The University of Texas at Austin
McCombs School of Business

MIS 381 INTRO. TO DATABASE MANAGEMENT

Advanced SQL

Summary queries, subqueries, functions

Tayfun Keskin
Visiting Clinical Professor, The University of Texas at Austin, McCombs School of Business
Associate Teaching Professor, University of Washington Seattle, Foster School of Business

QUESTIONS

- Any questions before we begin?

AGENDA

Lecture

Summary Queries

Subqueries, Functions

Hands-On

Exercises

Looking Forward

Homework 4

PL/SQL

QUESTION

Let’s speculate on this student entity:

What interesting questions might we ask?

Student
StudentId(PK) StudentFirstName StudentLastName StudentStreet StudentCity StudentState StudentZip StudentGender StudentDoB Major Minor GPA CreditsCompleted DateEnrolled

POSSIBLE QUERIES

- How many students do we have in a particular major?
  - Minor?
  - Gender?
  - State?
  - City?
- What is the average GPA of students in a particular major?
- What are the total and average credits completed by major based on date enrolled?
  - Does this change by minor?
  - Gender?

Student
StudentId(PK) StudentFirstName StudentLastName StudentStreet StudentCity StudentState StudentZip StudentGender StudentDoB Major Minor GPA CreditsCompleted DateEnrolled

SELECT SYNTAX FOR SUMMARY DATA

SELECT	Columns - Column names - Arithmetic expressions - Literals (text or numeric) - Scalar functions - Column functions
FROM	Table or view names
WHERE.	Conditions (qualifies rows)
ORDER BY	Sorts result rows
GROUP BY	Creates sub totals with column functions

- How many students do we have in a particular major?

```
SELECT count(studentid), major
FROM student
GROUP BY major
```

- What is the average GPA of students in a particular major?

```
SELECT avg(GPA), major
FROM student
GROUP BY major
```

COMMON ORACLE COLUMN FUNCTIONS

SELECT	Columns - Column names - Arithmetic expressions - Literals (text or numeric) - Scalar functions - Column functions
FROM	Table or view names
WHERE.	Conditions (qualifies rows)
ORDER BY	Sorts result rows
GROUP BY	Creates sub totals with column functions

- AVG
- COUNT
- MAX
- MEDIAN
- MIN
- SUM
- **Also known as aggregate and analytic functions**

COMMON ERRORS

```
SELECT count(studentid), major, minor FROM student
GROUP BY major
```

```
ORA-00979: not a GROUP BY expression
00979. 00000 -  "not a GROUP BY expression"
*Cause:
*Action:
```

```
SELECT count(studentid), major, minor FROM student
GROUP BY major, minor
```

COMMON ERRORS

```
SELECT studentid, major FROM student
GROUP BY major
```

```
ORA-00979: not a GROUP BY expression
00979. 00000 -  "not a GROUP BY expression"
*Cause:
*Action:
```

```
SELECT count(studentid), major FROM student
GROUP BY major
```

HAVING CLAUSE

SELECT	Columns - Column names - Arithmetic expressions - Literals (text or numeric) - Scalar functions - "Aggregate" functions
FROM	Table or view names
WHERE.	Conditions (qualifies rows)
ORDER BY	Sorts result rows
GROUP BY	Creates sub totals with column functions
HAVING	Aggregate condition

- How many majors do we have where the average GPA is less than 3.0? What is the count of student is such majors?

```
SELECT count(studentid), major
FROM student
GROUP BY major
HAVING (avg(GPA) < 3)
```

- How many students do we have by major with a GPA < 3

```
SELECT count(studentid), major
FROM student
WHERE GPA < 3
GROUP BY major
```

