SELECT Socsecno
      *
ERROR at line 1:
ORA-00904: "SOCSECNO": invalid identifier
```

Common Errors – FROM Keyword Missing

- ❖ This SELECT statement is missing the FROM clause – no table name for retrieving data is given.
- ❖ Without a table name, the database management system does not know which table to query.

```
SQL> /* SQL Example 4.17 */  
SQL> SELECT SSN;  
SELECT SSN  
      *  
ERROR at line 1:  
ORA-00923: FROM keyword not found where expected
```

Common Errors – Unknown Command or Invalid Command Structure

- ❖ In SQL Example 4.18, the order of the SELECT and FROM clauses is reversed.
- ❖ Oracle is very confused by this command and simply returns an unknown command error message.

```
SQL> /* SQL Example 4.18 */  
SQL> FROM Employee SELECT SSN;  
SP2-0734: unknown command beginning "FROM Emplo..." - rest of line ignored.
```

Common Errors – Error in Placing Comma

- ❖ Some types of syntax errors cause Oracle to return error messages that are not particularly helpful in debugging the error or return no error message at all.
- ❖ In SQL Example 4.19, a comma is missing after the *LastName* column specification. Instead of reporting that a comma is missing, Oracle produces a result set that is missing one column of data and treats *FirstName* as a column alias name.

```
/* SQL Example 4.19 */  
SELECT EmployeeID, LastName FirstName  
FROM Employee;  
EMPLOYEEID FIRSTNAME  
----- -----  
67555      Simmons  
33355      Boudreaux  
33344      Adams  
more rows will be displayed...
```

Common Errors – Error in Placing Comma

- ❖ This example has a comma after the last column name – a simple syntax error.
- ❖ Oracle expects another column so it reports a missing expression error.

```
/* SQL Example 4.20 */  
SELECT SSN, LastName, FirstName,  
FROM Employee;
```

ERROR at line 2:

ORA-00936: missing expression

Common Error – Restricting Field to too small a value

- ❖ If you mistakenly make a field too small using the COLUMN statement, SQL will return the value as ### sign.
 - ❖ If MoSalary is 4 numbers, it shows, MoSalaries of more than 4 digits, show as #.

```
SQL> COLUMN MoSalary FORMAT 9999;
SQL> SELECT MoSalary
2  FROM Employee;

MOSALARY
-----
#####  
#####  
5500

#####  
#####  
#####  
#####  
4550
#####
```

GROUP BY

Week 3 Chapter 6 Video 3
GROUP BY COMMAND
Dr. Anne Powell

Agenda

1. Aggregate Function Rules – COUNT
2. Aggregate Function Rules – the rest
- 3. GROUP BY command**
4. HAVING command
5. Examples

Learning Objectives

- ❖ Write queries with aggregate functions: COUNT, AVG, SUM, MAX, and MIN.
- ❖ Use the GROUP BY clause to answer complex managerial questions.
- ❖ Use the GROUP BY clause with the WHERE and ORDER BY clauses.
- ❖ Use the GROUP BY clause with NULL values.
- ❖ Nest aggregate functions.
- ❖ Use the HAVING clause to filter out rows from a result table.

GROUP BY

- ❖ The GROUP BY clause enables you to use aggregate functions to answer more complex managerial questions such as:
 - ❖ What is the average salary of employees in each department?
 - ❖ How many employees work in each department?
 - ❖ How many employees are working on a particular project?
- ❖ General format of SELECT using GROUP BY.

```
SELECT ColumnName1, AggregateFunction(ColumnName2)
```

```
FROM TableName
```

```
GROUP BY ColumnName1;
```

Example

- The following query displays how many employees work for each department?

```
/* SQL Example 6.12 */
SELECT DepartmentNumber "Department",
       COUNT(*) "Employee Count"
FROM Employee
GROUP BY DepartmentNumber;
```

Department Employee Count

----- -----

1	2
2	3
3	5

more rows will be displayed . . .

GROUP BY Clause

- The column name used in a GROUP BY does not have to be listed in the SELECT clause; however, it must be a column name from one of the tables listed in the FROM clause.
- This is a rewrite of the previous example without specifying the *DepartmentNumber* column as part of the result table, but as you can see below, the results are rather cryptic without the *DepartmentNumber* column to identify the meaning of the aggregate count.

```
/* SQL Example 6.13 */
SELECT COUNT(*) "Employee Count"
FROM Employee
GROUP BY DepartmentNumber;
Employee Count
```

2

3

5

more rows will be displayed . . .

But . . .

- ❖ Does it really make for a good report?

```
/* SQL Example 6.13 */  
SELECT COUNT(*) "Employee Count"  
FROM Employee  
GROUP BY DepartmentNumber;
```

Employee Count

2

3

5

more rows will be displayed . . .

GROUP BY Rules

- ❖ If you have column name(s) AND aggregate function(s) in the SELECT clause, then you **MUST** also have a GROUP BY clause.

- ❖ When a column name(s) is given in the SELECT clause, it must match a column name(s) listed in the GROUP BY clause.

Indenting SQL Code

- ❖ The following keywords are your signal to start a new line.
 - ❖ SELECT
 - ❖ FROM
 - ❖ WHERE
 - ❖ GROUP BY
 - ❖ HAVING
 - ❖ ORDER BY

GROUP BY with WHERE

- ❖ The WHERE clause **eliminates data table rows from consideration before any grouping** takes place. This query lists average hours worked for specific employees.

```
SELECT EmployeeID "EID", AVG(HoursWorked) "Avg Hours Worked"  
FROM ProjectAssignment  
WHERE EmployeeID BETWEEN 30000 AND 70000  
GROUP BY EmployeeID;
```

EID	Avg Hours Worked
33344	12.5333333
33358	41.2
66425	
66432	15.5
66532	14.8
67555	23.9

6 rows selected.

GROUP BY with ORDER BY

```
SELECT DepartmentNumber "Department",
       AVG(Salary) "Average Salary"
  FROM Employee
 GROUP BY DepartmentNumber
 ORDER BY AVG(Salary);
```

Department Average Salary

7	\$4,895
6	\$8,050
4	\$10,778

more rows will be displayed . . .

Another Example of Order BY

- ❖ This is the previous query rewritten to display average department salaries from largest to smallest.

```
/* SQL Example 6.24 */
SELECT DepartmentNumber "Department",
       AVG(Salary) "Average Salary"
  FROM Employee
 GROUP BY DepartmentNumber
 ORDER BY AVG(Salary) DESC;
```

Department Average Salary

3	\$26,119
5	\$22,325
9	\$17,525

more rows will be displayed . . .

GROUP BY and NULL Values

- ❖ This SELECT statement sums based on *HoursWorked* and groups based on *EmployeeID*. Note the NULL value.

```
/* SQL Example 6.21 */  
SELECT EmployeeID "EID", SUM(HoursWorked) "Total Hours Worked"  
FROM ProjectAssignment  
GROUP BY EmployeeID;  
  
EID      Total Hours Worked  
-----  
01885          10.2  
23100          10.3  
23232          24.8  
33344          37.6  
33358          41.2  
66425  
66432          31  
More rows will display . . .
```

Using GROUP BY With a Expressions

- ❖ Management needs to know what new average salary figures will be if all employees receive a 25% raise – this uses an expression, *Salary * 1.25*, as opposed to a column name.

```
/* SQL Example 6.17 */
SELECT AVG(Salary) "Current Average Salary",
       AVG(Salary * 1.25) "New Average Salary"
  FROM Employee
 GROUP BY Salary * 1.25;
```

Current Average Salary	New Average Salary
\$2,200	\$2,750
\$4,550	\$5,688
\$4,800	\$6,000

more rows will be displayed . . .

Another Example

- ❖ This gives average salary by department if everyone receives a 25% raise.

```
/* SQL Example 6.18 */
SELECT DepartmentNumber "Department",
       AVG(Salary) "Current Average Salary",
       AVG(Salary * 1.25) "New Average Salary"
FROM Employee
GROUP BY DepartmentNumber;

```

Department	Current Average Salary	New Average Salary
1	\$15,625	\$19,531
2	\$13,300	\$16,625
3	\$26,119	\$32,649

more rows will be displayed . . .

Nested Aggregate Functions

- ❖ Oracle supports a *vector* aggregate function when used inside a *scalar* aggregate function; for example, AVG inside MAX.
- ❖ This gives the largest average salary – later you will learn to add the department number to the result table.

```
/* SQL Example 6.20 */  
COLUMN "Largest Average Salary" FORMAT $999,999;  
SELECT MAX(AVG(Salary)) "Largest Average Salary"  
FROM Employee  
GROUP BY DepartmentNumber;  
Largest Average Salary  
-----  
$26,119
```

MULTI-TABLE QUERIES

Week 4 Chapter 7 Video 3

Outer JOINs

Dr. Anne Powell

Agenda

- ❖ Outer Joins
- ❖ Left Joins / Right Joins
- ❖ Natural Joins

OUTER JOIN Operations

- ❖ A LEFT OUTER JOIN includes all rows in the left table of a JOIN specification and any rows that match a left table row from the right table of a JOIN specification.
- ❖ A RIGHT OUTER JOIN includes all rows in the right table of a JOIN specification and any rows that match the right table row from the left table of the JOIN.
- ❖ Oracle also supports the FULL OUTER JOIN. We do not cover the FULL OUTER JOIN; however, it is used to display data where the result table would be very sparse.

LEFT OUTER JOIN – FROM Clause

```
/* SQL Example 7.21 - Join conditions in the FROM clause */
SELECT LastName "Last Name", FirstName "First Name",
       Name "Dependent", RelationshipToEmployee "Relationship"
  FROM Employee e LEFT OUTER JOIN Dependent d
    ON (e.EmployeeID = d.EmployeeID);
```

Last Name	First Name	Dependent	Relationship
Bock	Douglas	Jeffery	SON
Bock	Douglas	Deanna	DAUGHTER
Bock	Douglas	Michelle	DAUGHTER
Bock	Douglas	Mary Ellen	SPOUSE
Bock	Douglas	Rachael	DAUGHTER
Sumner	Elizabeth		
Bordoloi	Bijoy	Mita	SPOUSE
Bordoloi	Bijoy	Monica	DAUGHTER
more rows will display . . .			

- ❖ Note employee Elizabeth Sumner has no dependents.

LEFT OUTER JOIN – WHERE Clause

- ❖ Here SQL Example 7.21 has the LEFT OUTER JOIN specified with the WHERE clause by use of the plus (+) symbol. The *dependent* table is being outer joined with the *employee* table.

```
/* Join conditions in the WHERE clause */
SELECT LastName "Last Name", FirstName "First Name",
       Name "Dependent", RelationshipToEmployee
      "Relationship"
FROM Employee e, Dependent d
WHERE e.EmployeeID = d.EmployeeID(+);
```

RIGHT OUTER JOIN

- ❖ Not done as frequently
- ❖ The same as the LEFT OUTER JOIN, only it lists all rows of the Right table with any matches from the Left table.
- ❖ Therefore, you can just use the LEFT OUTER JOIN and switch your tables around!

RIGHT OUTER JOIN – FROM Clause

```
SELECT LastName "Last Name", FirstName "First Name",
       Name "Dependent", RelationshipToEmployee "Relationship"
FROM Dependent d RIGHT OUTER JOIN Employee e
  ON (d.EmployeeID = e.EmployeeID);
```

Last Name	First Name	Dependent	Relationship
Bock	Douglas	Jeffery	SON
Bock	Douglas	Deanna	DAUGHTER
Bock	Douglas	Michelle	DAUGHTER
Bock	Douglas	Mary Ellen	SPOUSE
Bock	Douglas	Rachael	DAUGHTER
Sumner	Elizabeth		
Bordoloi	Bijoy	Mita	SPOUSE
Bordoloi	Bijoy	Monica	DAUGHTER
more rows will display . . .			

OUTER JOINS and NULL values

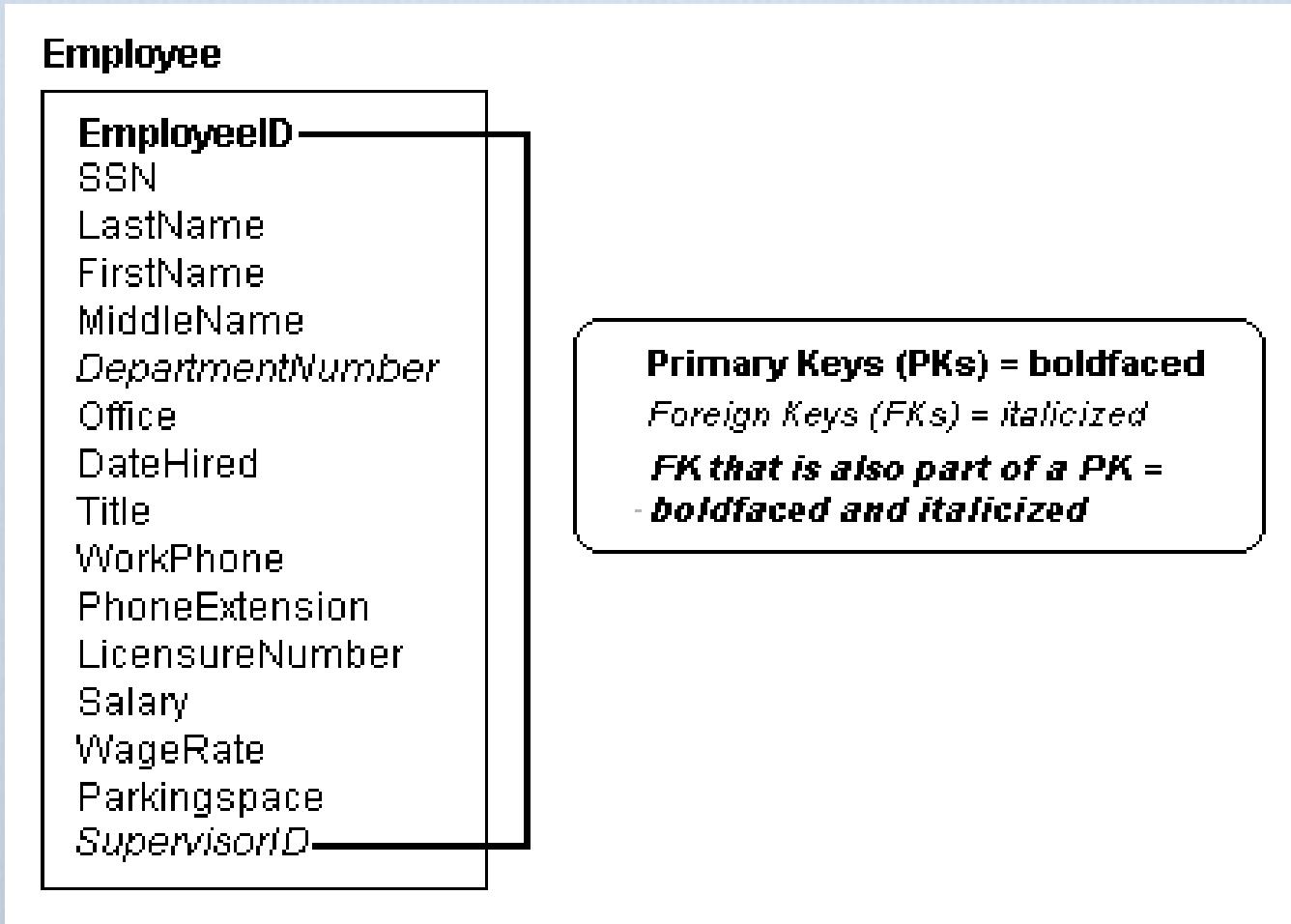
- ❖ Management might desire a listing of employees with no dependents in order to satisfy some governmental reporting requirement.
- ❖ We can take advantage of the fact that the *department* table's *Name* column will be NULL for employees with no dependents:

```
/* SQL Example */  
SELECT LastName || ', ' || FirstName "Employees Without  
Dependents"  
FROM Employee e LEFT OUTER JOIN Dependent d  
    ON (e.EmployeeID = d.EmployeeID)  
WHERE d.Name IS NULL;  
Employees Without Dependents  
-----  
Sumner, Elizabeth  
Eakin, Maxwell  
Webber, Eugene
```

more rows will display . . .

SELF-JOIN Operations

- SELF-JOIN operation – used when a relationship exists among rows that are stored within a single table.



SELF-JOIN Operations

```
/* SQL Example 7.29 */
SELECT e1.LastName || ' ' || e1.FirstName "Supervisor",
       e2.LastName || ' ' || e2.FirstName "Employee"
  FROM Employee e1 JOIN Employee e2
    ON (e1.EmployeeID = e2.SupervisorID)
 ORDER BY e2.SupervisorID DESC;
```

Supervisor

Employee

Becker Roberta

Brockwell Mary Ellen

Becker Roberta

Young Yvonne

Becker Roberta

Simmons Leslie

Simmons Lester

Boudreaux Beverly

Simmons Lester

Eakin Maxwell

Simmons Lester

Becker Roberta

Simmons Lester

Sumner Elizabeth

Simmons Lester

Webber Eugene

Simmons Lester

Klepper Robert

Klepper Robert

Zumwalt Mary

Klepper Robert

Quattromani Toni

more rows will display . . .

NATURAL Join

- ❖ Simplifies the syntax of an equijoin.
- ❖ This example gives a NATURAL join of the *department* and *project* tables.

