Chapters 5 & 6: Queries

CIS 399 Week 3
Chapter Overview

SELECT Statements
Selecting specific rows
Sorting data
Computations, Column aliases
Special functions in SELECT statements
Aggregate functions
Grouping results
The HAVING clause
Report formatting statements
The SELECT Statement

/* The SQL command that extracts information from relational database tables. Syntax:
SELECT [DISTINCT] <select-list>
FROM {<table list>}
[WHERE <conditions>]
[GROUP BY <group by list>]
[HAVING <group by conditions>]
[ORDER BY <order list> [ASC|DESC]]
[FOR UPDATE <for update options>]; */
SELECT examples

/* select all columns, all rows */
SELECT * FROM students;

/* select all columns, one row */
SELECT * FROM students WHERE id = 950123456;

/* select one column, one row */
SELECT first_name FROM students WHERE id = 950123456;
The DISTINCT keyword

/* when selecting text, you may get duplicates. To eliminate duplicates, use the DISTINCT keyword. */

/* get the distinct first names */
SELECT DISTINCT first_name FROM students;
Filtering results with a WHERE clause

/* similar behavior for update, delete clauses. Same relational, logical operators. Syntax:
WHERE <column name> <comparison operator> <search expression>
Examples:
WHERE first_name = ‘Fred’
WHERE student_id = 950123456
WHERE last_name <> ‘Jones’ */
Character strings and dates

/*! the same rules apply as before…
The default date format is DD–MMM–YYYY. Dates and strings must be delimited with single quotes. String comparisons are case sensitive. New operator – IS NULL: selects rows where that column value is null (column = null DOESN’T WORK!) */
Logical operator precedence

/*Parentheses are evaluated first. Then <, =, >. Then logical operators (NOT, AND, OR). All in a left-to-right manner.

Example: WHERE city = 'Arcata' AND yearbuilt < 2004 OR bedrooms = 3

*/
Sorting with ORDER BY

/* ORDER BY clause controls the order in which rows are displayed. Syntax:

ORDER BY {<column-name>|<result-column-number>} [ASC | DESC]
[NULLS FIRST | NULLS LAST]
[, {<column-name>|<result-column-number>} [ASC | DESC]
[NULLS FIRST | NULLS LAST]}… */
ORDER BY example

/* select the graduating students, and sort by last name, first name */

SELECT first_name, last_name
FROM students
WHERE year = 2
ORDER BY 2 ASC, 1 ASC NULLS FIRST;
Top-N queries in Oracle

/* some databases give you a “TOP 10” or “FIRST 10” keyword:
SELECT TOP 10 salary FROM employee
 ORDER BY salary DESC;
Not Oracle. To get the top 10 values, use an “inline view” subquery: */
SELECT salary
FROM (SELECT salary FROM employees
 ORDER BY salary DESC NULLS LAST)
 WHERE ROWNUM <= 10;
Computations in queries

/* +, -, *, / are all supported for NUMBER, DATE, INTERVAL types. */

SELECT 5*4 FROM DUAL; /*returns 20*/

/* to select a student’s age in days: */
SELECT last_name, first_name, SYSDATE - birthdate FROM students;
Getting age in years, months...

/* to select a student’s age in years: */
SELECT last_name, first_name, (SYSDATE – birthdate) / 365.24 FROM students;

/* we will see how else to work with dates later in the chapter… */
Concatenating Columns