```
/* SQL Example 7.31 */  
SELECT DepartmentNumber "Dept #", d.DepartmentName  
    "Department", p.ProjectTitle "Project"  
FROM Department d NATURAL JOIN Project p  
WHERE DepartmentNumber = 6;
```

Dept #	Department	Project
6	Pediatrics-Gynecology	New Pediatric Monitors
6	Pediatrics-Gynecology	Child Care Center

NATURAL Join Limitations

- ❖ You cannot specify a LOB column with a NATURAL JOIN.
- ❖ Columns involved in the join cannot be qualified by a table name or alias as shown in this example in either the SELECT, FROM, WHERE, or other clause.

```
/* SQL Example 7.33 */

SELECT d.DepartmentNumber "Dept #", d.DepartmentName
      "Department", p.ProjectTitle "Project"
FROM Department d NATURAL JOIN Project p
WHERE DepartmentNumber = 6;
```

ERROR at line 1:

ORA-25155: column used in NATURAL join cannot have qualifier

UNION and UNION ALL

- ❖ SQL union is used to combine the result of two SELECT statements.
- ❖ Using UNION will remove any duplicates resulting from the two SELECT statements.
- ❖ The two SELECT statements must have the same number of fields chosen with the same datatypes.
- ❖ Using UNION ALL will not filter out duplicate rows between the two SELECT statements.

Subqueries

CMIS 563

Week 5 Chapter 8 Video 3
Using ANY, ALL, Nesting subqueries
Dr. Anne Powell

Agenda

- ❖ 1. Explanation of subqueries / Subquery Rules
- ❖ 2. Using IN, ORDER BY, and comparison operators
- ❖ **3. Using ANY and ALL keywords / Nest subqueries at multiple levels**
- ❖ 4. Correlated subqueries / EXISTS operator
- ❖ 5. In-class examples

Comparison Operators Modified with the ALL or ANY Keywords

- The ALL and ANY keywords can modify a comparison operator to allow an outer query to accept multiple values from a subquery.
- The general form of the WHERE clause for this type of query is shown here.

```
WHERE <expression> <comparison_operator> [ALL |  
ANY] (subquery)
```

- Subqueries that use these keywords may also include GROUP BY and HAVING clauses.

An *ALL* Keyword Subquery

- ❖ The **ALL** keyword modifies the greater than (>) comparison operator to mean greater than all values – list employees with a salary greater than ALL of the salaries of department 8 employees. Why include the NOT NULL condition?

```
/* SQL Example 8.18 */  
SELECT LastName "Last Name", FirstName "First Name",  
       Salary "Salary"  
FROM Employee  
WHERE Salary > ALL  
(SELECT Salary  
      FROM Employee  
     WHERE DepartmentNumber = 8 AND Salary IS NOT NULL);
```

Last Name	First Name	Salary
Becker	Robert	\$23,545
Jones	Quincey	\$30,550
Barlow	William	\$27,500
...		

6 rows selected.

An *ANY* Keyword Subquery

- ❖ The **ANY** keyword is not as restrictive as the **ALL** keyword.
- ❖ When used with the greater than comparison operator, "> ANY" means greater than some value.
- ❖ Suppose hospital management needs the employee name and salary of any employee that has a salary that is greater than that of *any* employee with a salary that exceeds \$23,000. This query is not the same as asking for a listing of employees with salaries that exceed \$23,000.

An *ANY* Keyword Subquery

- ❖ Examine this query in detail.

```
/* SQL Example 8.20 */
SELECT LastName "Last Name", FirstName "First Name",
       Salary "Salary"
FROM Employee
WHERE Salary > ANY
      (SELECT Salary
       FROM Employee
       WHERE Salary > 23000);
```

Last Name	First Name	Salary
Smith	Susan	\$32,500
Jones	Quincey	\$30,550
Barlow	William	\$27,500

An *ANY* Keyword Subquery

- ❖ For each employee, the inner query finds a list of salaries that are greater than \$23,000.
- ❖ Four employees have such a salary.

```
/* SQL Example 8.21 */  
SELECT Salary  
    FROM Employee  
   WHERE Salary > 23000;  
  
SALARY  
-----  
$23,545  
$30,550  
$27,500  
$32,500
```

An *ANY* Keyword Subquery

- ❖ The outer query looks at all the values in the list and determines whether an employee earns more than *any* of the salaries in the intermediate result table (here, this means more than \$23,545).
- ❖ The employees listed in the final result table all earn more than the \$23,545 listed in the intermediate result table.

Last Name	First Name	Salary
Smith	Susan	\$32,500
Jones	Quincey	\$30,550
Barlow	William	\$27,500

ALL vs ANY

- ❖ What do you get in this query? Why? Can you explain the results?

```
SELECT LastName "Last Name", FirstName "First  
Name",  
      Salary "Salary"
```

```
FROM Employee
```

WHERE Salary > ALL

```
(SELECT Salary  
     FROM Employee  
    WHERE Salary > 23000);
```

An " $= ANY$ "(Equal Any) Example

- ❖ The " $= ANY$ " operator is exactly equivalent to the IN operator.
- ❖ For example, to find the names of employees that have male dependents, you can use either IN or " $= ANY$ " – both of the queries shown below will produce an identical result table.

/* SQL Example 8.22 */	/* SQL Example 8.23 */
<pre>SELECT LastName "Last Name", FirstName "First Name" FROM Employee WHERE EmployeeID IN (SELECT EmployeeID FROM Dependent WHERE Gender = 'M');</pre>	<pre>SELECT LastName "Last Name", FirstName "First Name" FROM Employee WHERE EmployeeID = ANY (SELECT EmployeeID FROM Dependent WHERE Gender = 'M');</pre>

A " \neq ANY" (Not Equal Any) Example

- ❖ The " \neq ANY" (not equal any) is not equivalent to the NOT IN operator.
- ❖ If a subquery of employee salaries produces an intermediate result table with the salaries \$2,200, \$22,000, \$23,000, then the WHERE clause shown here means "NOT \$2,200" AND "NOT \$22,000" AND "NOT \$23,000."

```
WHERE SALARY NOT IN (2200, 22000, 23000);
```

- ❖ However, the " \neq ANY" comparison operator and keyword combination shown in this next WHERE clause means "NOT \$2,200" OR "NOT \$22,000" OR "NOT \$23,000."

```
WHERE SALARY != ANY (2200, 22000, 23000);
```

A " \neq ANY" (Not Equal Any) Example

```
SELECT Salary FROM Employee  
WHERE SALARY NOT IN (2200, 22000, 23000)  
ORDER BY Salary;
```

SALARY

4550

4800

4895

5500

6500

...

...

18 rows selected.

A " \neq ANY" (Not Equal Any) Example

```
SELECT Salary FROM Employee  
WHERE SALARY !=ANY (2200, 22000, 23000)  
ORDER BY Salary;
```

SALARY

2200

2200

4550

4800

4895

.....

.....

22 rows selected.

A " \neq ANY" (Not Equal Any) Example

- ❖ Suppose a human resource manager needs a listing of employees that do not have dependents. You might write the following erroneous query – note that this query returns ALL employee names. The solution is to use the **NOT IN** operator.

```
/* SQL Example 8.24 */
SELECT LastName "Last Name", FirstName "First Name"
FROM Employee
WHERE EmployeeID != ANY
  (SELECT DISTINCT EmployeeID
   FROM Dependent);
Last Name      First Name
-----
Simmons        Lester
Boudreaux      Beverly
Adams          Adam
. . .
24 rows selected.
```

MULTIPLE LEVELS OF NESTING

- ❖ Subqueries may themselves contain subqueries.
- ❖ When the WHERE clause of a subquery has as its object another subquery, these are termed *nested subqueries*.
- ❖ Oracle places no practical limit on the number of queries that can be nested in a WHERE clause.
- ❖ Consider the problem of producing a listing of employees that worked more than 10 hours on the project named *Remodel ER Suite*.

Example 8.11

```
/* SQL Example 8.11 */
SELECT LastName "Last Name", FirstName "First Name"
FROM Employee
WHERE EmployeeID IN
    (SELECT EmployeeID
     FROM ProjectAssignment
     WHERE HoursWorked > 10 AND ProjectNumber IN
          (SELECT ProjectNumber
           FROM Project
           WHERE ProjectTitle = 'Remodel ER Suite'));
```

Last Name	First Name
Bordoloi	Bijoy
Klepper	Robert
Smith	Susan

Understanding Subquery 8.11

- ❖ In order to understand how this query executes, begin your examination by figuring out which tables are needed to determine the employee names who have worked more than 10 hours on a certain project.
 - ❖ EMPLOYEE is needed to get the lastName and firstName.
 - ❖ PROJECTASSIGNMENT is needed to get the hours worked.
 - ❖ PROJECT is needed to get data on the title ‘Remodel ER Suite’.on with the lowest subquery by executing it independently of the outer queries.
- ❖ Because we want to display the names, we know EMPLOYEE query will be the outer most query. Looking at the ERD, we know PROJECTASSIGNMENT is the table connecting EMPLOYEE to the PROJECT title. So PROJECT must be the inner most query

Understanding Subquery 8.11

- ❖ Begin your examination with the lowest subquery by executing it independently of the outer queries.

```
/* SQL Example 8.12 */  
SELECT ProjectNumber  
FROM Project  
WHERE ProjectTitle = 'Remodel ER Suite';
```

PROJECTNUMBER

Understanding Subquery 8.11

- ❖ Now substitute the project number into the **IN** operator list for the intermediate subquery and execute it.
- ❖ The intermediate result table lists three *EmployeeID* column values for employees that worked more than 10 hours on project 4.

```
/* SQL Example 8.13 */  
SELECT EmployeeID  
FROM ProjectAssignment  
WHERE HoursWorked > 10 AND ProjectNumber IN (4);
```

EMPLOYEEID

23100

66432

88505

Understanding Subquery 8.11

- ❖ Finally, substitute these three *EmployeeID* column values into the **IN** operator listing for the outer query in place of the subquery.

```
/* SQL Example 8.14 */  
SELECT LastName "Last Name", FirstName "First Name"  
FROM Employee  
WHERE EmployeeID IN ('23100', '66432', '88505');
```

Last Name	First Name
Bordoloi	Bijoy
Klepper	Robert
Smith	Susan

So why not just use this:

```
SELECT LastName "Last Name", FirstName "First  
Name"  
  
FROM Employee e JOIN ProjectAssignment pa ON  
(e.employeeID = pa.employeeID) JOIN Project p  
ON (p.projectNumber = pa.projectNumber)  
  
WHERE e.EmployeeID IN ('23100', '66432', '88505')  
AND pa.hoursWorked > 10 AND p.projectNumber =  
4;
```

Additional Functions

CMIS 563

Week 6 Chapter 10 Video 3
Conversions / DATE functions / DECODE
Dr. Anne Powell

Agenda

- ❖ 1. CHAR functions
- ❖ 2. Mathematical functions
- ❖ **3. Conversion / DATE functions / DECODE**

Learning Objectives

- ❖ Use character functions to manipulate CHAR type data.
- ❖ Use mathematical functions to manipulate NUMBER type data.
- ❖ **Use conversion functions to convert data from one data type to another data type.**
- ❖ **Use date functions to manipulate DATE type data.**
- ❖ **Use the DECODE function to complete value substitutions.**

Conversion Functions

- ❖ Primary purpose – to convert one data type to another data type. This is termed *explicit conversion* and allows you as the programmer exact control over how Oracle treats data.

Function	Use/Definition
CAST	Convert an Oracle data type to another Oracle data type.
TO_CHAR (and TO_NCHAR)	Converts a character, numeric, or datetime value to a character string.
TO_DATE	Converts a character string or number to a datetime value.

TO_CHAR and TO_DATE Functions

- ❖ These functions format output and convert data from one data type to another. The general form of these functions:

```
TO_CHAR (ValueToConvert, { 'FormatString', 'NLSparameter' })  
TO_DATE (CharacterString, { 'FormatString', 'NLSparameter' })
```

- ❖ The TO_CHAR function converts NCHAR, NVARCHAR2, CLOB, NCLOB, **NUMBER**, BINARY, FLOAT, BINARY_DOUBLE, and **DATE** data types to a VARCHAR2 character string.
- ❖ When the *FormatString* parameter is omitted, the date conversion is to the default date format – generally DD-MON-YY.

TO_CHAR and TO_DATE Functions

- *NLSparameter* – an optional parameter value that specifies the national language to use if one other than the current default is required.
- TO_DATE function – the mirror-image of TO_CHAR and converts a date value to a character string.
- Both of these functions can be used to format output by using a wide range of formatting options.

Example 10.19 – TO_CHAR Function

- ❖ This query converts (formats) *ChargeAmount*, a NUMBER column to character data.

```
/* SQL Example 10.19 */

SELECT TO_CHAR(ChargeAmount, '$99,999.99') "Charge"
FROM Treatment
WHERE PatientID = '100002';
```

Charge

\$35.00

\$30.00

Numeric Formatting Characters

Format	Example	Use/Description
,	(comma) 9,999	Specifies to position a comma within a numeric value that is a formatted string.
.	(period) 999.99	Specifies to locate a decimal point within a formatted string.
\$	(dollar sign) \$999.99	Includes a dollar sign as a leading character.
0	(zero) 0999 9990	Includes leading zeros. Includes trailing zeros.
9	(nine) 9999	Specifies the maximum number of numeric positions to allot for numbers converted to characters. A minus sign is displayed if the number is negative. Leading zeros are blank unless the value zero is a significant digit.
U	(letter U) U9999	Specifies to include the Euro dual currency symbol. (current value of the NLS_DUAL_CURRENCY parameter).

TO_CHAR and TO_DATE Functions

- ❖ The default date format for use with TO_CHAR and TO_DATE can be set by assigning a value to the NLS_DATE_FORMAT (national language support date format) parameter.
- ❖ The ALTER SESSION command shown here sets the format from the default of DD-MON-YY to DD-MON-YYYY to display a full, four-digit year.

```
/* SQL Example 10.20 */
```

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY';
Session altered.

SQL> SELECT SYSDATE
  2  FROM Dual;

SYSDATE
-----
24-MAY-2022
```

Date Formatting Characters

FORMAT	USE/DESCRIPTION
D	Day of week
DD	Days in month
DDD	Days in year
DY	Three-letter day abbreviation
DAY	Day spelled out – padded with blank characters to 9 characters in length
HH, HH12, and HH24	Hour of day; Hour of day (hours 1 – 12); Hour of day (hours 1 – 24);
MI	Minute (0 - 59)
MM	Month – numbered 01 to 12
MON	Month spelled out in abbreviated 3-character format
MONTH	Month spelled out - padded with blank characters to 9 characters in length
SS	Second (0– 59)
Y, YY, YYY, and YYYY	Year in 1, 2, 3, or 4-year formats

Example 10.21 – TO_CHAR Formatting

- ❖ The SELECT statement in SQL Example 10.21 demonstrates formatting output for the *patient* table's *BirthDate* column.

```
/* SQL Example 10.21 */  
SELECT BirthDate "Birth Date",  
       TO_CHAR(BirthDate, 'MONTH DD, YYYY') "Spelled Out"  
FROM Patient;
```

Birth Dat Spelled Out

```
-----  
02-JAN-96 JANUARY 02, 1996  
02-FEB-67 FEBRUARY 02, 1967  
14-FEB-79 FEBRUARY 14, 1979
```

More rows will be displayed ...

DATE Functions

Function	Use/Definition
ADD_MONTHS	Adds the specified number of months to the specified date and returns that date.
CURRENT_DATE	Returns the current date in the session time zone, in a value in the Gregorian calendar of datatype DATE.
CURRENT_TIMESTAMP	Returns the current date and time in the session time zone, in a value of datatype TIMESTAMP WITH TIME ZONE. The time zone offset reflects the current local time of the SQL session.
ROUND	Rounds date values in the same fashion as the function rounds numbers.
SYSDATE	Returns the current system date and time.
TRUNC	Truncates times to midnight of the date specified.

SYSDATE Function

- The SYSDATE function returns the current date and time from the computer's operating system.
- You can select SYSDATE from any table, so in this respect, SYSDATE is a sort of pseudo column.
- In the example shown here, the SYSDATE is selected from the *employee* table.

```
❖ /* SQL Example 10.26 */  
❖ SELECT SSN, SYSDATE  
❖ FROM Employee;  
❖ SSN          SYSDATE  
❖ ----- -----  
❖ 981789642 24-MAY-22  
❖ 890536222 24-MAY-22  
❖ 890563287 24-MAY-22
```

CURRENT_DATE and CURRENT_TIMESTAMP Functions

- ❖ SQL Example 10.26 shows the values returned for CURRENT_DATE and CURRENT_TIMESTAMP functions from the *dual* pseudo table. The time returned is accurate to a very small fraction of a second.

```
SELECT CURRENT_DATE "Date", CURRENT_TIMESTAMP "Date and Time"  
FROM Dual;
```

Date	Date and Time
-----	-----

```
24-MAY-22 24-MAY-22 02.37.27.331871 PM -05:00
```

Date Arithmetic

- Oracle provides the capability to perform date arithmetic.
- Example: Adding seven (7) to a value stored in a date column will produce a date that is one week later than the stored date.
- Likewise, subtracting 7 from a stored date will produce a date that is a week earlier than the stored date.
- You can also subtract or compute the difference between two date values.
- Subtracting two date columns will produce the number of days between the two dates.

Example 10.27 – Date Arithmetic

- How long has the manager of department 3 managed?

```
/* SQL Example 10.27 */  
COLUMN "Manager ID" FORMAT A10;  
COLUMN "Last Name" FORMAT A15;  
COLUMN "Number Days" FORMAT 99999999999;  
SELECT d.ManagerID "Manager ID", LastName "Last Name",  
       SYSDATE - d.ManagerAssignedDate "Number Days"  
FROM Department d JOIN Employee e  
  ON (d.ManagerID = e.EmployeeID)  
WHERE d.DepartmentNumber = 3;
```

Manager ID	Last Name	Number Days
10044	Sumner	251

Your answer will vary depending on when you execute the query (and which database with dates you have downloaded) .

ADD_MONTHS Function

- Situation: A human resources manager needs to know the ten-year anniversary dates for current department managers in order to determine if any of the managers are eligible for a service award.
- You could execute a query that adds 3,650 days (10 years at 365 days/year) to the *ManagerAssignedDate* column of the *department* table; however, this type of date arithmetic would fail to take into consideration leap years that have 366 days.
- The ADD_MONTHS function solves this problem by adding the specified number of months to a specified date. The format of the function is:

❖ ADD_MONTHS(StartDate, NumberofMonths)

Example 10.28 – ADD_MONTHS Function

- This query displays the required ten-year anniversary information.

```
/* SQL Example 10.28 */  
  
SELECT d.ManagerID "Manager ID", LastName "Last Name",  
       ManagerAssignedDate "Start Date",  
       ADD_MONTHS(ManagerAssignedDate, 120) "10 Yr Anniversary"  
  FROM Department d JOIN Employee e  
    ON (d.ManagerID = e.EmployeeID)  
 ORDER BY d.DepartmentNumber;
```

Manager ID	Last Name	Start Dat	10 Yr Ann
23232	Eakin	21-AUG-17	21-AUG-27
23244	Webber	10-JAN-21	10-JAN-31
10044	Sumner	15-SEP-21	15-SEP-31

More rows will be displayed ...

DECODE Function

- The DECODE function enables you to use If-Then-Else logic when displaying values. The general format is:

```
DECODE(Expression, Search1, Result1, Search2, Result2, ..., Else  
Default)
```

- The *Expression* can be a column value of any data type, or a result from some type of computation or function.
- The *Expression* is compared to *Search1* and if *Expression = Search1*, then *Result1* is returned. If not, then the search continues to compare *Expression = Search2* in order to return *Result2*, etc.
- If the *expression* does not equal any of the *Search* values, then the *Default* value is returned. The *Else Default* can be a column value or the result of some type of computation or function.

Example 10.30 – DECODE and TRUNC Functions

- For employees working more than 30 hours the TRUNC function yields a value of 1 or more. The DECODE function is searching for a zero value.

```
/* SQL Example 10.30 */

COLUMN "Employee ID" FORMAT A11;

SELECT EmployeeID "Employee ID", HoursWorked "Hours Worked",
       DECODE(TRUNC(HoursWorked/30), 0, 'Worked OK', 'Worked Very Hard')
  "Work Status"
FROM ProjectAssignment
WHERE ProjectNumber = 4;
```

Employee ID	Hours Worked	Work Status
23100	10.3	Worked OK
33344	5.1	Worked OK
66432	19.2	Worked OK
88505	34.5	Worked Very Hard

Summary

- ❖ In this chapter you learned to use various function such as TO_CHAR and TO_DATE to manipulate character, numeric, and date data.
- ❖ You familiarized with many of the mathematical functions.
- ❖ You learned to concatenate.
- ❖ You used DECODE to apply If-Then-Else logic to a value stored in a column.

SQL*Plus Reports

CMIS 563

Week 7 Chapter 9 Video 3
Master Detail Reports
Dr. Anne Powell

Agenda

- ❖ 1. Report Formatting
- ❖ 2. Control Break Reports
- ❖ 3. Master Detail Reports**
- ❖ 4. Interactive Programs

Learning Objectives

- ❖ Create a SQL*Plus program command file and view command file settings.
- ❖ Select data to display in reports.
- ❖ Format all aspects of report layout.
- ❖ Create a control break report including clearing breaks and using the BREAK and COMPUTE commands.
- ❖ **Use the SPOOL command to produce report listing files.**
- ❖ **Create Master-Detail reports including the use of variables in report titles and footers with the COLUMN command NEW_VALUE clause.**
- ❖ Use the ACCEPT, PROMPT, and PAUSE commands for interactive program execution with substitution variables.
- ❖ Define user variables for interactive reporting and pass parameter values through the START command.

The SPOOL Command

- ❖ The SPOOL command routes the output from a SQL*Plus program to the specified filename.
- ❖ The SPOOL command shown below routes output to a file named **report9-2.lst**.
- ❖ The "lst" filename extension is short for listing; however, you can specify any filename extension that you desire.
- ❖ The SPOOL OFF command terminates writing to the output file.

```
SPOOL report9-2.lst
```

```
SPOOL OFF
```

- ❖ Place the SPOOL command immediately before and after the SELECT command that retrieves data for the report—this keeps you from spooling Oracle output responses to other commands in your program.

Master-Detail Reports

- ❖ A Master-Detail report is a form of control break report because the report presents information that is "grouped."
- ❖ The report typically displays data rows from more than one table, such as the *department* and *project* tables of our database.
 - ❖ Each department controls numerous projects, and a project belongs to a single department.
 - ❖ The *department* table contains "master" rows because the *department* table is on the "one" side of the one-to-many relationship.
 - ❖ The associated *project* table rows provide the "detail" information.

SQL Example 9.3 – Part 1

- ❖ A sample Master-Detail program

```
/* SQL Example 9.3 */  
-- Program: ch9-3.sql  
-- Programmer: apowell; Date: today's date  
-- Description: A sample Master-Detail report  
TTITLE CENTER 'Department Number:' DepartmentNumberVar SKIP 2  
BTITLE SKIP 1 CENTER 'Not for external dissemination.'  
REPHEADER 'Project Report #3 - prepared by A. Powell' SKIP 2  
REPFOOTER SKIP 3 '- Last Page of Report -'  
SET LINESIZE 65  
SET PAGESIZE 15  
SET NEWPAGE 1
```

SQL Example 9.3 – Part 2

- ❖ The report produced is on the next slide.

```
-- define department variable
COLUMN DepartmentNumber NEW_VALUE DepartmentNumberVar NOPRINT

-- set column sizes based on alias column names
COLUMN DepartmentName FORMAT A22
COLUMN ProjectTitle FORMAT A25
COLUMN Location FORMAT A15

BREAK ON DepartmentNumber SKIP PAGE

SELECT d.DepartmentNumber, d.DepartmentName,
       p.ProjectTitle, p.Location
FROM Department d JOIN Project p
      ON (d.DepartmentNumber = p.DepartmentNumber)
WHERE d.DepartmentNumber IN (3, 6)
ORDER BY d.DepartmentNumber;
```

Project Report #3

Department Number: 3

Project Report #3 - prepared by A. Powell

DEPARTMENTNAME	PROJECTTITLE	LOCATION
Emergency-Surgical	Remodel ER Suite	Maryville
Emergency-Surgical	Add Crash Cart Equipment	Edwardsville

Not for external dissemination.

More...

Department Number: 6

DEPARTMENTNAME	PROJECTTITLE	LOCATION
Pediatrics-Gynecology	New Pediatric Monitors	St. Louis
Pediatrics-Gynecology	Child Care Center	Alton

- Last Page of Report -

Not for external dissemination.

TTITLE and Variable Data

- ❖ Information in the TTITLE report line identifies the "master column" that controls the page breaks – *DepartmentNumber*.
- ❖ Reference a column value TTITLE by storing the column value to a *program variable*, then specify the program variable name in the TTITLE command.

```
TTITLE CENTER 'Department Number:' DepartmentNumberVar SKIP 2
```

- ❖ A special form of the COLUMN command is used to define a program variable as shown below.

```
-- define department variable
```

```
COLUMN DepartmentNumber NEW_VALUE DepartmentNumberVar NOPRINT
```

- ❖ The **NEW_VALUE** clause defines the variable name. Follow Oracle's naming rules when naming program variables.

BREAK Command for Master-Detail Report

- ❖ The BREAK command used in the program must break on the master column for the report.
- ❖ Here, the master column is the *DepartmentNumber* from the *department* table, although the *DepartmentNumber* from the *project* table could also have been used.

```
BREAK ON DepartmentNumber SKIP PAGE
```

Using Views in Master-Detail Reports

```
/* SQL Example 9.4 */

-- Program: ch9-4.sql

-- Programmer: apowell; today's date

-- Description: A revised Master-Detail program with a view.

TTITLE CENTER 'Department: ' DepartmentVar SKIP 2

BTITLE SKIP 1 CENTER 'Not for external dissemination.'

REPHEADER 'Project Report #4 - prepared by A. Powell' SKIP 2

REPFOOTER SKIP 3 '- Last Page of Report -'

SET LINESIZE 75;

SET PAGESIZE 15;

SET NEWPAGE 1;
```

Using Views in Master-Detail Reports

```
-- Create a view to use in the SELECT command
CREATE OR REPLACE VIEW vwDepartmentProject (Department,
    Project, Location) AS
SELECT d.DepartmentNumber || '-' || d.DepartmentName,
    p.ProjectTitle, p.Location
FROM Department d JOIN Project p
    ON (d.DepartmentNumber = p.DepartmentNumber)
WHERE d.DepartmentNumber IN (3, 6)
ORDER BY d.DepartmentNumber;

COLUMN Department NEW_VALUE DepartmentVar NOPRINT
COLUMN Project FORMAT A30
COLUMN Location FORMAT A20

BREAK ON Department SKIP PAGE;

SELECT Department, Project, location
FROM vwDepartmentProject;
```

Using Views in Master-Detail Reports

- ❖ The revised COLUMN command uses a NEW_VALUE clause to store the value of the *Department* column of the view to a variable named *DepartmentVar*.
- ❖ This variable is used in the TTITLE command to display the department name at the top of each page.
- ❖ The BREAK command breaks on the *Department* column as the master column.
- ❖ The SELECT statement is greatly simplified because the program selects from the view.
- ❖ If the view already exists, then the code to create the view can be deleted from program ch9-4.sql.

Project Report #4 – Uses View

Department: 3-Emergency-Surgical

Project Report #4 – prepared by A. Powell

PROJECT	LOCATION

Remodel ER Suite	Maryville
Add Crash Cart Equipment	Edwardsville

Not for external dissemination.

Department: 6-Pediatrics-Gynecology

PROJECT	LOCATION

New Pediatric Monitors	St. Louis
Child Care Center	Alton

- Last Page of Report -

Not for external dissemination.

Primary Keys Foreign Keys

Dr. Anne Powell
CMIS563

A Primary Key

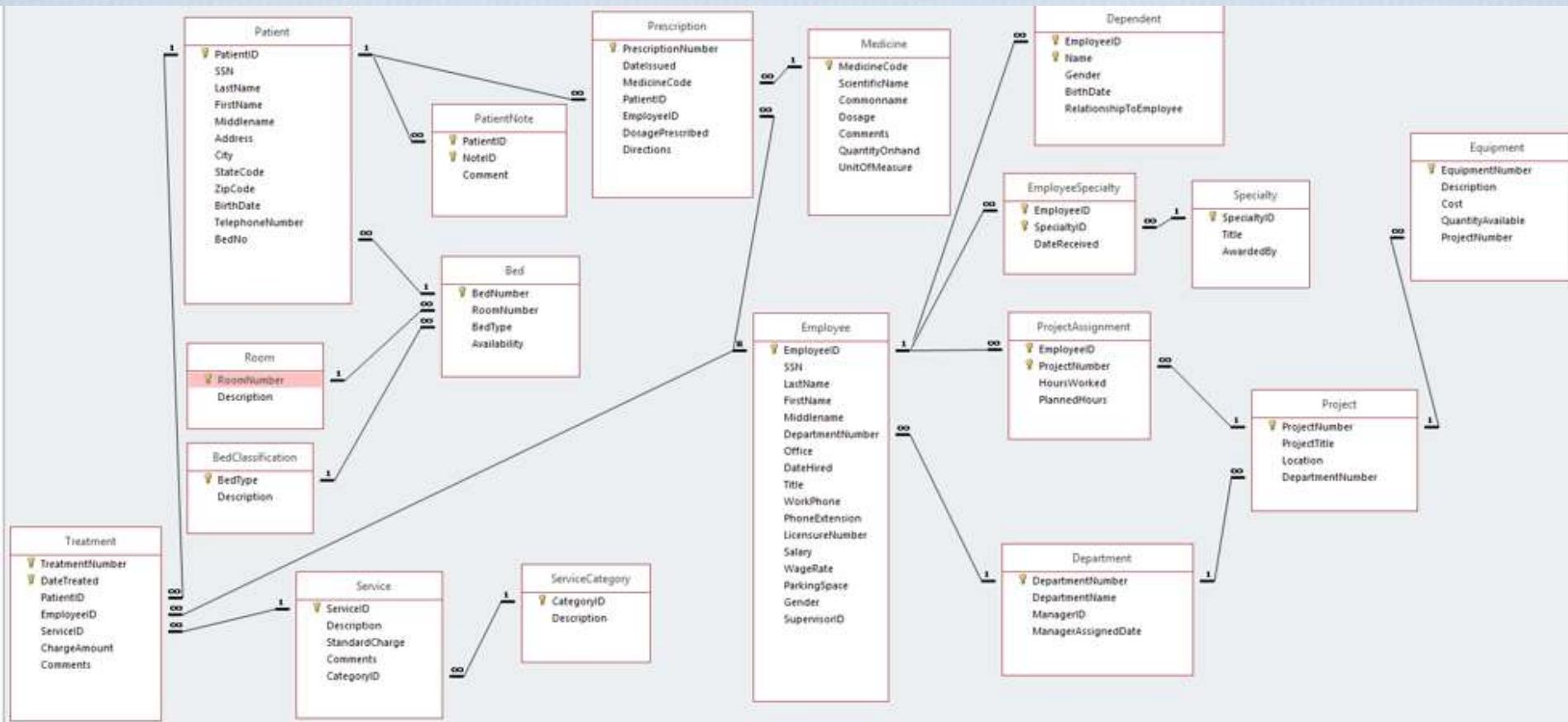
- ❖ Attribute(s) that uniquely identify an entity (relation) instance.
- ❖ If you know the value of the primary key, you will be able to uniquely identify a single row.
- ❖ You must be able to answer all three of these questions with these answers when selecting the primary key
 - ❖ Is the value unique? – Yes
 - ❖ Will the value ever need to be repeated? – No
 - ❖ Will the value ever be null? - No

A Composite Key

- ❖ A **composite key** is a key that contains two or more attributes.
- ❖ For a key to be unique, it must often become a composite key.