/* use the “||” operator to concatenate column values */

SELECT first_name || ' ' || last_name FROM students;
Single row SQL Functions

Character Functions (UPPER, LOWER)
Numeric Functions (ABS, MOD, ROUND)
Conversion Functions (TO_CHAR)
Date Functions (SYSDATE)
Regular Expression Functions  new!
Special Functions (DECODE, TRANSLATE)
# Character functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTR(&lt;str1&gt;,&lt;str2&gt;)</td>
<td>Searches &lt;str1&gt; and returns the position at which &lt;str2&gt; occurs.</td>
</tr>
<tr>
<td>LENGTH(&lt;str&gt;)</td>
<td>Returns the number of characters in &lt;str&gt;.</td>
</tr>
<tr>
<td>LOWER(&lt;str&gt;)</td>
<td>Returns the lowercase version of &lt;str&gt;.</td>
</tr>
<tr>
<td>UPPER(&lt;str&gt;)</td>
<td>Returns the uppercase version of &lt;str&gt;.</td>
</tr>
<tr>
<td>INITCAP(&lt;str&gt;)</td>
<td>Converts the initial letter of each word of &lt;str&gt; to uppercase.</td>
</tr>
<tr>
<td>NVL(&lt;str&gt;, &lt;value&gt;)</td>
<td>Returns &lt;value&gt; if &lt;str&gt; is NULL; else, &lt;str&gt; is returned.</td>
</tr>
<tr>
<td>NVL2(&lt;str&gt;, &lt;value1&gt;, &lt;value2&gt;)</td>
<td>Returns &lt;value1&gt; if &lt;str&gt; is not NULL; else, returns &lt;value2&gt; if &lt;str&gt; is NULL.</td>
</tr>
<tr>
<td>REPLACE(&lt;str1&gt;,&lt;str2&gt;,&lt;str3&gt;)</td>
<td>Searches &lt;str1&gt; for occurrence of &lt;str2&gt;. If found, it replaces &lt;str2&gt; with &lt;str3&gt;.</td>
</tr>
<tr>
<td>LPAD(&lt;str&gt;, &lt;width&gt;, [,&lt;pad string&gt;])</td>
<td>Pads string &lt;str&gt; with spaces to the left to right align the &lt;str&gt; for a total width of &lt;width&gt;. You can supply an optional pad character.</td>
</tr>
<tr>
<td>RPAD(&lt;str&gt;, &lt;width&gt;, [,&lt;pad string&gt;])</td>
<td>Pads string &lt;str&gt; with spaces to the right to bring total length to &lt;width&gt; characters. You can supply an optional pad character.</td>
</tr>
<tr>
<td>SUBSTR(&lt;str&gt;, &lt;start&gt; [,&lt;length&gt;])</td>
<td>Returns the substring of &lt;str&gt; that begins at position &lt;start&gt;. &lt;Length&gt; is an optional length to return.</td>
</tr>
<tr>
<td>TRIM(&lt;str&gt;)</td>
<td>Trims spaces from the left and the right of &lt;str&gt;.</td>
</tr>
<tr>
<td>LTRIM(&lt;str&gt;)</td>
<td>Trims spaces from left side of &lt;str&gt;.</td>
</tr>
<tr>
<td>RTRIM(&lt;str&gt;)</td>
<td>Trims spaces from right side of &lt;str&gt;.</td>
</tr>
</tbody>
</table>
# Numeric Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(&lt;n&gt;)</td>
<td>Returns the absolute value of &lt;n&gt;.</td>
<td>ABS(5) = 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ABS(-58) = 58</td>
</tr>
<tr>
<td>CEIL(&lt;n&gt;)</td>
<td>Returns smallest integer greater than or equal to &lt;n&gt;.</td>
<td>CEIL(6.8) = 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CEIL(-34.5) = -34</td>
</tr>
<tr>
<td>FLOOR(&lt;n&gt;)</td>
<td>Returns largest integer less than or equal to &lt;n&gt;.</td>
<td>FLOOR(78.9) = 78</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FLOOR(-3.1) = -4</td>
</tr>
<tr>
<td>MOD(&lt;n&gt;,&lt;m&gt;)</td>
<td>Returns the remainder of &lt;n&gt; divided by &lt;m&gt;.</td>
<td>MOD(7,3) = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MOD(29,6) = 5</td>
</tr>
<tr>
<td>POWER(&lt;n&gt;,&lt;k&gt;)</td>