Identifying the primary key

- ❖ Select the attribute
 - ❖ Underline it
-
- ❖ EMPLOYEE (empID, empName, empAddr, empJob)
 - ❖ STUDENT (studentID, studentName, studentMajor)



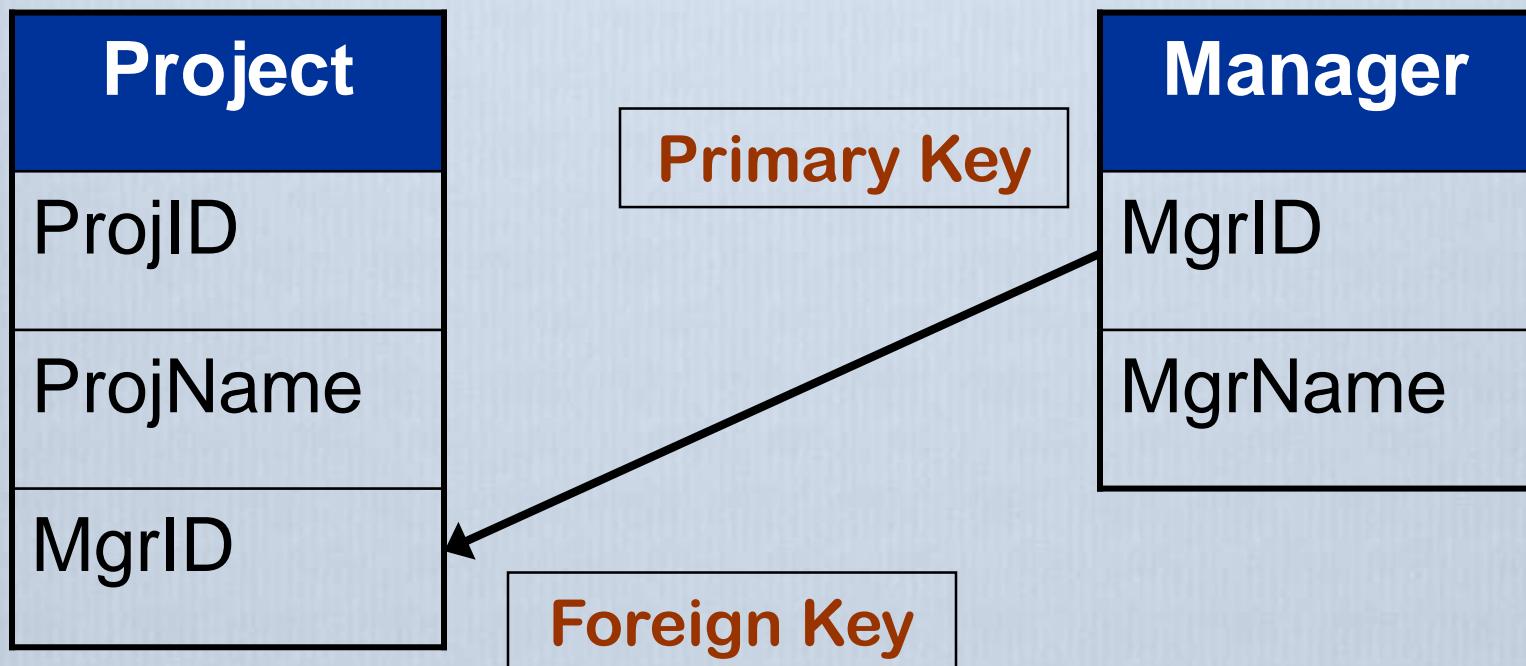
Relationships Between Tables

- ❖ A table may be related to other tables.
- ❖ For example
 - ❖ An Employee works in a Department
 - ❖ A Patient receives a Prescription
 - ❖ A Bed is assigned to a Patient

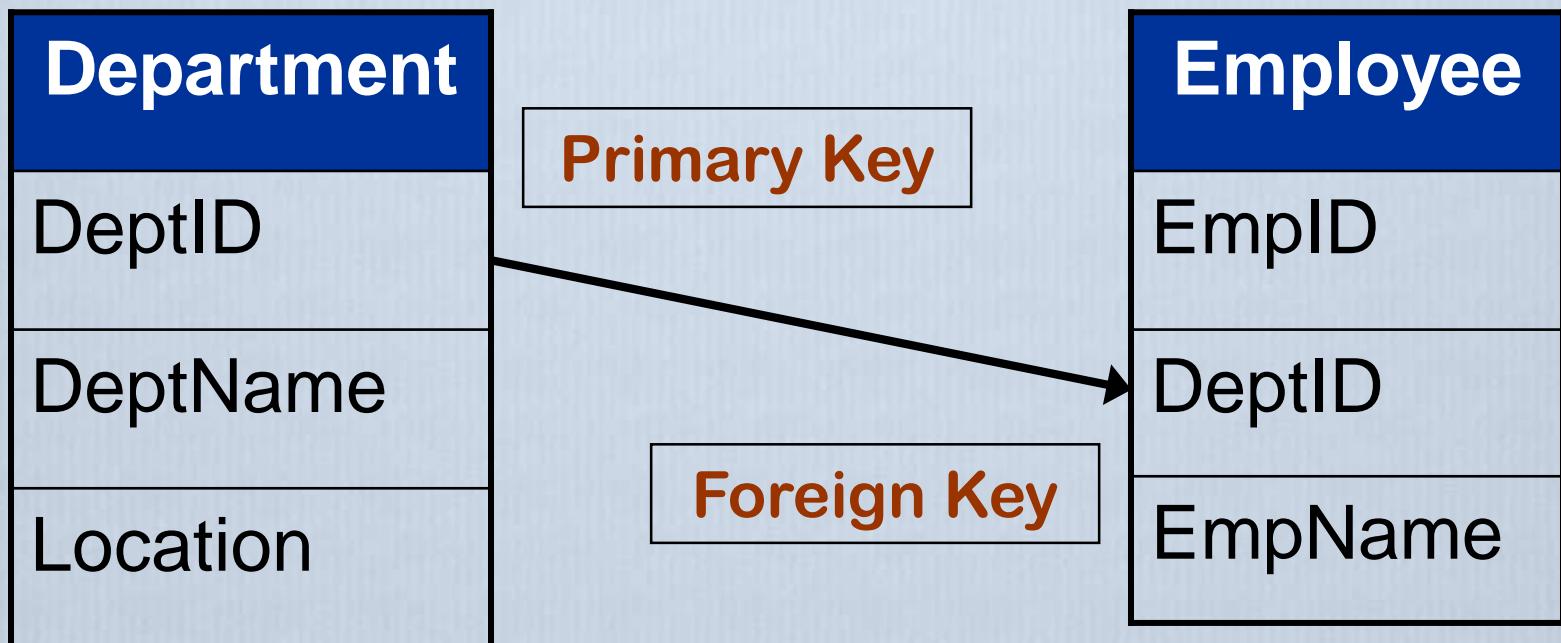
How do we relate tables? – A Foreign Key

- ❖ To preserve relationships, you may need to create a **foreign key**.
- ❖ A foreign key is a primary key from one table placed into another table.
- ❖ The key is called a foreign key in the table that received the key.

Foreign Key Example I



Foreign Key Example II

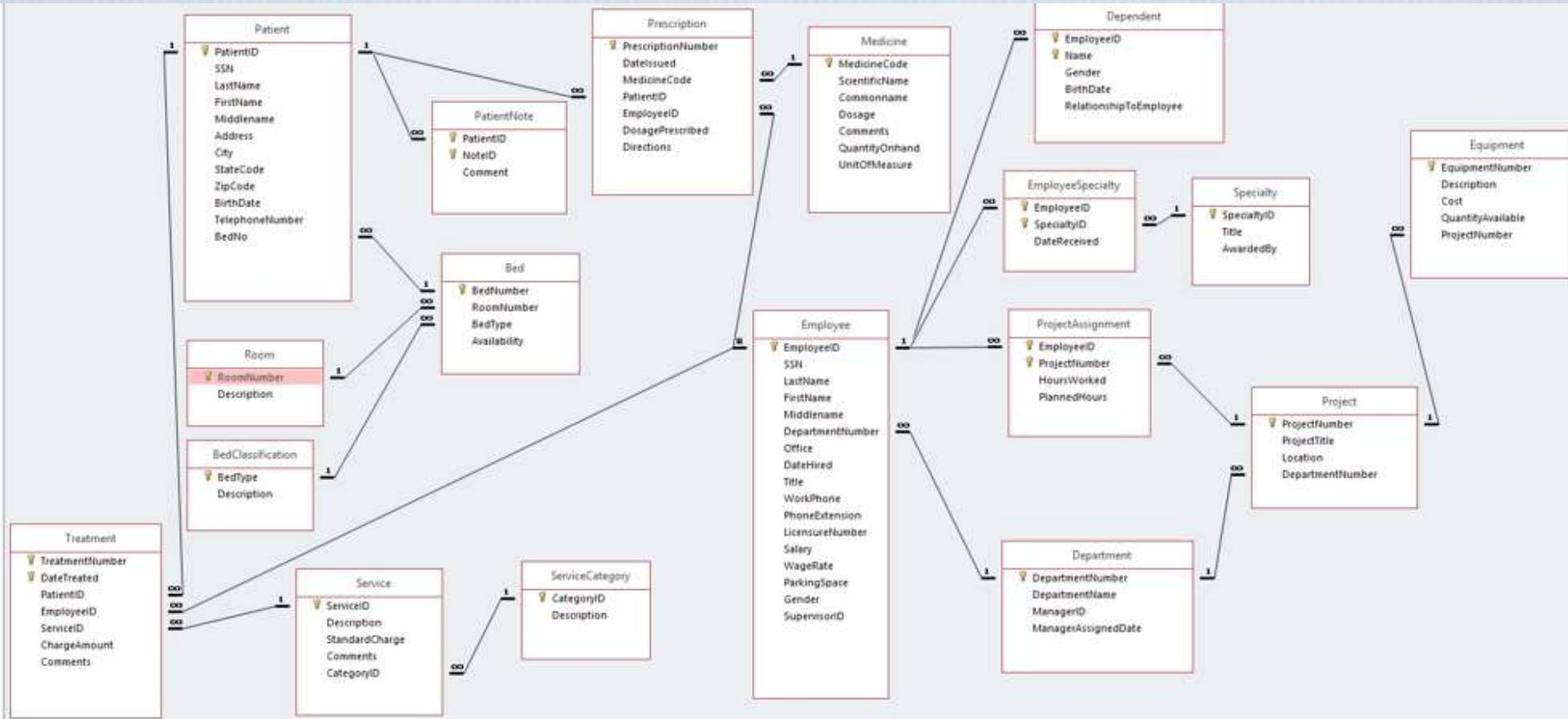


Some rules...

- ❖ Each relationship between two entities has ONE foreign key.
- ❖ How do we know which entity's primary key we use as a foreign key in the other entity?
 - ❖ i.e., PatientID is the primary key in Patient and the foreign key in Prescription. WHY isn't PrescriptionNumber the foreign key in Patient instead?

For each relationship:

- ❖ In a 1:m relationship, the primary key of the ONE entity becomes the foreign key in the MANY entity (always!)
- ❖ In a 1:1 relationship it doesn't matter which primary key becomes a foreign key – but only pick one!
- ❖ In a m:m relationship – special circumstance that we will cover in the Associative entity video!
- ❖ I'll repeat because it is IMPORTANT: In a 1:m relationship, the primary key of the ONE entity becomes the foreign key in the MANY entity!



Referential Integrity

- ❖ Referential integrity states that every value of a foreign key must match a value of an existing primary key.
- ❖ Example (see previous slide):
 - ❖ If EmpID = 4 in EMPLOYEE has a DeptID = 7 (a foreign key), a Department with DeptID = 7 must exist in DEPARTMENT.
 - ❖ The primary key value must exist before the foreign key value is entered.

Single Table Queries

Week 2 Chapter 4 Video 4
DISTINCT – WHERE
Dr. Anne Powell

Agenda

- ❖ 1. Basic SELECT commands
- ❖ 2. Column Formatting
- ❖ 3. Common Errors
- ❖ **4. WHERE - DISTINCT commands**
- ❖ 5. ORDER BY command
- ❖ 6. In-class practice

SQL – WHERE Clause

- ❖ This SELECT statement only selects specific fields (SSN and FirstName).
- ❖ The WHERE clause specifies a single SSN.

```
/* SQL Example WHERE clause */  
  
SELECT SSN, FirstName  
  
FROM Employee  
  
WHERE SSN = '215243964';  
  
SSN      FIRSTNAME  
----- -----  
215243964    Douglas
```

WHERE - Character Data

- ❖ Comparison operators can be used with columns containing character data.
- ❖ Literal character strings and dates must be enclosed with *single quotation ('')* marks.

```
/* SQL Example */  
SELECT EmployeeID, LastName, FirstName  
FROM Employee  
WHERE Gender = 'M';  
EMPLOYEEID LASTNAME          FIRSTNAME  
-----  
67555      Simmons           Lester  
33344      Adams             Adam  
33358      Thornton         Billy  
more rows will be displayed...
```

WHERE – Numeric Example

```
COLUMN Salary FORMAT $99,999.99  
COLUMN LastName FORMAT A15;  
COLUMN FirstName FORMAT A15;  
SELECT EmployeeID, LastName, FirstName, Salary  
FROM Employee  
WHERE Salary >= 25000;
```

EMPLOYEEID	LASTNAME	FIRSTNAME	SALARY
88303	Jones	Quincey	\$30,550.00
88404	Barlow	William	\$27,500.00
88505	Smith	Susan	\$32,500.00

3 rows selected

- ❖ NOTE: Do not use the \$ symbol, comma, or single quotes when specifying a numeric, currency condition.

Comparison Operators

Operator	Meaning
=	equal to
<	less than
>	greater than
>=	greater than or equal to
<=	less than or equal to
!=	not equal to
<>	not equal to
!>	not greater than
!<	not less than

DISTINCT Clause

The **DISTINCT** clause eliminates duplicate rows in a result table.
DISTINCT must be specified prior to any column names.

```
/* SQL Example 4.21 */
SELECT Salary
FROM Employee
WHERE Salary < 10000;
```

SALARY

\$5,500.00
\$4,550.00
\$6,500.00
\$4,800.00
\$2,200.00
\$2,200.00
\$4,895.00

7 rows selected.

```
/* SQL Example 4.22 */
SELECT DISTINCT Salary
FROM Employee
WHERE Salary < 10000;
```

SALARY

\$2,200.00
\$4,550.00
\$4,800.00
\$4,895.00
\$5,500.00
\$6,500.00

6 rows selected.

a

NOTE: If you format a column once in a query, any subsequent query (UNTIL you log out) will retain that formatting. In this example, a prior query had formatted Salary to COLUMN Salary FORMAT \$99,999.99;

HAVING

Week 3 Chapter 6 Video 4
HAVING command
Dr. Anne Powell

Agenda

1. Aggregate Function Rules – COUNT
2. Aggregate Function Rules – the rest
3. GROUP BY command
- 4. HAVING command**
5. Examples

Learning Objectives

- ❖ Write queries with aggregate functions: COUNT, AVG, SUM, MAX, and MIN.
- ❖ Use the GROUP BY clause to answer complex managerial questions.
- ❖ Nest aggregate functions.
- ❖ Use the GROUP BY clause with NULL values.
- ❖ Use the GROUP BY clause with the WHERE and ORDER BY clauses.
- ❖ **Use the HAVING clause to filter out rows from a result table.**

HAVING

- ❖ Is like a WHERE statement on the output of the GROUP BY
- ❖ Is often interchangeable with Where UNLESS you are using an aggregate (sum, avg, count, etc). (WHERE can't do that so you have to use HAVING)
- ❖ The HAVING clause is to aggregate functions (sum, avg, count, etc) what the WHERE clause is for column names and expressions.

HAVING

- ❖ The HAVING and WHERE clauses do the same thing - filter rows from inclusion in a result table based on a condition.
- ❖ A WHERE clause is used to filter rows **BEFORE** the GROUPING action.
- ❖ A HAVING clause filters rows **AFTER** the GROUPING action.

GROUP BY with HAVING

We want a report with the average of all salaries of employees within a department – BUT, only for departments where the average salary is over \$16,000.

```
SELECT DepartmentNumber "Department",
       AVG(Salary) "Average Salary"
  FROM Employee
 GROUP BY DepartmentNumber
 HAVING AVG(Salary) > 16000;
```

Department	Average Salary

3	\$26,119
5	\$22,325
9	\$17,525

GROUP BY with WHERE

We want a report with the average of all salaries of employees within a department – BUT, only for those employees who have a salary greater than \$16,000.

```
SELECT DepartmentNumber "Department", AVG(Salary) "Average Salary"  
FROM Employee  
WHERE Salary > 16000  
GROUP BY DepartmentNumber;
```

Department Average Salary

1	16250
2	17675
3	26119
5	22325
6	23000
8	19760
9	17525

HAVING with WHERE

```
SELECT DepartmentNumber "Department",
       AVG(Salary) "Average Salary"
  FROM Employee
 WHERE DepartmentNumber <> 3
 GROUP BY DepartmentNumber
 HAVING AVG(Salary) > 16000;
```

Department	Average Salary
5	\$22,325
9	\$17,525

NOTE: Acts on WHERE criteria first, then does GROUP BY, then finds departments with avg salary above \$16,000.

The logical process of prior slide

- ❖ Conceptually, SQL performs the following steps in the query on slide 44:
 1. The WHERE clause filters rows that do not meet the condition *DepartmentNumber* $<>$ 3.
 2. The GROUP BY clause collects the surviving rows into one or more groups for each unique *DepartmentNumber*.
 3. The aggregate function calculates the average salary for each *DepartmentNumber* grouping.
 4. The HAVING clause filters out the rows from the result table that do not meet the condition: average salary greater than \$16,000.

Order of operations for SQL

- ❖ 1. FROM – pulls tables needed to run queries
- ❖ 2. WHERE – ids rows needed for queries
- ❖ 3. GROUP BY
- ❖ 4. HAVING
- ❖ 5. SELECT
- ❖ 6. ORDER BY

GROUP BY and HAVING rules

- ❖ Columns listed in a SELECT clause must also be listed in the GROUP BY expression or they must be arguments of aggregate functions.
- ❖ A GROUP BY expression can only contain column names that are in the SELECT clause listing.
- ❖ Columns in a HAVING expression must be either:
 - ❖ Single-valued—arguments of an aggregate function, for instance, or
 - ❖ Listed in the SELECT clause listing or GROUP BY clause.

What is this doing?

```
SELECT DepartmentNumber "Department",
       COUNT(*) "Employee Count"
  FROM Employee
 GROUP BY DepartmentNumber
 HAVING COUNT(*) >= 3;
Department Employee Count
-----
```

2	3
3	5
6	4
8	5

Do you understand this query?

```
COLUMN "Top Salary" FORMAT $999,999;
COLUMN "Low Salary" FORMAT $999,999;
SELECT DepartmentNumber "Department", COUNT(*)  
    "Employee Count",  
        MAX(Salary) "Top Salary", MIN(Salary) "Low Salary"  
FROM Employee  
GROUP BY DepartmentNumber  
HAVING COUNT(*) >= 3;
```

Department	Employee Count	Top Salary	Low Salary
2	3	\$17,850	\$4,550
3	5	\$32,500	\$16,500
6	4	\$23,000	\$2,200
8	5	\$22,000	\$5,500

Error in Using the HAVING Clause

- ❖ The HAVING clause is a conditional option that is directly related to the GROUP BY clause option because a HAVING clause eliminates rows from a result table based on the result of a GROUP BY clause.
- ❖ In Oracle, A HAVING clause will not work without a GROUP BY clause.

```
/* SQL Example 6.29 */  
SELECT DepartmentNumber, AVG(Salary)  
FROM Employee  
HAVING AVG(Salary) > 33000;
```

ERROR at line 1: ORA-00937: not a single-group group function

Questions?



MULTI-TABLE QUERIES

Week 4 Chapter 7 Video 4
VIEW

Dr. Anne Powell

VIEWs: Learning Objectives

- ❖ Create a single table and join table views—include functions; drop a view.
- ❖ Insert, update, and delete table rows using a view.
- ❖ Create a view with errors.
- ❖ Create a materialized view; drop a materialized view.

VIEWS

- ❖ A database view is a *logical* or *virtual table* based on a query.
- ❖ It is useful to think of a *view* as a stored query.
- ❖ Views are created through use of a CREATE VIEW command that incorporates use of the SELECT statement.
- ❖ Views are queried just like tables.

VIEWS

```
CREATE VIEW vwEmpParking  
(parking_space, last_name,  
first_name, ssn) AS
```

```
SELECT parkingSpace, lastName,  
firstName, SSN  
  
FROM employee  
  
ORDER BY parkingSpace;
```

View Created.

VIEWS

```
SELECT *  
FROM vwEmpParking;
```

PARKING_SPACE	LAST_NAME	FIRST_NAME	SSN
1	Bordoloi	Bijoy	999666666
3	Joyner	Suzanne	999555555
32	Zhu	Waiman	999444444

more rows are displayed...

- ❖ Notice that the only columns in the query are those defined as part of the view.

VIEWS

- ❖ Additionally, we have renamed the columns in the view so that they are slightly different than the column names in the underlying employee table.
- ❖ Further, the rows are sorted by *parking_space* column even though there is no ORDER BY in the SELECT command used to access the view.

CREATING A VIEW

- ❖ `CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW <view name> [(column alias name....)] AS <query> [WITH [CHECK OPTION] [READ ONLY] [CONSTRAINT]];`
- ❖ The OR REPLACE option is used to create a view that already exists. This option is useful for modifying an existing view without having to drop or grant the privileges that system users have acquired with respect to the view .
- ❖ If you attempt to create a view that already exists without using the OR REPLACE option, Oracle will return the ORA-00955: *name is already used by an existing object* error message and the CREATE VIEW command will fail.

DROPPING VIEW

- A DBA or view owner can drop a view with the DROP VIEW command. The following command drops a view named *dept_view*.

```
DROP VIEW vwEmpParking;
```

View dropped.

Example (no renaming of fields)

```
CREATE VIEW vwEmpView7 AS  
SELECT ssn, firstName, lastName  
FROM employee  
WHERE departmentNumber = 7;
```

View created.

- A simple query of the *empview7* shows the following data.

SELECT *	FROM vwEmpView7;	SSN	FIRSTNAME	LASTNAME
		-----	-----	-----
			790543232	Betty
				Boudreaux

Example

- It is also possible to create a view that has exactly the same structure as an existing database table.
- The view named *dept_view* shown next has exactly the same structure as *department* table.

```
CREATE VIEW dept_view AS  
SELECT *  
FROM department;
```

View created.

VIEW STABILITY

- A view does not actually store any data. The data needed to support queries of a view are retrieved from the underlying database tables and displayed to a result table whenever a view is queried. The result table is only stored temporarily.
- If a table that underlies a view is dropped, then the view is no longer valid. Attempting to query an invalid view will produce an ORA-04063: view "VIEW_NAME" has errors error message.

INSERTING , UPDATING, AND DELETING TABLE ROWS THROUGH VIEWS

- You can insert a row if the view in use is one that is updateable (not read-only) (see extra info slides about read-only).
- A view is updateable if the INSERT command does not violate any constraints on the underlying tables.
- This rule concerning constraint violations also applies to UPDATE and DELETE commands.

Inserting additional rows: Example

```
CREATE OR REPLACE VIEW dept_view AS  
SELECT departmentNumber, departmentName  
FROM department;
```

```
INSERT INTO dept_view VALUES (18, 'Department 18');  
INSERT INTO dept_view VALUES (19, 'Department 20');
```

```
SELECT *  
FROM dept_view;  
DEPAR DEPARTMENTNAME
```

```
7 Production  
3 Admin and Records  
1 Headquarters  
18 Department 18  
19 Department 20
```

Example

```
UPDATE dept_view SET departmentName = 'Department 1'  
WHERE departmentNumber = 1;
```

1 row updated.

```
SELECT departmentNumber, departmentName  
FROM dept_view  
WHERE departmentNumber < 3;  
  
DEPARTMENTNUMBER DEPARTMENTNAME
```

-
-
- 1 Department 1
 - 2 Radiology
 - 3 Emergency-Surgical

A Summary of VIEW Facts

- A view can display data from more than one table.
- Views can be used to update the underlying tables. Views can also be limited to read-only access.
- Views can change the appearance of data. For example, a view can be used to rename columns from tables without affecting the base table.
- A view that has columns from more than one table cannot be modified by an INSERT, DELETE, or UPDATE command if a grouping function, GROUP BY clause is part of the view definition.

A Summary of VIEW Facts

- A row cannot be inserted in a view in which the base table has a column with the NOT NULL or other constraint that cannot be satisfied by the new row data.

Extra Info: CREATING A VIEW

- ❖ The FORCE option allows a view to be created even if a base table that the view references does not already exist.
- ❖ This option is used to create a view prior to the actual creation of the base tables and accompanying data. Before such a view can be queried, the base tables must be created and data must be loaded into the tables. This option can also be used if a system user does not currently have the privilege to create a view.
- ❖ The NOFORCE option is the opposite of FORCE and allows a system user to create a view if they have the required permissions to create a view, and if the tables from which the view is created already exist. This is the default option.

Extra Info: CREATING A VIEW

- ❖ The WITH READ ONLY option allows creation of a view that is read-only. You cannot use the DELETE, INSERT, or UPDATE commands to modify data for the view.
- ❖ The WITH CHECK OPTION clause allows rows that can be selected through the view to be updated. It also enables the specification of constraints on values.
- ❖ The CONSTRAINT clause is used in conjunction with the WITH CHECK OPTION clause to enable a database administrator to assign a unique name to the CHECK OPTION. If the DBA omits the CONSTRAINT clause, Oracle will automatically assign the constraint a system-generated name that will not be very meaningful.

Extra Info: Example

- We can recreate the view by using the OR REPLACE clause to create a view that is *read-only* by specifying a WITH READ ONLY clause.
- The new version of *dept_view* will restrict data manipulation language operations on the view to the use of the SELECT command.

```
CREATE OR REPLACE VIEW dept_view AS  
SELECT *  
FROM department WITH READ ONLY CONSTRAINT  
vw_dept_view_read_only;
```

View created.

Extra Info: FUNCTIONS AND VIEWS – A JOIN VIEW

- In addition to specifying columns from existing tables, you can use single row functions consisting of number, character, date, and group functions as well as expressions to create additional columns in views.
- This can be extremely useful because the system user will have access to data without having to understand how to use the underlying functions.

Extra Info: Example

```
CREATE OR REPLACE VIEW dept_salary  
    (name, min_salary, max_salary, avg_salary) AS  
SELECT d.dpt_name, MIN(e.emp_salary),  
      MAX(e.emp_salary), AVG(e.emp_salary)  
FROM employee e, department d  
WHERE e.emp_dpt_number=d.dpt_no  
GROUP BY d.dpt_name;
```

View created.

*SELECT **

FROM dept_salary;

<i>NAME</i>	<i>MIN_SALARY</i>	<i>MAX_SALARY</i>	<i>AVG_SALARY</i>
-------------	-------------------	-------------------	-------------------

<i>Admin and Records</i>	25000	43000	31000
<i>Headquarters</i>	55000	55000	55000
<i>Production</i>	25000	43000	34000

Subqueries

CMIS 563

Week 5 Chapter 8 Video 4
Correlated subqueries / EXISTS
Dr. Anne Powell

Agenda

- ❖ 1. Explanation of subqueries / Subquery Rules
- ❖ 2. Using IN, ORDER BY, and comparison operators
- ❖ 3. Using ANY and ALL keywords / Nest subqueries at multiple levels
- ❖ **4. Correlated subqueries / EXISTS operator**
- ❖ 5. In-class examples

Correlated Subqueries

- ❖ A *correlated subquery* is one where the inner query depends on values provided by the outer query.
- ❖ This means the inner query is executed repeatedly, once for each row that might be selected by the outer query.

Example 7.26

- ❖ List the employee with the largest salary in each department.
- ❖ The inner query compares the employee department number column (*DepartmentNumber*) of the *employee* table with alias *e2* to the same column for the alias table name *e1*. The value of *e1.DepartmentNumber* is treated like a variable—it changes as the Oracle server examines each row of the employee table.
- ❖ The subquery results are correlated with each individual row of the main query—thus, the term *correlated subquery*.

```
/* SQL Example 8.26 */  
SELECT LastName "Last Name", FirstName "First Name",  
       DepartmentNumber "Dept", Salary "Salary"  
FROM Employee e1  
WHERE Salary =  
      (SELECT MAX(Salary)  
       FROM Employee e2  
      WHERE e2.DepartmentNumber = e1.DepartmentNumber);
```

CORRELATED SUBQUERIES

❖ Output

Last Name	First Name	Dept	Salary
Simmons	Lester	8	\$22,000
Bock	Douglas	1	\$16,250
Bordoloi	Bijoy	2	\$17,850
Smith	Susan	3	\$32,500
Klepper	Robert	4	\$15,055
Quattromani	Toni	5	\$22,325
Becker	Roberta	6	\$23,000
Boudreaux	Betty	7	\$4,895
Schultheis	Robert	9	\$17,525

9 rows selected.

Subqueries and the EXISTS operator

- ❖ When a subquery uses the **EXISTS** operator, the subquery functions as an *existence test*.
- ❖ The WHERE clause of the outer query tests for the existence of rows returned by the inner query.
- ❖ The subquery does not actually produce any data; rather, it returns a value of TRUE or FALSE.
- ❖ The general format of a subquery WHERE clause with an EXISTS operator is shown here.
- ❖ Note that the NOT operator can also be used to negate the result of the EXISTS operator.

WHERE [NOT] EXISTS (subquery)

Example 8.27

- ❖ The subquery executes for each *employee* row by searching the *dependent* table for one or more rows that meet the subquery WHERE clause. If at least one row is found, the subquery returns a TRUE value – when the outer query receives a TRUE value, the employee row is included in the result table.

```
/* SQL Example 8.27 */  
SELECT LastName "Last Name", FirstName "First Name"  
FROM Employee e  
WHERE EXISTS  
(SELECT *  
FROM Dependent d  
WHERE d.EmployeeID = e.EmployeeID);
```

Last Name	First Name
Bock	Douglas
Bordoloi	Bijoy
Boudreaux	Beverly
Simmons	Lester

Subqueries and the EXISTS Operator

Notice that subqueries using the **EXISTS** operator are a bit different from other subqueries in the following ways:

- ❖ A subquery that uses an EXISTS operator always is a correlated subquery.
- ❖ The keyword EXISTS is not preceded by a column name, constant, or other expression.
- ❖ The parameter in the SELECT clause of a subquery that uses an EXISTS operator almost always consists of an asterisk (*) – there is no reason to list column names since you are simply testing for the existence of rows that meet the conditions specified in the subquery.
- ❖ The subquery evaluates to TRUE or FALSE rather than returning any data.

Subqueries and the EXISTS Operator

- ❖ The EXISTS operator is very important, because there is often no alternative to its use.
- ❖ All queries that use the IN operator or a modified comparison operator ($=$, $<$, $>$, etc. modified by ANY or ALL) can be expressed with the EXISTS operator.
- ❖ However, some queries formulated with EXISTS cannot be expressed in any other way!
- ❖ Why not always use EXISTS instead of IN or a comparison operator?
- ❖ The next slide explains why.

Subqueries and the EXISTS Operator

- ❖ This query uses the = ANY operator. The subquery produces an intermediate result table, then the outer query checks each employee table row against the intermediate result table.
- ❖ A table with 5, 000 rows requires 5,000 iterations.

```
/* SQL Example 8.28 */
SELECT LastName "Last Name"
FROM Employee
WHERE EmployeeID = ANY
    (SELECT EmployeeID
     FROM Dependent);
Last Name
-----
Bock
Bordoloi
Boudreaux
Simmons
```

Subqueries and the EXISTS Operator

- ❖ This query uses the **EXISTS** operator and is a correlated subquery; therefore, the subquery processes once for every row processed by the outer query.
- ❖ A table with 5,000 rows will process the subquery 5,000 times for each outer query row for a total of 25,000,000 iterations! Obviously, the **= ANY** operator query is more efficient.

```
/* SQL Example 8.29 */
SELECT LastName "Last Name"
FROM Employee e
WHERE EXISTS
    (SELECT *
     FROM Dependent d
     WHERE d.EmployeeID = e.EmployeeID);
```

Last Name
Bock
Bordoloi
Boudreaux
Simmons

Subqueries and the NOT EXISTS Operator

- ❖ The NOT EXISTS operator is the mirror-image of the EXISTS operator.
- ❖ A query that uses NOT EXISTS in the WHERE clause is satisfied if the subquery returns no rows.