<td>Returns &lt;n&gt; raised to the &lt;k&gt; power.</td>
<td>POWER(2,3) = 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POWER(5,2) = 25</td>
</tr>
<tr>
<td>ROUND(&lt;n&gt; [,&lt;m&gt;])</td>
<td>Returns &lt;n&gt; rounded to &lt;m&gt; decimal places. If &lt;m&gt; is omitted, then rounds to nearest integer.</td>
<td>ROUND(7.467,2) = 7.47</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ROUND(96.87,−1) = 100</td>
</tr>
<tr>
<td>SIGN(&lt;n&gt;)</td>
<td>Returns −1 if &lt;n&gt; is negative, 0 if &lt;n&gt; is zero, and +1 if &lt;n&gt; is positive.</td>
<td>SIGN(−23) = −1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SIGN(456) = 1</td>
</tr>
<tr>
<td>SQRT(&lt;n&gt;)</td>
<td>Returns the square root of &lt;n&gt;.</td>
<td>SQRT(81) = 9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQRT(7) = 2.645751</td>
</tr>
<tr>
<td>TRUNC(&lt;n&gt; [,&lt;k&gt;])</td>
<td>Returns the truncated value of &lt;n&gt; to &lt;k&gt; decimal places. Second argument is optional.</td>
<td>TRUNC(56.999) = 56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUNC(789.559,1) = 789.5</td>
</tr>
</tbody>
</table>
# Date Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ADD_MONTHS(&lt;date&gt;, &lt;no. of months&gt;)</code></td>
<td>Adds specified &lt;no. of months&gt; to date value. (Subtracts if &lt;no. of months&gt; is negative.)</td>
<td><code>ADD_MONTHS('10–OCT–06',3) returns 10–JAN–07.</code></td>
</tr>
<tr>
<td><code>EXTRACT(&lt;ymd&gt; FROM &lt;date&gt;)</code></td>
<td>Extracts an integer value that is the year, month, or day of the date.</td>
<td><code>EXTRACT(YEAR FROM '10–OCT–85') returns 1985.</code></td>
</tr>
<tr>
<td><code>LAST_DAY(&lt;date&gt;)</code></td>
<td>Returns the last day of the month that contains the date.</td>
<td><code>LAST_DAY(SYSDATE) returns the last day of this month.</code></td>
</tr>
<tr>
<td><code>MONTHS_BETWEEN(&lt;date1&gt;, &lt;date2&gt;)</code></td>
<td>Returns the months between two dates. If &lt;date1&gt; is more recent than &lt;date2&gt;, then the result is positive.</td>
<td><code>MONTHS_BETWEEN (SYSDATE, HireDate) returns the number of elapsed months between today and the HireDate field for each row.</code></td>
</tr>
<tr>
<td><code>NEW_TIME(&lt;date&gt;,&lt;cur-time-zone&gt;,&lt;new-time-zone&gt;)</code></td>
<td>Returns the date and time in another time zone.</td>
<td><code>NEW_TIME(SYSDATE, 'PST', 'EST')</code></td>
</tr>
<tr>
<td><code>NEXT_DAY(&lt;date&gt;, &lt;string&gt;)</code></td>
<td>Returns the date of the first day of the specified name (&lt;string&gt;) that is later than the date, &lt;date&gt;, specified.</td>
<td><code>NEXT_DAY('14–JUN–2006', 'Tuesday') returns the date '20–JUN–06.'</code></td>
</tr>
<tr>
<td><code>SYSDATE</code></td>
<td>Returns the current date and time. (SYSDATE has no arguments.)</td>
<td>'07–OCT–06' if that is the current date.</td>
</tr>
<tr>
<td><code>TO_CHAR(&lt;date&gt;,&lt;format&gt;)</code></td>
<td>Convert date/time to a string whose format is specified by &lt;format&gt;.</td>
<td><code>TO_CHAR(SYSDATE, 'MM/DD/YYYY HH24:MM;SS')</code></td>
</tr>
<tr>
<td><code>TO_DATE(&lt;str&gt;,&lt;format&gt;)</code></td>
<td>Convert &lt;str&gt; to a date using &lt;format&gt; to interpret the string.</td>
<td><code>TO_DATE('10/30/2006' 'MM/DD/YYYY')</code></td>
</tr>
</tbody>
</table>
Conversion Functions