An ORDER BY exception

- ❖ ROWNUM is a pseudocolumn which returns a number indicating the order in which Oracle selects the rows from the table. The first row selected has a ROWNUM of 1; the second has 2 and so on. If you need to restrict the number of returned rows from your query, you can use ROWNUM. If you are doing this, you will need to use an ORDER BY in the inner query so that the returned results are returned in the order desired.
- ❖ In this case, your SELECT statement is on the FROM clause and not the WHERE clause.

For example

- ❖ You want a list of employee ID and the date hired of the 3 longest tenured employees.

```
/* Example 8.31 */  
SELECT EmployeeID, dateHired  
FROM  
  (SELECT employeeID, dateHired  
   FROM employee  
   ORDER BY dateHired)  
WHERE ROWNUM <=3;
```

EMPLO DATEHIRED

66427	14-DEC-79
88101	14-DEC-82
88202	14-DEC-82

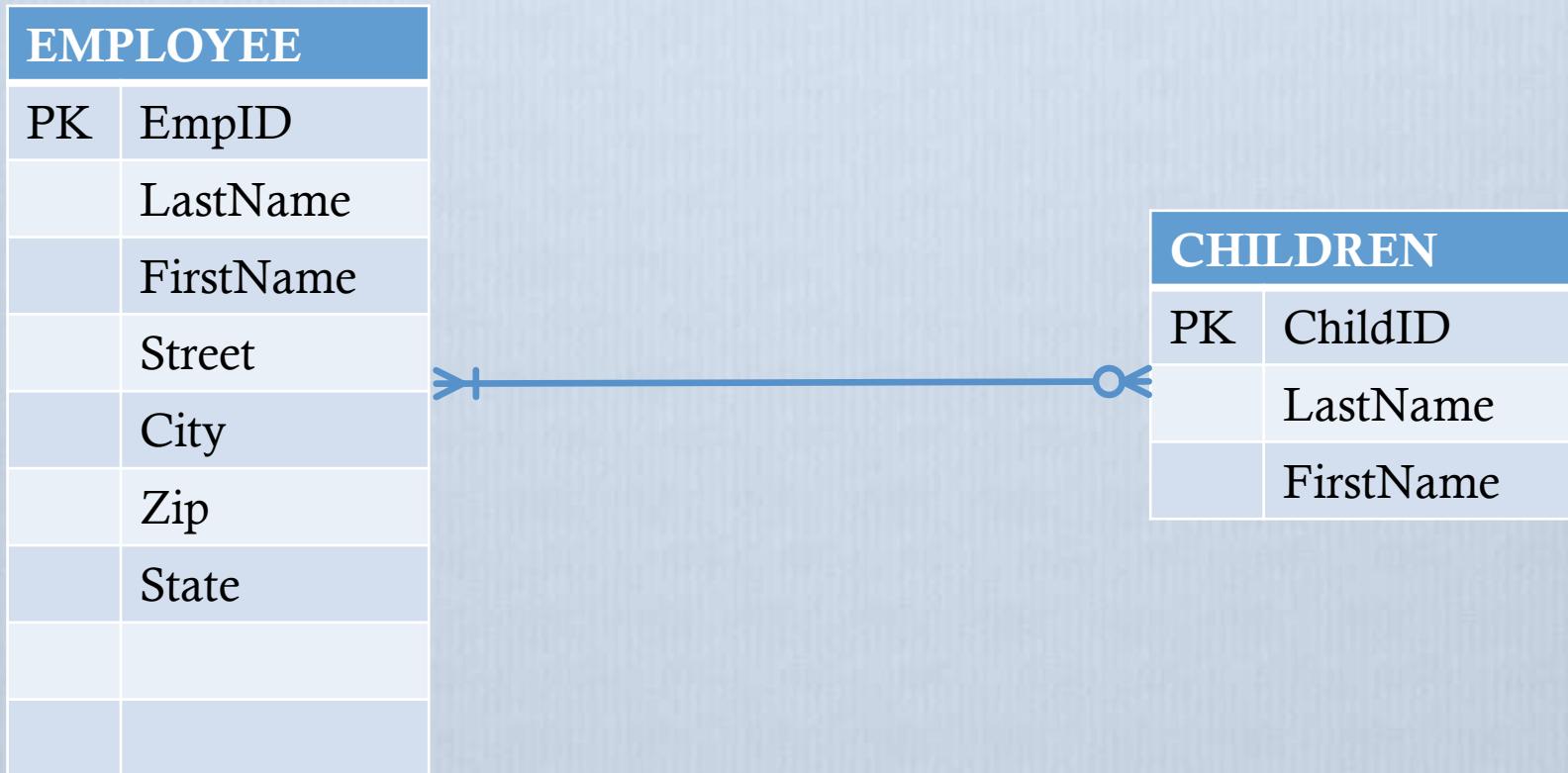
Summary

- ❖ A subquery is simply a query inside another query, generally the object of a WHERE clause, and must produce data values where the data is JOIN COMPATIBLE with the expression of the WHERE clause of the outer query.
- ❖ Subqueries can be nested inside of other subqueries.
- ❖ You also used operators such as IN and NOT IN for queries and learned to use ALL and ANY to modify comparison operators.
- ❖ Aggregate functions are commonly used in subqueries to return a *scalar* result table.
- ❖ The *correlated subquery* is one where the inner query depends on values provided by the outer query.
- ❖ You also learned to use the EXISTS operator for subqueries that test the existence of rows that satisfy a specified criteria. You also learned where to place the ORDER BY clause for an outer query.

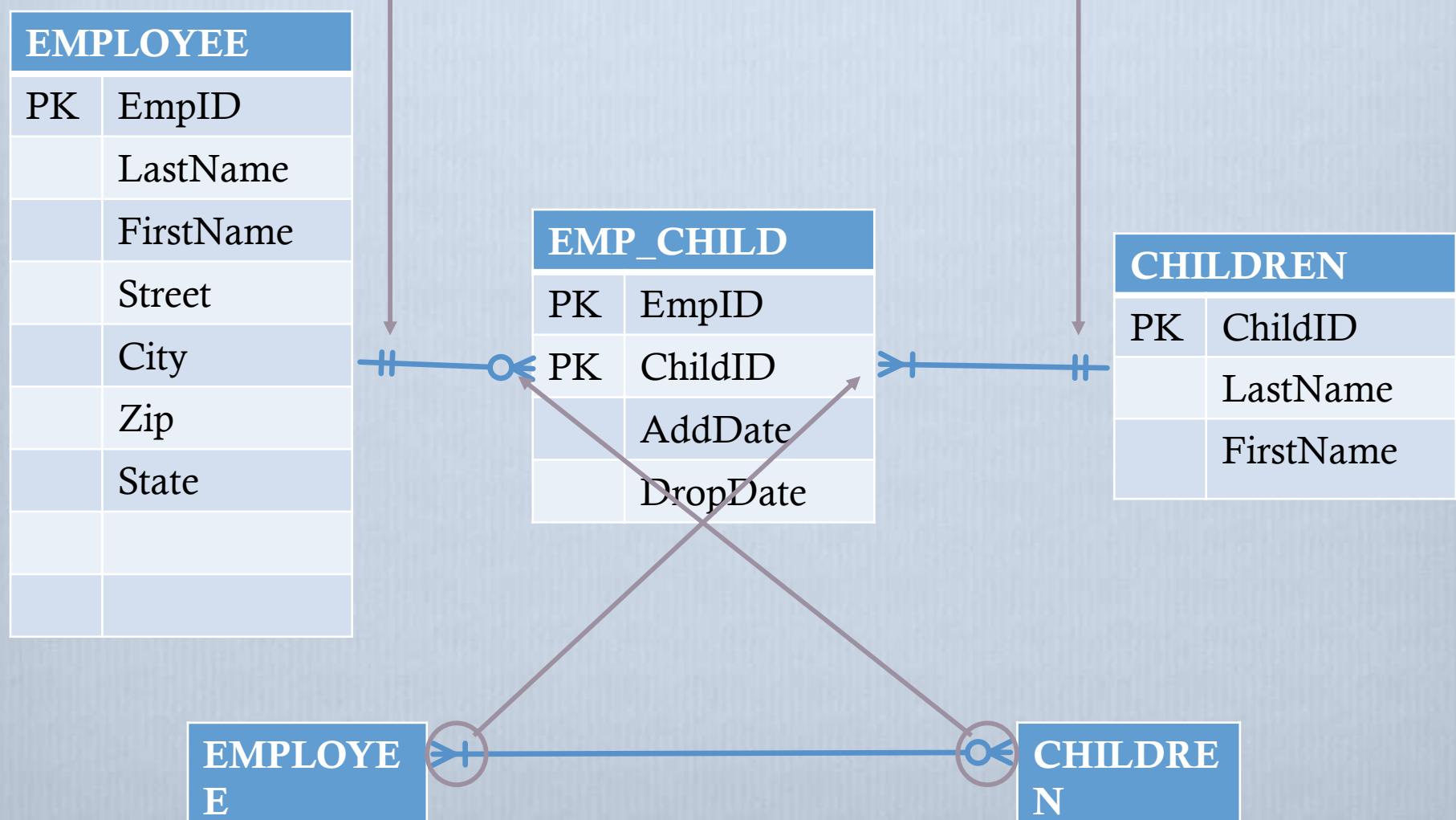
Associative Entities

Dr. Anne Powell
CMIS563

Associative entities



Associative entities



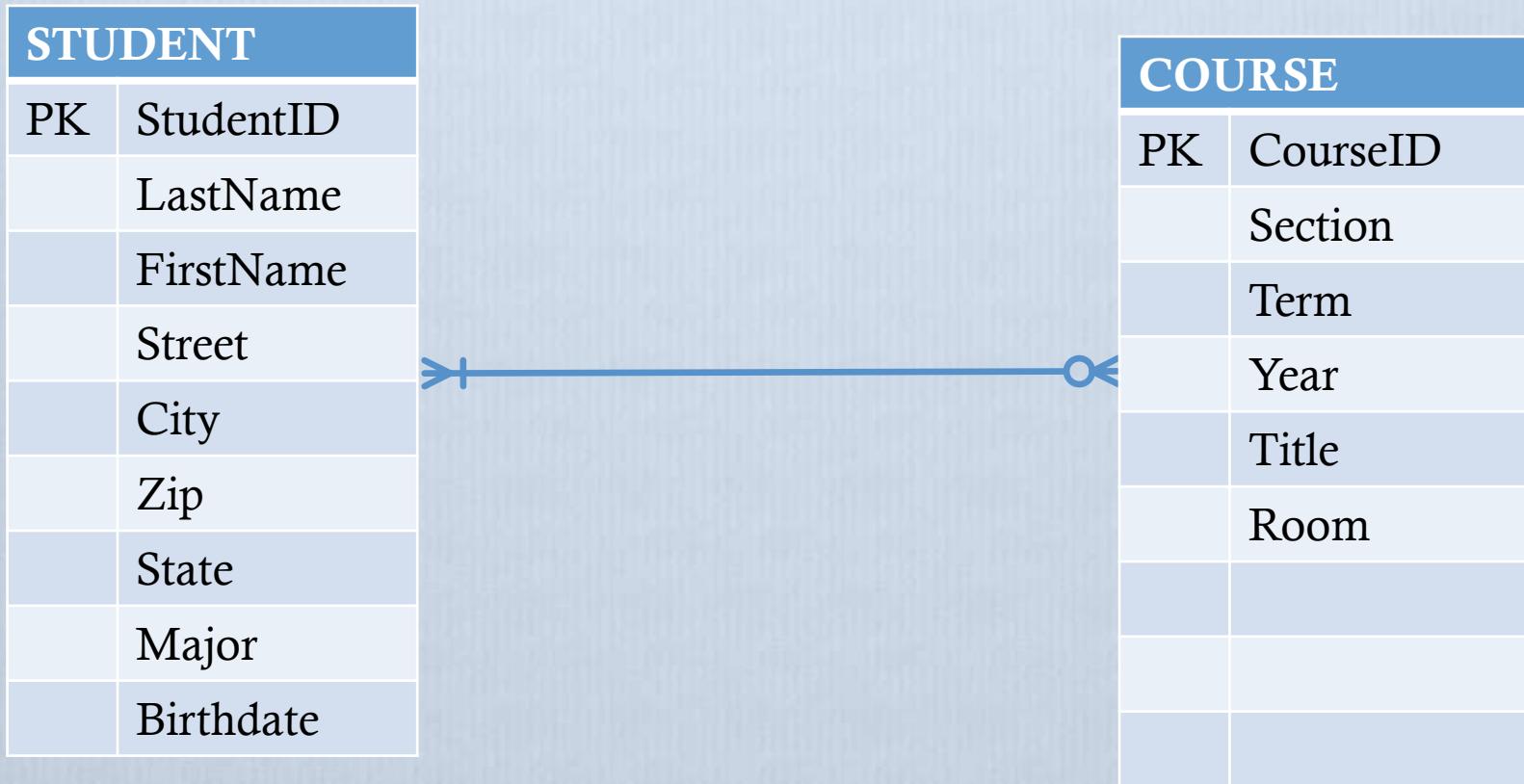
Associative entities

EmpID	LastName	FirstName	Street	City	Zip	State
800564	Jones	Sarah	45 McMahon	Edwardsville	62026	IL
800493	Smith	Diane	4190 Spruce Lane	Edwardsville	62026	IL
811120	Franks	Adam	987 Euclid Ave	Collinsville	62053	IL
849083	Laughlin	Chris	2 Sunnyvale Drive	Granite City	62040	IL
872389	Laughlin	Abigail	2 Sunnyvale Drive	Granite City	62041	IL

EmpID	ChildID	AddDate	DropDate
849083	932038	9/24/2005	5/2/2010
872389	932038	5/2/2010	
872389	953245	6/15/2010	
811120	945021	5/11/2011	
811120	942359	8/1/1994	11/30/2020
800493	908432	1/23/2012	

ChildID	FirstName	LastName	Birthdate
953245	Alex	Jordan	12/8/07
908432	Fred	Smith	1/23/12
945021	Keira	Franks	5/11/11
942359	Donovan	Franks	8/1/02
932038	Samuel	Laughlin	9/24/05

Another example



Fixed!

STUDENT	
PK	StudentID
	LastName
	FirstName
	Street
	City
	Zip
	State
	Major
	Birthdate

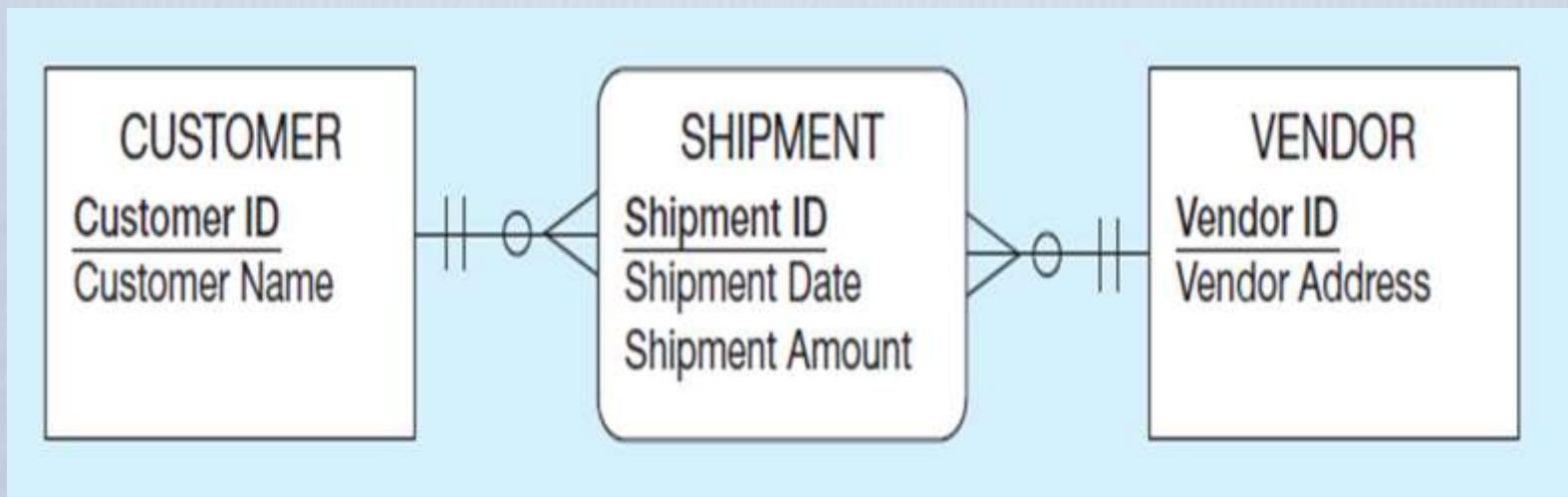
REGISTRATION	
PK	StudentID
PK	CourseID
	DateRegistered