TO_CHAR (<date or number> [, <format>])
TO_DATE (<string> [, <format>])
TO_NUMBER (<string> [, <format>])
## Number formatting

<table>
<thead>
<tr>
<th>Element</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>9999</td>
<td>Returns a value with the specified number of digits and leading space, if positive or hyphen is negative. The 9s dictate the maximum number of digits to display.</td>
</tr>
<tr>
<td>, (comma)</td>
<td>9,999</td>
<td>Returns a comma in the specified location. Comma cannot occur as the first element or after a decimal character.</td>
</tr>
<tr>
<td>. (period)</td>
<td>99.99</td>
<td>Returns a period and marks the location of the beginning of any decimal places.</td>
</tr>
<tr>
<td>0</td>
<td>0999</td>
<td>Returns leading zeroes.</td>
</tr>
<tr>
<td></td>
<td>9990</td>
<td>Returns trailing zeroes.</td>
</tr>
<tr>
<td>$</td>
<td>$999,999</td>
<td>Returns a leading dollar sign.</td>
</tr>
<tr>
<td>B</td>
<td>B999</td>
<td>Returns blanks for the integer part of a fixed number when the integer part is zero.</td>
</tr>
<tr>
<td>MI</td>
<td>9999MI</td>
<td>When the value is negative, it returns a trailing minus sign (-). Positive values return a trailing blank.</td>
</tr>
<tr>
<td>RN</td>
<td>RN</td>
<td>Returns a value as Roman numerals in uppercase.</td>
</tr>
<tr>
<td></td>
<td>rn</td>
<td>Returns a value as Roman numerals in lowercase.</td>
</tr>
<tr>
<td>S</td>
<td>S9999 or 9999S</td>
<td>Returns a <em>leading</em> minus sign (S999) for negative values or a positive sign (+) for a positive value.</td>
</tr>
<tr>
<td>EEEE</td>
<td>9.99EEEE</td>
<td>Returns a value in scientific notation (e.g., 1.78E+03).</td>
</tr>
</tbody>
</table>
# Date Formatting

<table>
<thead>
<tr>
<th>Perspective</th>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Century</td>
<td>CC</td>
<td>Two digit century.</td>
<td>20, 21</td>
</tr>
<tr>
<td>Quarter</td>
<td>Q</td>
<td>One–digit quarter of the year.</td>
<td>1, 2, 3, or 4</td>
</tr>
<tr>
<td>Year</td>
<td>YYYY, YYY, YY, Y, RR</td>
<td>Year represented by four, three, two, or one digit. RR is rounded to nearest year.</td>
<td>2006, 006, 06, 6</td>
</tr>
<tr>
<td>Month</td>
<td>MONTH, Month, MON, Mon</td>
<td>Month represented as full name or abbreviated. Capitalization matches result.</td>
<td>JANUARY, January, JAN, or Jan, respectively</td>
</tr>
<tr>
<td></td>
<td>MM</td>
<td>Two–digit month of the year.</td>
<td>01, 02, ...,12</td>
</tr>
<tr>
<td>Week</td>
<td>WW, W</td>
<td>Two–digit week of the year or one–digit week of the month.</td>
<td>01–52; 1–5</td>
</tr>
<tr>
<td>Day</td>
<td>DDD, DD, D</td>
<td>Three–digit day of the year; two–digit day of the month; one–digit day of the week.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DAY, Day, DY, Dy</td>
<td>Day of the week: complete, abbreviated, and capitalized or not.</td>
<td>SUNDAY, Sunday, SUN, Sun</td>
</tr>
</tbody>
</table>
## Time Formatting

<table>
<thead>
<tr>
<th>Perspective</th>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hour</td>
<td>HH24, HH</td>
<td>Two-digit hour in 24- or 12-hour format.</td>
<td>23, 11</td>
</tr>
<tr>
<td>Minute</td>
<td>MI</td>
<td>Two-digit minute.</td>
<td>Range: 0–59</td>
</tr>
<tr>
<td>Second</td>
<td>SS</td>
<td>Two-digit second.</td>
<td>Range 0–59</td>
</tr>
<tr>
<td>Separators</td>
<td>–/;:,·:</td>
<td>Characters that you can use to separate date and time values.</td>
<td>DD–MON–YYYY; HH:MM:SS; YYYY/MM/DD</td>
</tr>
<tr>
<td>Time suffixes</td>
<td>AM, A.M., PM, P.M.</td>
<td>AM, A.M., PM, or P.M. as appropriate (specify one or the other).</td>
<td>12:45 P.M.</td>
</tr>
<tr>
<td></td>
<td>AD, BC, A.D., B.C.</td>
<td>AD, BC, A.D., or B.C. as needed. Specify only AD or BC (without or without periods).</td>
<td>1452 B.C.</td>
</tr>
<tr>
<td></td>
<td>TH</td>
<td>Supplies suffix for numbers as needed.</td>
<td>1ST, 2ND, 3RD, or 15TH</td>
</tr>
<tr>
<td></td>
<td>SP</td>
<td>Number is spelled out.</td>
<td>DDSP produces FIFTEEN for a day value of 15</td>
</tr>
<tr>
<td></td>
<td>TZR</td>
<td>Time zone region.</td>
<td>PST, EST</td>
</tr>
</tbody>
</table>

CIS 399 - Week 3  Chapters 5 and 6 - Queries
Special Functions

/* DECODE function allows for conditional substitution. Example: */

/* NVL and NVL2 replace null with a value. Example – assume null is incomplete: */
SELECT NVL(grade, ‘I’) from grades;
Translate function

/* the TRANSLATE function converts the occurrences of characters in one string with corresponding characters in another string. Example: */

SELECT first_name, last_name, ssn
FROM employees
WHERE TRANSLATE (ssn, '0123456789', 'XXXXXXXXXX') != 'XXX-XX-XXXX';
# Aggregate Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG(&lt;n&gt;)</td>
<td>Returns the average of &lt;n&gt;.</td>
</tr>
<tr>
<td>COUNT(*)</td>
<td>Returns the number of rows retrieved by the query.</td>
</tr>
<tr>
<td>MAX(&lt;n&gt;)</td>
<td>Returns the maximum value of &lt;n&gt;.</td>
</tr>
<tr>
<td>MIN(&lt;n&gt;)</td>
<td>Returns the minimum value of &lt;n&gt;.</td>
</tr>
<tr>
<td>SUM(&lt;n&gt;)</td>
<td>Returns the sum of &lt;n&gt;.</td>
</tr>
</tbody>
</table>
Aggregate Example

/* get the number, min, max and average sale price for houses in a zip code */

SELECT COUNT(sale_price), MIN (sale_price), MAX (sale_price), AVG (sale_price) FROM listings WHERE zip_code = '12345';
GROUP BY clause

/* the GROUP BY clause is helpful in generating categorized summary statistics in a query. Example: */

/* get the count, min, max, avg for ALL zip codes */
SELECT zip_code, COUNT(sale_price), MIN (sale_price), MAX (sale_price), AVG (sale_price) FROM listings GROUP BY zip_code;
HAVING clause

/* The HAVING clause filters the results of a GROUP BY clause. (similar to a WHERE clause). Example: */

/* find the number of houses in each zip code having an average sale price of more than $100,000 */
SELECT zipcode, COUNT(*) FROM properties GROUP BY zipcode HAVING COUNT(*) > 10;
Formatting Output

/* COLUMN command sets the heading and formatting of a column */
COLUMN [column-name|alias] [options]

/* options: CLEAR, FORMAT, HEADING, WRAPPED, WORD WRAPPED, JUSTIFY, NULL, ON, OFF*/
COLUMN student_id FORMAT 999–99–9999 HEADING “Student ID”
Formatting, Cont.

/* setting headers and footers */
TTITLE | BTITLE [options [text|variable]...] | [ON|OFF]

/* options: LEFT, CENTER, RIGHT, SKIP */
/* variables: SQL.LNO, SQL.PNO, SQL.USER */

TTITLE LEFT SQL.USER CENTER ‘Properties Available’ RIGHT ‘Confidential’ SKIP 3
Chapter 6: Multitable Queries & Views

Joins: Natural, Inner, Outer, Self
Table Aliases
Set Operations
Subqueries
Correlated Subqueries
Simple and Complex Views
Inner Joins

/* INNER JOINs are Joins where one row from one table is matched with one or more rows from another table using a join condition. Syntax: */

SELECT <column list> FROM <table1> [alias1] [INNER] JOIN <table2> [<alias2>] ON {<table1>|<alias1>.<join column1> = <table2>|<alias2>.<join column2>};
INNER JOIN example

/* select property owners with their properties */

SELECT FirstName, LastName, Properties.Address, Properties.City
FROM Customers INNER JOIN Properties
ON Customers.CustomerID = Properties.OwnerID;
Natural Join

/* Natural joins are similar to inner joins, but no columns are specified. All columns with the same name & type are joined in an ‘equijoin’ condition.

Syntax: */

SELECT <column list> FROM <table1> [<alias1>] NATURAL JOIN <table2> [<alias2>];
/* select the property information for the current listings */

SELECT Address, City, Bedrooms, Bathrooms, SqFt, AskingPrice
FROM Properties NATURAL JOIN Listings
WHERE SqFt >= 2000 AND Bathrooms > 3 AND Bedrooms > 3
ORDER BY City, AskingPrice;
Complex INNER JOIN statements