COURSE	
P	CourseID
K	
	Section
	Term
	Year
	Title
	Room

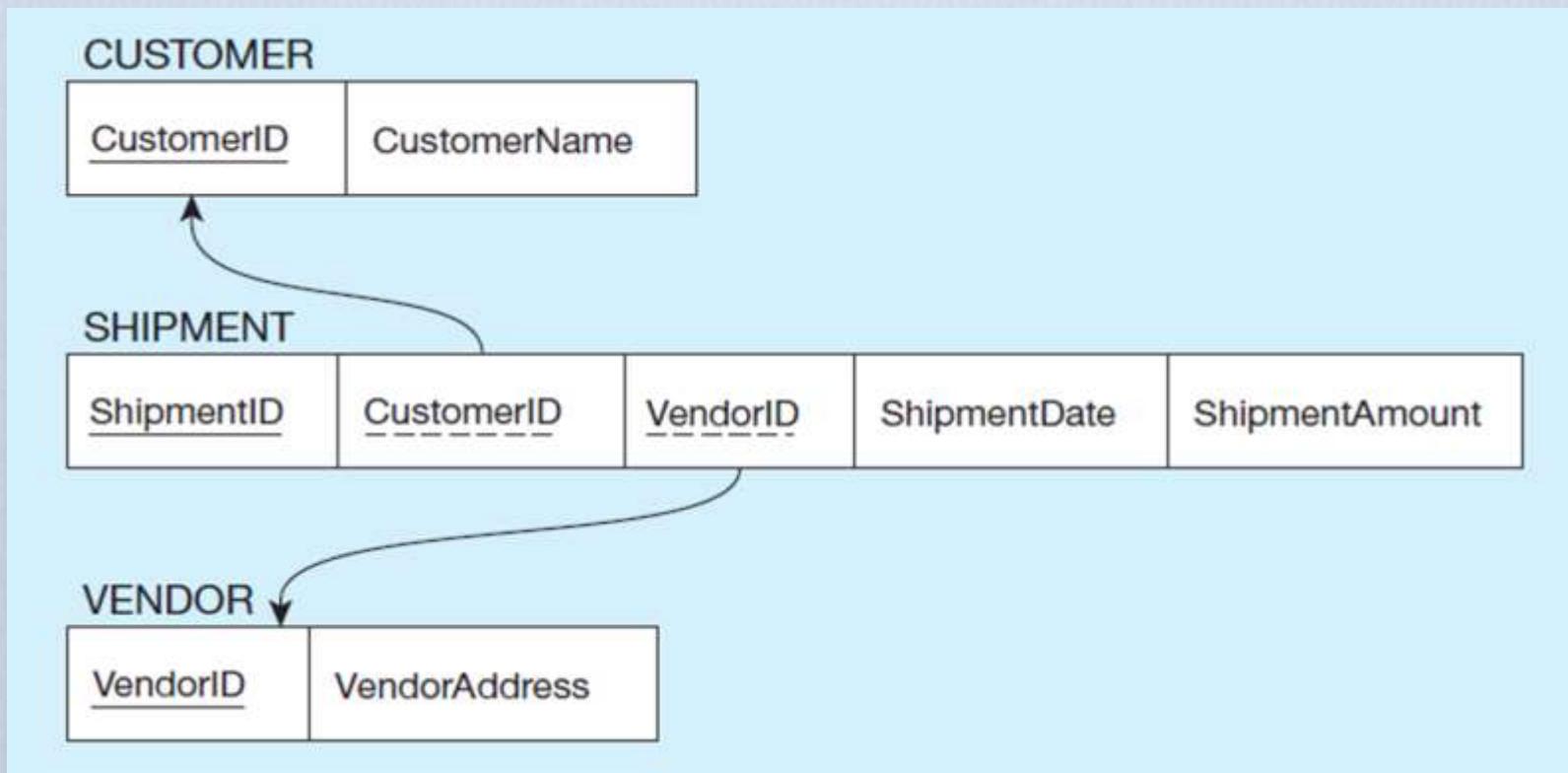


Example of Mapping an Associative Entity with an Identifier (1 of 2)

a) SHIPMENT associative entity



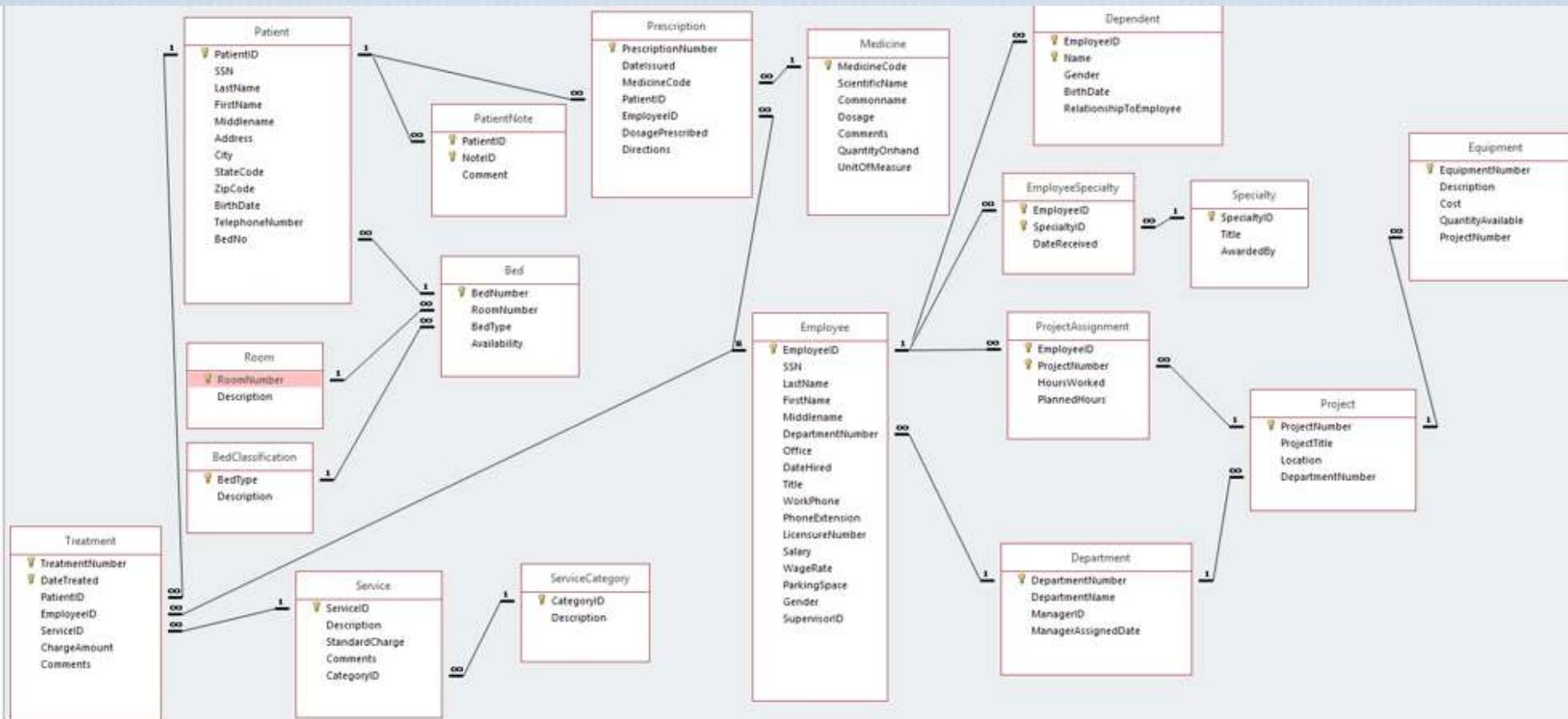
Example of Mapping an Associative Entity with an Identifier (2 of 2)



If an associative entity is created using a new identifier (primary key) we must STILL Connect that associative entity with its related entities. These are two 1:m Relationships, and the PK of the 1 MUST become a FK in the many.

To recap with foreign keys:

- ❖ Mapping Binary Relationships
 - ❖ **One-to-Many** – Primary key on the one side becomes a foreign key on the many side
 - ❖ **Many-to-Many** – Create a **new relation** with the primary keys of the two entities as its primary key
 - ❖ **One-to-One** – Primary key on mandatory side becomes a foreign key on optional side



Single Table Queries

Week 2 Chapter 4 Video 5
ORDER BY command
Dr. Anne Powell

Agenda

- ❖ 1. Basic SELECT commands
- ❖ 2. Column Formatting
- ❖ 3. Common Errors
- ❖ 4. DISTINCT – WHERE commands
- ❖ 5. ORDER BY command**
- ❖ 6. In-class practice

ORDER BY Clause

- ❖ Output from a SELECT statement can be sorted by using the optional ORDER BY clause.

```
/* SQL Example Order By */  
SELECT LastName, FirstName  
FROM Employee  
WHERE LastName >= 'J'  
ORDER BY LastName;  
  
LASTNAME          FIRSTNAME  
-----  
Jones             Quincey  
Klepper          Robert  
Quattromani      Toni  
Schultheis       Robert  
more rows will be displayed...
```

ORDER BY – Numeric Descending

```
SELECT LastName, FirstName, Salary  
FROM Employee  
WHERE Salary > 25000  
ORDER BY Salary DESC;
```

LASTNAME	FIRSTNAME	SALARY
Smith	Susan	32500
Jones	Quincey	30550
Barlow	William	27500

ORDER BY - Multiple Columns

- ❖ You can sort within a sort by specifying a major sort column and minor sort column.
- ❖ The major sort column is specified first, then a comma, then the minor sort column.

```
/* SQL Example Multiple Order Columns */  
SELECT DepartmentNumber, LastName, FirstName  
FROM Employee  
ORDER BY DepartmentNumber, LastName;
```

DEPARTMENTNUMBER	LASTNAME	FIRSTNAME
1	Bock	Douglas
1	Eakin	Maxwell
2	Bordoloi	Bijoy
2	Smith	Alyssa
2	Webber	Eugene
3	Barlow	William

Combining DESC with ORDER BY

- ❖ The major sort column is descending and the minor sort column is ascending within *DepartmentNumber*.

```
SELECT DepartmentNumber, LastName, FirstName  
FROM Employee  
ORDER BY DepartmentNumber DESC, LastName;
```

DEPARTMENTNUMBER	LASTNAME	FIRSTNAME
9	Schultheis	Robert
8	Adams	Adam
8	Boudreaux	Beverly
8	Clinton	William
8	Simmons	Lester
8	Thornton	Billy
7	Boudreaux	Betty

Class Examples

Week 3 Chapter 6 Video 5
Examples

Dr. Anne Powell

Agenda

1. Aggregate Function Rules – COUNT
2. Aggregate Function Rules – the rest
3. GROUP BY command
4. HAVING command
5. Examples

In-Class Examples (ICE)

- ❖ 1. A new government reporting regulation requires the hospital to report the number of regular beds in use by the hospital. The `bed_type` codes for these beds are R1, R2, or R3. The information is stored in the *BedType* column of the BED table. The result table should have a single output column labeled **Number of Regular Beds**.

ICE 2

- ❖ 2. A change in the government reporting regulation regarding available hospital beds requires a count by type of all types of beds *that are available*. Display the information as two columns, one for *BedType* and one for the associated count. The result table should have several rows, one for each bed type. Use a single query. Additionally, the *BedType* output column should be formatted as A8 and have a heading of **Bed Type**. The count column should have a heading of **Number Counted**. Use a GROUP BY clause.

ICE 3-5

- ❖ 3. Another government regulation requires a report of the number of services provided by the hospital counted by the standard service categories. The data is stored in the SERVICE table. Display the result table as two columns, one for *CatagoryID* and one for the associated count. The result table should have several rows, one for each service category. Use a single query. The column headings should be **Service Category** and **Number of Services**. You will need to use a GROUP BY clause. Service Category needs to be formatted A16.
- ❖ 4. Create a new query providing the same information as in #3, but limit the display rows to those services that have been used at least 20 times.
- ❖ 5. Rewrite query #3 to exclude the counting of services in categories injections and laboratories (codes = INJ, LAB).

MULTI-TABLE QUERIES

Week 4 Chapter 7 Video 5

In Class Examples

Dr. Anne Powell

Examples related to Video 1

- ❖ Provide a list of patients who have been prescribed prescriptions. Include both the patient name and the prescription number.
- ❖ HR would like a listing of all employee first names and last names that includes the department name they work in as well as their manager's ID.
- ❖ Provide a list of projects that need equipment costing more than \$500. Include the project name, equipment name, and the cost of the equipment. Order by equipment cost.
- ❖ Management needs a listing of beds that are classified as bed type E2 Emergency Room-fixed. Include the BedType and Description of the bed type as well as the Room number and bed number that an E2 bed can be found in.

Examples related to Video 2

- ❖ Provide a list of all patient names who have been prescribed a prescription along with the name of the employee who prescribed the prescription. Include the prescription number also.
- ❖ Provide a list of all patient names who have been prescribed a prescription with the CommonName ‘Demerol’. This query joins 3 tables. Why? We are only asking for data from 2 tables.
- ❖ The Chief of Pharmacy requires a listing of patients receiving prescriptions. The result table must display the patient name (last and first concatenated from the patient table), medicine common name (CommonName from the medicine table), and employee who prescribed the medicine (last and first name concatenated from the employee table). Sort the result table by the patient last then first name. This query joins four tables. Why? We are only asking for data from 3 tables.

Examples related to video 3

- ❖ Management would like a report listing **all** department names and all the corresponding project titles that have been undertaken by each department. (Video 3 Outer Join because the report must contain **ALL** department names, even those who may not be doing any projects at this time).
- ❖ An inventory report is needed that lists all the projects currently undertaken as well as equipment being used by each project. Even if the project is not using any equipment, management wants to see all projects in the report.
- ❖ The Party Planning committee is making nametags for all employees and their dependents for the upcoming Christmas party. They need to know the first names of all employees and all dependents (don't filter out duplicates!).

Example related to Video 4

- ❖ Create a VIEW that shows all employee names with the department name they are affiliated with.

Subqueries

CMIS 563

Week 5 Chapter 8 Video 5
In Class Examples
Dr. Anne Powell

Agenda

- ❖ 1. Explanation of subqueries / Subquery Rules
- ❖ 2. Using IN, ORDER BY, and comparison operators
- ❖ 3. Using ANY and ALL keywords / Nest subqueries at multiple levels
- ❖ 4. Correlated subqueries / EXISTS operator
- ❖ 5. In-class examples

In Class Examples

- ❖ The Prescription table stores data about medicine that has been prescribed for patients. The Chief of Pharmacology needs a listing of patients (last and first names) that have received either valium or Flexeril (medicineCode = '9999003' or '9999008'). Use a subquery approach.
- ❖ The nursing director wants a list of all bed number and the room number the bed is in for the Radiology ward. These beds have Radiology in their description. You don't know how many bedTypes might have Radiology in their description – or if any more might be added in the future. You need to make sure you query takes into account more bedTypes may be added in the future.

In Class Examples

- ❖ Management needs a listing of employee names (last and first) who have worked more than 15 hours on a project. Each employee only needs to be listed once.

- ❖ Management needs a listing of patient names (last and first) who have received a service treatment ‘General Panel’ (description column of the Service table); however, you do not know the serviced for this service. Your query should have a subquery within a subquery using the Patient, Treatment, and Service tables. Sort the output by patient last name then by patient first name.

In Class Examples

- ❖ Accounting needs a list of all services that have a standard charge greater than the average cost of all services.

- ❖ Accounting also needs a list of all services that have a standard charge greater than \$500. Use the ALL command for this example.

In Class Examples

- ❖ Who are the least paid employees with different titles?
Display the employee names (firstname and lastname concatenated), Title, and the Salary. Format so all columns fit on one line (title: a15; lastName a12; firstName a12)

Single Table Queries

Week 2 Chapter 4 Video 6
In class examples
Dr. Anne Powell

Agenda

- ❖ 1. Basic SELECT commands
- ❖ 2. Column Formatting
- ❖ 3. Common Errors
- ❖ 4. DISTINCT – WHERE commands
- ❖ 5. ORDER BY command
- ❖ **6. In-class practice**

Let's try some examples

- ❖ 1) Run a query that displays the common name and quantity on hand of the medications used at the hospital.
- ❖ 2) Run a query that displays the first, last, and middle names of employees as well as the date hired.
- ❖ 3) We need to know all the dependents in our database. Run a query that lists the dependent names, employeeID associated with the dependent name, and relationship to employee.
- ❖ 4) Run a query that displays the common medication name and quantity on hand sorted from largest to smallest.
- ❖ 5) We need to know which employee has been overprescribing certain medicines. Run a query that displays the EmployeeID, Medicine Code, and Dosage Prescribed sorted by employee ID and then by dosage prescribed.
- ❖ 6) Run a query that displays the first, last, and middle names of employees as well as the date hired. Sort the results by most recent date hired and then by last name.