/* Three or more tables! */

SELECT FirstName, LastName, Address, City, Bedrooms, Bathrooms, AskingPrice
FROM Agents
INNER JOIN Listings
ON AgentID = ListingAgentID
INNER JOIN Properties
ON Listings.PropertyID = Properties.PropertyID
INNER v. OUTER JOINs

inner join

left outer join

selected rows

CIS 399 - Week 3

Chapters 5 and 6 - Queries
OUTER JOIN statements

/* OUTER JOINs returns a row when the join condition is met, and when one of the join column conditions contains a null value. Syntax: */

SELECT <column list> FROM <table1>[<alias1>] {LEFT|RIGHT|FULL} [OUTER] JOIN <table2> [<alias2>] ON {<table1>|<alias1>}.<join column1> = {<table2>|<alias2>}.<join column2>;
/* select all the agents, and get the count of the properties that they are representing */

SELECT FirstName, LastName, COUNT(ListingID)
FROM Agents a LEFT OUTER JOIN CustAgentList c
  ON a.AgentID = c.AgentID
GROUP BY FirstName, LastName
ORDER BY LastName, FirstName;
Self JOINs

/* JOINing a table to itself is called a self join. Both INNER and OUTER JOIN statements are supported. Aliases are required. INNER JOIN Example: */

SELECT m.FirstName||' '||m.LastName AS Manager, e.FirstName||' '||e.LastName AS Employee
FROM EmpSelfJoin e INNER JOIN EmpSelfJoin m
ON e.BossID = m.EmployeeID
ORDER BY m.LastName, e.LastName;
FULL JOINs

/* FULL JOIN statements select all records from both tables, and match them on the join condition. Rarely seen in the wild. */

SELECT FirstName, LastName, COUNT(ListingID)
FROM Agents a FULL OUTER JOIN CustAgentList c
ON a.AgentID = c.AgentID
GROUP BY FirstName, LastName
ORDER BY LastName, FirstName;
FULL v. LEFT OUTER JOINs

Selected Rows  Excluded Rows

Selected Rows
Non-Equijoin Queries

/* join conditions do not have to be equalities – valid operators include BETWEEN, <, <=, >, >=, <> and !>. Example: */

SELECT e.LastName, e.HireDate FROM EmpSelfJoin e JOIN CompanyHistory h ON e.HireDate BETWEEN h.BeginDate AND h.EndDate;
Set operations

/* sometimes you want to combine the output from two or more queries into one result set. Operators: UNION, UNION ALL, MINUS, INTERSECT. */

<table>
<thead>
<tr>
<th>Set operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNION</td>
<td>Returns all unique rows retrieved by the queries.</td>
</tr>
<tr>
<td>UNION ALL</td>
<td>Returns all the rows retrieved by the queries, including any duplicate rows.</td>
</tr>
<tr>
<td>INTERSECT</td>
<td>Returns rows that are retrieved by both queries.</td>
</tr>
<tr>
<td>MINUS</td>
<td>Returns the rows that remain when the rows retrieved by a second query are removed (subtracted) from the rows retrieved by the first query.</td>
</tr>
</tbody>
</table>
Set Operations, Diagrammed

UNION

UNION ALL

INTERSECT

MINUS
Set Operators

/* Data types of each column must match. The number of columns must match. The first SELECT determines the column names. ORDER BY clauses go at the end. Syntax: */
SELECT <…>
{UNION | UNION ALL | INTERSECT | MINUS}
SELECT <…>
UNION Example

SELECT Table_Name AS "Object", 'Table' AS "Type"
FROM User_Tables
WHERE Table_Name NOT LIKE 'BIN%'
    UNION
SELECT Sequence_Name "Object", 'Sequence' AS "Type"
FROM User_Sequences
    UNION
SELECT Trigger_Name "Object", 'Trigger' AS "Type"
FROM User_Triggers
ORDER BY 2,1;
Union Example #2

**Employees**

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>EmployeeID</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam</td>
<td>Jones</td>
<td>950234567</td>
<td>M</td>
</tr>
<tr>
<td><strong>Sally</strong></td>
<td><strong>Smith</strong></td>
<td><strong>950345678</strong></td>
<td><strong>F</strong></td>
</tr>
<tr>
<td>Jim</td>
<td>Wilson</td>
<td>950456789</td>
<td>M</td>
</tr>
<tr>
<td>Mary</td>
<td>Jackson</td>
<td>950567890</td>
<td>F</td>
</tr>
</tbody>
</table>

**Students**

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>StudentID</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>Johnson</td>
<td>950234560</td>
<td>M</td>
</tr>
<tr>
<td>George</td>
<td>Yorke</td>
<td>950345679</td>
<td>M</td>
</tr>
<tr>
<td><strong>Heather</strong></td>
<td><strong>Baggins</strong></td>
<td><strong>950456788</strong></td>
<td><strong>F</strong></td>
</tr>
<tr>
<td>Mary</td>
<td>Thomas</td>
<td>950567897</td>
<td>F</td>
</tr>
</tbody>
</table>

SELECT First, Last, EmployeeID as ID
FROM Employees WHERE Gender = ‘F’
UNION
SELECT First, Last, StudentID
FROM Students WHERE Gender = ‘F’

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sally</strong></td>
<td>Smith</td>
<td>950345678</td>
</tr>
<tr>
<td>Mary</td>
<td>Jackson</td>
<td>950567890</td>
</tr>
<tr>
<td>Heather</td>
<td>Baggins</td>
<td>950456788</td>
</tr>
<tr>
<td>Mary</td>
<td>Thomas</td>
<td>950567897</td>
</tr>
</tbody>
</table>
Join Example (for comparison)

Employees

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>EmployeeID</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam</td>
<td>Jones</td>
<td>950234567</td>
<td>M</td>
</tr>
<tr>
<td>Sally</td>
<td>Smith</td>
<td>950345678</td>
<td>F</td>
</tr>
<tr>
<td>Jim</td>
<td>Wilson</td>
<td>950456789</td>
<td>M</td>
</tr>
<tr>
<td>Mary</td>
<td>Jackson</td>
<td>950567890</td>
<td>F</td>
</tr>
</tbody>
</table>

Classes

<table>
<thead>
<tr>
<th>ClassID</th>
<th>CRN</th>
<th>Instructor</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIS 210</td>
<td>44100</td>
<td>950345678</td>
<td>CS I</td>
</tr>
<tr>
<td>CIS 211</td>
<td>44101</td>
<td>950345678</td>
<td>CS II</td>
</tr>
<tr>
<td>CIS 212</td>
<td>44102</td>
<td>950234567</td>
<td>CS III</td>
</tr>
<tr>
<td>CIS 313</td>
<td>44103</td>
<td>950234567</td>
<td>Data S...</td>
</tr>
</tbody>
</table>

SELECT e.First, e.Last, c.ClassID, c.Name FROM Employees e
INNER JOIN Classes c ON e.EmployeeID = c.Instructor
WHERE EmployeeID = 950345678;

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>ClassID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sally</td>
<td>Smith</td>
<td>CIS 210</td>
<td>CS I</td>
</tr>
<tr>
<td>Mary</td>
<td>Jackson</td>
<td>CIS 211</td>
<td>CS II</td>
</tr>
</tbody>
</table>
MINUS example

/* we want the list of Orick residents who do not have properties for sale */

SELECT Address, City
FROM Customers
WHERE City = 'Orick'
MINUS
SELECT Address, City
FROM Properties
WHERE City = 'Orick';
INTERSECT example

/* we want the list of Orick residents who do have properties for sale */

SELECT Address, City
FROM Customers
WHERE City = 'Orick'
INTERSECT
SELECT Address, City
FROM Properties
WHERE City = 'Orick';
Subqueries

/* A subquery is a query that is contained within another SQL statement. Single-row example: */

/* select a list of agents who were hired by Redwood after Tobias Carling was hired */

SELECT FirstName, LastName FROM Agents
WHERE HireDate > (SELECT HireDate from Agents where FirstName = 'Tobias' and LastName = 'Carling');
Multiple-Row Subquery Example

/* Multiple-row subqueries can use the IN or NOT IN operators */

SELECT DISTINCT FirstName, LastName
FROM Agents
WHERE AgentID NOT IN (SELECT AgentID FROM CustAgentList);
ANY or ALL in Subqueries

/* Use the ANY or ALL operators with the comparison operators to compare a value with the values in a list. */

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=ANY</td>
<td>Equal to any value the subquery returns. It is the same as the IN operator.</td>
</tr>
<tr>
<td>&lt;ANY</td>
<td>Smaller than the largest value the subquery returns.</td>
</tr>
<tr>
<td>&gt;ANY</td>
<td>Larger than the smallest value the subquery returns.</td>
</tr>
<tr>
<td>&gt;ALL</td>
<td>Larger than the largest value the subquery returns.</td>
</tr>
<tr>
<td>&lt;ALL</td>
<td>Smaller than the smallest value the subquery returns.</td>
</tr>
</tbody>
</table>
ANY / ALL examples