Questions?



Adding Power to your Queries

Week 2 Chapter 5 Video 7
Aliases and Logical Operators
Dr. Anne Powell

Agenda

- ❖ 7. Aliases
- ❖ 7. Logical Operators
- ❖ 8. Ranges and Lists
- ❖ 9. Character Matching
- ❖ 9. NULL
- ❖ 10. Math and attributes
- ❖ 11. In-class practice

Aliases

- ❖ If the defined attribute (column) name in your database is too long, or you would just prefer to have the report display a name other than the attribute name, you can use an alias.
- ❖ In the example below, the resulting report will NOT have a column header of LastName, but will have a column header of Last Name – much more readable and professional.

```
SELECT LastName "Last Name", FirstName "First Name",
       DateHired "Date Hired", Gender, Salary
```

Logical Operators

- AND – joins two or more conditions, and returns results only when all of the conditions are true.
- OR – joins two or more conditions, and returns results when any of the conditions are true.
- NOT – negates the expression that follows it – a false condition evaluates as true and a true condition evaluates as false.

AND

and note Aliases of Late Name, First Name, Date Hired

```
COLUMN "Date Hired" FORMAT A10;
COLUMN Salary FORMAT $99,999.99
SELECT LastName "Last Name", FirstName "First Name",
       DateHired "Date Hired", Gender, Salary
FROM Employee
WHERE LastName > 'E' AND DateHired > '20-Jun-71'
      AND Gender = 'M' AND Salary > 20000
ORDER BY LastName;
```

Last Name	First Name	Date Hired	Gender	SALARY
Jones	Quincey	01-JAN-90	M	\$30,550.00
Simmons	Lester	03-MAR-98	M	\$22,000.00

2 rows selected.

AND

- ❖ NOTE: if you are doing a compare using AND and it is associated with the SAME attribute, you must list the attribute twice!

```
SELECT LastName "Last Name", FirstName "First  
Name", Salary "Salary"  
  
FROM Employee  
  
WHERE Salary >= 5000 AND Salary <= 10000
```

AND Operator – Empty Result Table

- ❖ Here the two simple conditions joined by the AND operator yield an empty result table.
- ❖ Employee 261-22-3803 Gender is ‘M’.

```
/* SQL Example 5.3 */  
  
SELECT LastName, FirstName  
FROM Employee  
  
WHERE SSN = '261223803' AND Gender = 'F';  
no rows selected.
```

OR

```
SELECT LastName "Last Name", FirstName "First Name",
Gender
FROM Employee
WHERE Gender = 'F' OR LastName >= 'T'
ORDER BY LastName;
```

Last Name	First Name	Gender
-----	-----	-----
Becker	Roberta	F
Boudreaux	Beverly	F
<i>...some rows have been deleted here.</i>		
Thornton	Billy	M
Young	Yvonne	F
Zumwalt	Mary	F

13 rows selected.

The Logical NOT Operator

- The NOT operator can simplify writing a WHERE condition.
- Example: You need to produce a listing of all employees that are NOT assigned to Department 7.
- Without the NOT operator, the WHERE condition might look similar to this:
 - ❖ `WHERE DepartmentNumber = 1 OR DepartmentNumber = 2 OR DepartmentNumber = 3 OR ... (more clauses) !`
- The next slide shows the query with the NOT operator.

NOT

```
SELECT LastName "Last Name", FirstName  
      "First Name", DepartmentNumber "Dept"  
FROM Employee  
WHERE NOT DepartmentNumber = 7  
ORDER BY LastName;
```

Last Name	First Name	Dept
-----------	------------	------

-----	-----	-----
-------	-------	-------

Adams	Adam	8
-------	------	---

Barlow	William	3
--------	---------	---

Becker	Robert	3
--------	--------	---

more rows will display (a total of 23)...

Combining OR and AND

Order of operators is important – AND is done first

```
SELECT LastName "Last Name", FirstName "First  
Name", DepartmentNumber "Dept", Gender  
FROM Employee  
WHERE LastName > 'T' AND Gender = 'F'  
      OR DepartmentNumber = 2  
ORDER BY LastName;
```

Last Name	First Name	Dept	Gender
Bordoloi	Bijoy	2	M
Smith	Alyssa	2	F
Webber	Eugene	2	M
Young	Yvonne	6	F
Zumwalt	Mary	4	F

5 rows selected.

Combining OR and AND

AND is done first UNLESS there are parentheses

if what you really wanted was everyone who's last name is > T And then of those whose last name is > T you want either females or those in department 2

```
SELECT LastName "Last Name", FirstName "First  
Name",  
       DepartmentNumber "Dept", Gender  
FROM Employee  
WHERE LastName > 'T' AND  
      (Gender = 'F' OR DepartmentNumber = 2)  
ORDER BY LastName;
```

Last Name	First Name	Dept	Gender
Webber	Eugene	2	M
Young	Yvonne	6	F
Zumwalt	Mary	4	F

3 rows selected.

Adding Power to your Queries

Week 2 Chapter 5 Video 8

Ranges and Lists

Dr. Anne Powell

Agenda

- ❖ 7. Aliases
- ❖ 7. Logical Operators
- ❖ **8. Ranges and Lists**
- ❖ 9. Character Matching
- ❖ 9. NULL
- ❖ 10. Math and attributes
- ❖ 11. In-class practice

Ranges and Lists

- ❖ IN or NOT IN – finds results in/out of a non-consecutive group of values
- ❖ BETWEEN or NOT BETWEEN – finds results that are within or outside of a consecutive group

Example

- ❖ We want to know certain data about beds, but only for the bednumbers 33, 2001, 103 ,1001, and 1004.

IN

```
SELECT BedNumber, RoomNumber, BedType, Availability  
FROM Bed  
WHERE BedNumber = 33 OR BedNumber = 103 OR BedNumber = 1001  
      OR BedNumber = 1004 OR BedNumber = 2001  
ORDER BY BedType;
```

BEDNUMBER	ROOMNUMBER	BEDTYPE	AVAIL
-----	-----	-----	-----
33	MSN214	R1	N
2001	SW3001	R1	Y
103	RA0077	RA	Y
1001	SUR001	SU	
1004	SUR004	SU	

5 rows selected.

```
WHERE BedNumber IN (33, 103, 1001, 1004, 2001)
```

NOT IN

```
SELECT BedNumber, RoomNumber, BedType, Availability  
FROM Bed  
  
WHERE BedType NOT IN ('E1', 'E2', 'E3', 'P1', 'R2', 'R3', 'SU')  
ORDER BY BedType;
```

BEDNUMBER	ROOMNUMBER	BEDTYPE	AVAIL
5004	PED104	P2	Y
5005	PED105	P2	Y
5006	PED111	P2	N

more rows will display (a total of 39)...

- ❖ *Note the single quotes around the text in the WHERE statement – also note that what is in single quotes must match data exactly! ‘e1’ would result in E1 beds being shown.*

IN and NOT IN – Common Errors –

Missing Commas Where Required

```
/* SQL Example 5.15 */

SELECT BedNumber, RoomNumber, BedType, Availability
FROM Bed
WHERE BedNumber IN (33 103 1001 1004 2001)
ORDER BY BedType;
```

ERROR at line 3:

ORA-00907: missing right parenthesis

BETWEEN

```
/* SQL Example Between */  
SELECT LastName "Last Name", FirstName "First Name",  
       Salary "Salary"  
FROM Employee  
WHERE Salary >= 5000 AND Salary <= 10000  
ORDER BY Salary;
```

Last Name	First Name	Salary
-----	-----	-----
Adams	Adam	\$5,500.00
Zumwalt	Mary	\$6,500.00

2 rows selected.

```
WHERE Salary BETWEEN 5000 AND 10000
```

Combining Between, AND, OR

```
SELECT LastName "Last Name", FirstName "First Name",
       Salary "Salary"
  FROM Employee
 WHERE Salary BETWEEN 4000 AND 5000
   OR Salary BETWEEN 6000 AND 8000
 ORDER BY Salary;
```

Last Name	First Name	Salary
-----	-----	-----
Smith	Alyssa	\$4,550.00
Brockwell	Mary Ellen	\$4,800.00
Boudreaux	Betty	\$4,895.00
Zumwalt	Mary	\$6,500.00

4 rows selected.

NOT BETWEEN

```
SELECT LastName "Last Name", FirstName  
      "First Name", Salary "Salary"  
FROM Employee  
WHERE Salary NOT BETWEEN 3000 AND 30000  
ORDER BY Salary;
```

Last Name	First Name	Salary
Simmons	Leslie	\$2,200.00
Young	Yvonne	\$2,200.00
Jones	Quincey	\$30,550.00
Smith	Susan	\$32,500.00

4 rows selected.

BETWEEN Operator – Common Errors –

Comma Where None Is Allowed

```
/* SQL Example 5.21 */
```

```
SELECT LastName "Last Name", Salary "Salary"
```

```
FROM Employee
```

```
WHERE Salary BETWEEN 25,000 and 40,000;
```

ERROR at line 3:

ORA-00905: missing keyword

Adding Power to your Queries

Week 2 Chapter 5 Video 9
Character Matching and Nulls
Dr. Anne Powell

Agenda

- ❖ 7. Aliases
- ❖ 7. Logical Operators
- ❖ 8. Ranges and Lists
- ❖ 9. Character Matching**
- ❖ 9. NULL**
- ❖ 10. Math and attributes
- ❖ 11. In-class practice

Character Matching

- ❖ LIKE or NOT LIKE
- ❖ Wildcards
 - ❖ % - Matches any string of zero or more characters,
 - ❖ _ (underscore) - Matches any single character.
 - ❖ [] (brackets) - any single character within a specified range such as 'a'to 'd', inclusive [a-d] or a set of characters such as [aeiouy]
 - ❖ [^] (not brackets) any single character **not** in the specified range or set. (e.g., [^a-f])
- ❖ **NOTE*** Comparison operators (=, >, <, etc) cannot be used with wildcards.

More Examples

- LIKE '**%inger**' will search for every name that ends with 'inger' (Ringer, Stringer).
- LIKE '**%en%**' will search for every name that has the letters 'en' in the name (Bennet, Green, McBadden, Enright).
- LIKE '**_heryl**' will search for every six-letter name ending with 'heryl' (Cheryl). Notice how this is different than '**%heryl**' which would return names that are six characters or more.

LIKE

```
SELECT LastName "Last Name", FirstName "First  
Name"  
FROM Employee  
WHERE LastName LIKE 'Bo%';
```

Last Name	First Name
Boudreaux	Beverly
Bock	Douglas
Bordoloi	Bijoy
Boudreaux	Betty

4 rows selected.

Combining LIKE and NOT LIKE

```
SELECT TreatmentNumber, PatientID, EmployeeID  
FROM Treatment  
WHERE Comments LIKE '%EKG%' AND EmployeeID NOT LIKE  
'01%';
```

TREATMENTNUMBER PATIENTID EMPLOYEEID

7	100305	66425
8	100504	66532
1	100303	88777

3 rows selected.

Common Error

- ❖ Comparison operators (=, >, <, etc) cannot be used with wildcards. This query does not work ... replace the equal sign (=) with the LIKE operator.

```
/* SQL Example 5.26 */  
  
SELECT DISTINCT EmployeeID  
  
FROM ProjectAssignment  
  
WHERE EmployeeID = '%32';
```

no rows selected.

- ❖ What needs to be done to correct this query?

Unknown Values – IS NULL and IS NOT NULL

- NULL value is the absence of a value – it is **not synonymous** with "zero" (numerical values) or "blank" (character values).
 - NULL values allow users to distinguish between a deliberate entry of zero/blank and a non-entry of data.
- ❖ /* SQL Example 5.27 */
- ❖ SELECT LastName "Last Name", FirstName "First Name",
❖ WageRate "Wage"
❖ FROM Employee
❖ WHERE Salary IS NULL;
- ❖ Last Name First Name Wage
- ❖ ----- ----- -----
- ❖ Thornton Billy 8.28
- ❖ Clinton William 7.35
- ❖ 2 rows selected.

IS NULL

```
SELECT LastName "Last Name", FirstName "First Name",
       WageRate "Wage"
  FROM Employee
 WHERE Salary IS NULL;
```

Last Name	First Name	Wage
Thornton	Billy	8.28
Clinton	William	7.35

2 rows selected.

IS NOT NULL

```
SELECT *
FROM ProjectAssignment
WHERE HoursWorked IS NOT NULL;
EMPLO PROJECTNUMBER HOURSWORKED PLANNEDHOURS
-----
33358          5        41.2          65
66532          7        14.8
67555          7        12.2          15.8
33344          8        23            21
67555          8        24.1          18
```

16 rows selected.

Example

```
❖ /* SQL Example 5.29 */  
❖ SELECT *  
❖ FROM ProjectAssignment  
❖ WHERE HoursWorked = 0;  
❖ no rows selected.
```

- The query did not return a result table because none of the rows in the assignment table have a zero value for work_hours. Thus, you can see that zero (0) is a value, not an "unknown value."

Adding Power to your Queries

Week 2 Chapter 5 Video 10

Math and attributes

Dr. Anne Powell

Agenda

- ❖ 7. Aliases
- ❖ 7. Logical Operators
- ❖ 8. Ranges and Lists
- ❖ 9. Character Matching
- ❖ 9. NULL
- ❖ **10. Math and attributes**
- ❖ 11. In-class practice

Expressions in WHERE Clause

- ❖ An expression is formed by combining a column name or constant with an arithmetic operator.
- ❖ When expression is used in a SELECT clause with a column name it has no effect on the table's underlying values.
- ❖ The result of an arithmetic operation on NULL is NULL.

Expression Operators

SYMBOL	OPERATION	ORDER
*	Multiplication	1
/	Division	1
+	Addition	2
-	Subtraction	2

Multiplication

```
COLUMN "Annual Salary" FORMAT $9,999,999.99;
SELECT LastName "Last Name", FirstName "First
Name",
      Salary*12 "Annual Salary"
FROM Employee
WHERE Salary*12 > 300000
ORDER BY LastName;
```

Last Name	First Name	Annual Salary
Barlow	William	\$330,000.00
Jones	Quincey	\$366,600.00
Smith	Susan	\$390,000.00

3 rows selected.

Division

```
/* SQL Example Division */
SELECT EmployeeID, ProjectNumber "Project",
       HoursWorked/40 "Avg Hours/Week"
FROM ProjectAssignment
WHERE ProjectNumber = 8
ORDER BY EmployeeID;
```

EMPLOYEEID	Project	Avg Hours/Week
------------	---------	----------------

23232	8	
33344	8	.575
67555	8	.6025

3 rows selected.

Filtering Out NULL Values

```
/* SQL Example 5.35 */
SELECT EmployeeID, ProjectNumber "Project",
       HoursWorked/40 "Avg Hours/Week"
FROM ProjectAssignment
WHERE ProjectNumber = 8 AND
      HoursWorked IS NOT NULL
ORDER BY EmployeeID;
```

EMPLOYEEID	Project	Avg Hours/Week
33344	8	.575
67555	8	.6025

2 rows selected.

Adding Power to your Queries

Week 2 Chapter 5 Video 11

In class examples

Dr. Anne Powell

Agenda

- ❖ 7. Aliases
- ❖ 7. Logical Operators
- ❖ 8. Ranges and Lists
- ❖ 9. Character Matching
- ❖ 9. NULL
- ❖ 10. Math and attributes
- ❖ 11. In-class practice

Let's try some examples

- 1) We need to know what equipment has an original cost between \$1,000 and \$5,000.
- 2) We need to know what equipment has an original cost that is less than \$1,000 or more than \$5,000.
- 3) We need a list of bed numbers, room numbers, bedtypes, and availability – but only for bed types of RA, SU, and E1. Sort the results by type of bed and then room number. Format room number (10), bedtype (7), and availability (5).
- 4) Provide a list of patients whose prescription dosage tells them to take their prescription orally. Provide the PatientID, EmployeeID, and Date Issued.
- 5) Provide a list of all employees who earn at least \$2,000 per week. Provide the First Name and Last Name as well as their weekly salary.

Questions?