/* list the agents that make more than everyone in team 10 */

SELECT AgentID, BaseSalary, Team FROM AgentsHR
WHERE BaseSalary > ALL (SELECT BaseSalary
FROM AgentsHR WHERE Team = 10);

/* list the agents that make more than someone in team 10 */

SELECT AgentID, BaseSalary, Team FROM AgentsHR
WHERE BaseSalary > ANY (SELECT BaseSalary
FROM AgentsHR WHERE Team = 10) AND Team <> 10;
Correlated Subqueries

/* Correlated subqueries reference one or more columns in the outer or containing query. Oracle executes the inner query once for each row in the outer query: O(n^2). */

SELECT * FROM AgentsHR outer WHERE outer.Vacation > (SELECT AVG(Vacation) FROM AgentsHR inner WHERE outer.Team = inner.Team) ORDER BY Team;
The EXISTS Operator

/* The EXISTS operator is used to determine whether a condition is true in a subquery (NOT EXISTS determines whether it is false). Syntax: */

```
SELECT <outer select list> FROM <outer table> outer WHERE EXISTS (SELECT ‘X’ from <inner-table> inner WHERE <outer-table.column-name> = <inner-table.column-name>);
```
EXISTS example

SELECT COUNT(*)
AS "Customers w/o agents"
FROM Customers outer
WHERE NOT EXISTS
  (SELECT 'X'
   FROM CustAgentList inner
   WHERE outer.CustomerID =
   inner.CustomerID);
Subqueries in DML Statements

/* delete the agents who do not have listings (yikes!) */

DELETE
FROM Agents
WHERE NOT EXISTS
(SELECT 'X' from Listings where AgentID = ListingAgentID);
Views

A VIEW is a predefined query on one or more *base tables*. It is a “virtual table”. Why? Because tables are normalized, and reports can be complex joins. From the user’s viewpoint, they behave the same as tables.
View Benefits:

Security: Views restrict user’s access to table rows and columns.

Query Simplification: A view which captures a complex join makes it easier to use.

Privacy: Views hide the base tables

Data Independence: If underlying tables have to be modified, then views can replace the old table definitions for backwards-compatibility
Simple Views

Syntax:

CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW <view-name>
[(<column-name>,...,<column-name>)] AS <subquery>
[WITH CHECK OPTION [CONSTRAINT <constraint-name>]] [WITH READ ONLY];
VIEW examples

CREATE OR REPLACE VIEW MaleEmployees AS SELECT * FROM Agents WHERE Gender = 'M' WITH READ ONLY;

CREATE OR REPLACE VIEW Employees AS SELECT AgentID, FirstName, LastName, Title FROM Agents;
Querying Views

CREATE OR REPLACE VIEW Employees AS
SELECT AgentID, FirstName, LastName, Title FROM Agents;

/* select the brokers */

SELECT LastName, FirstName, Title FROM Employees WHERE Title = 'Broker' ORDER BY 1;
Modifying Data using Views

WITH READ ONLY prevents updating, inserting or deleting rows in a view.
WITH CHECK OPTION prevents updating, inserting or deleting rows that violate the check condition.
All modifications have to honor the constraints of the underlying table.
Examples…
Complex Views

/* Views with GROUP BY or DISTINCT clause or one or more functions or expressions. DML operations are not allowed on some complex expressions. Example: */

CREATE OR REPLACE VIEW AgentView AS
SELECT AgentID, FirstName, LastName, BirthDate, (SYSDATE - BirthDate)/365.25 AS Age
FROM Agents;
CREATE OR REPLACE VIEW ForSale(AgID, AgentFirst, AgentLast, Bid, Ask, Addr, City, Br, Ba, Ft, SellerFirst, SellerLast) AS

FROM Agents a
INNER JOIN CustAgentList b ON a.AgentID = b.AgentID
INNER JOIN Listings c ON b.ListingID = c.ListingID
INNER JOIN Properties d ON c.PropertyID = d.PropertyID
INNER JOIN Customers e ON d.OwnerID = e.CustomerID
WHERE b.ContactReason = 'Sell';
Listing Views, Dropping Views

/* listing views */

SELECT view_name, text_length, text
FROM user_views
ORDER BY view_name;

/* drop a view */
DROP VIEW AgentView;