LOOKING FORWARD

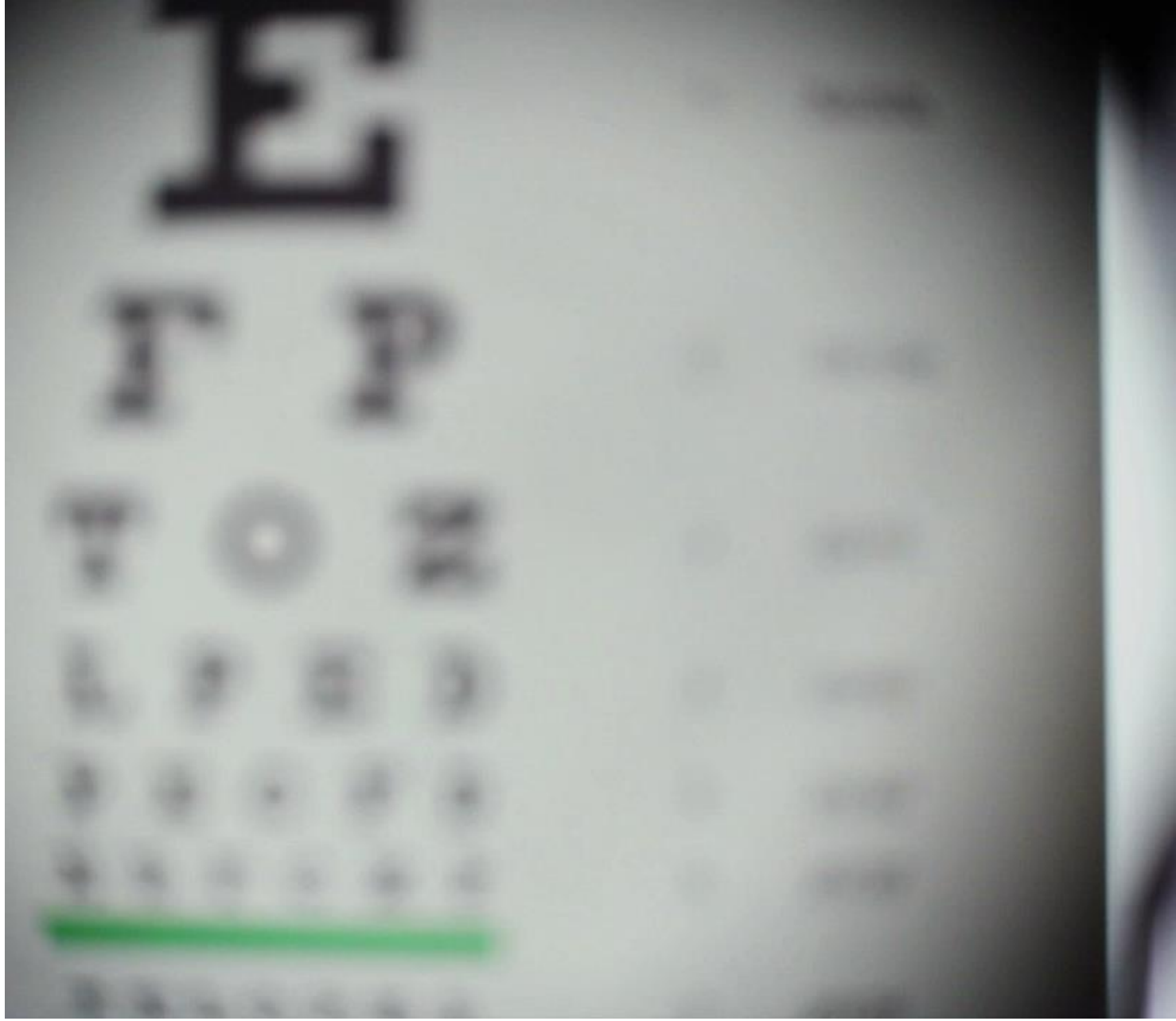
Read Chapters:

- 3: Single table queries
- 4: Multiple table (Joins)
- 7: DML (insert, update…)
- 5: Summary queries
- 6: Subqueries
- 8: Data types and functions

Homework 4, Quiz 4

HBSP Package

Exam 2



THANK YOU

PART 2:
SUBQUERIES AND FUNCTIONS

REMINDER QUESTION

What is a subquery?

WHAT IS A SUBQUERY?

- Sometime the logic needed in a SELECT statement requires another SELECT statement – this second SELECT statement is called a subquery
- The syntax for a subquery is the same as the syntax for the SELECT statement
- A subquery can return a single value, a result set that contain a single column, or a result set that contains one or more columns
- The first SELECT statement is called the outer query

SUBQUERIES CAN BE PART OF …

- … a WHERE clause as a search condition
- … a HAVING clause as a search condition
- … the FROM clause as a table specification
- … a SELECT clause as a column specification

SUBQUERIES CAN BE PART OF …

- … a WHERE clause as a search condition
- … a HAVING clause as a search condition
  - When a subquery returns a single value, it can be used anywhere an expression is evaluated
  - When a subquery returns a single-column result set with two or more rows, it can be used in a place of a list of values, such as the list for an IN operator

SUBQUERIES CAN BE PART OF …

- … the FROM clause as a table specification
  - When a subquery returns a result set with two or more columns, it can be used in a FROM clause
  - A subquery coded in a FROM clause is called an inline view
  - An inline view should have an alias
  - All calculated values should have a name (alias)
  - Inline views are not as efficient as views

SUBQUERIES CAN BE PART OF …

- … a SELECT clause as a column specification
  - A subquery must return a single value
  - Usually a correlated subquery (i.e., a query that is executed once for each row processed by the outer query)
  - Such subqueries can usually be restated as joins

SUBQUERIES CAN BE PART OF …

- … while syntactically correct subqueries are not used in the GROUP BY and ORDER BY clauses
- … since joins are usually more efficient than subqueries, they are rarely used in a SELECT clause

JOINS VS. SUBQUERIES

- **Advantages of Joins:**
  - A join can include columns from both tables.
  - A join is more intuitive when it uses an existing relationship.
- **Advantages of Subqueries:**
  - A subquery can pass an aggregate value to the outer query.
  - A subquery is more intuitive when it uses an ad hoc relationship.
  - Long, complex queries can be easier to code with subqueries.

PROCEDURE FOR BUILDING COMPLEX QUERIES

- State the problem to be solved in English.
- Use pseudocode to outline the query.
- If necessary, use pseudocode to outline each subquery.
- Code the subqueries and test them.
- Code and test the final query.

LOOKING FORWARD

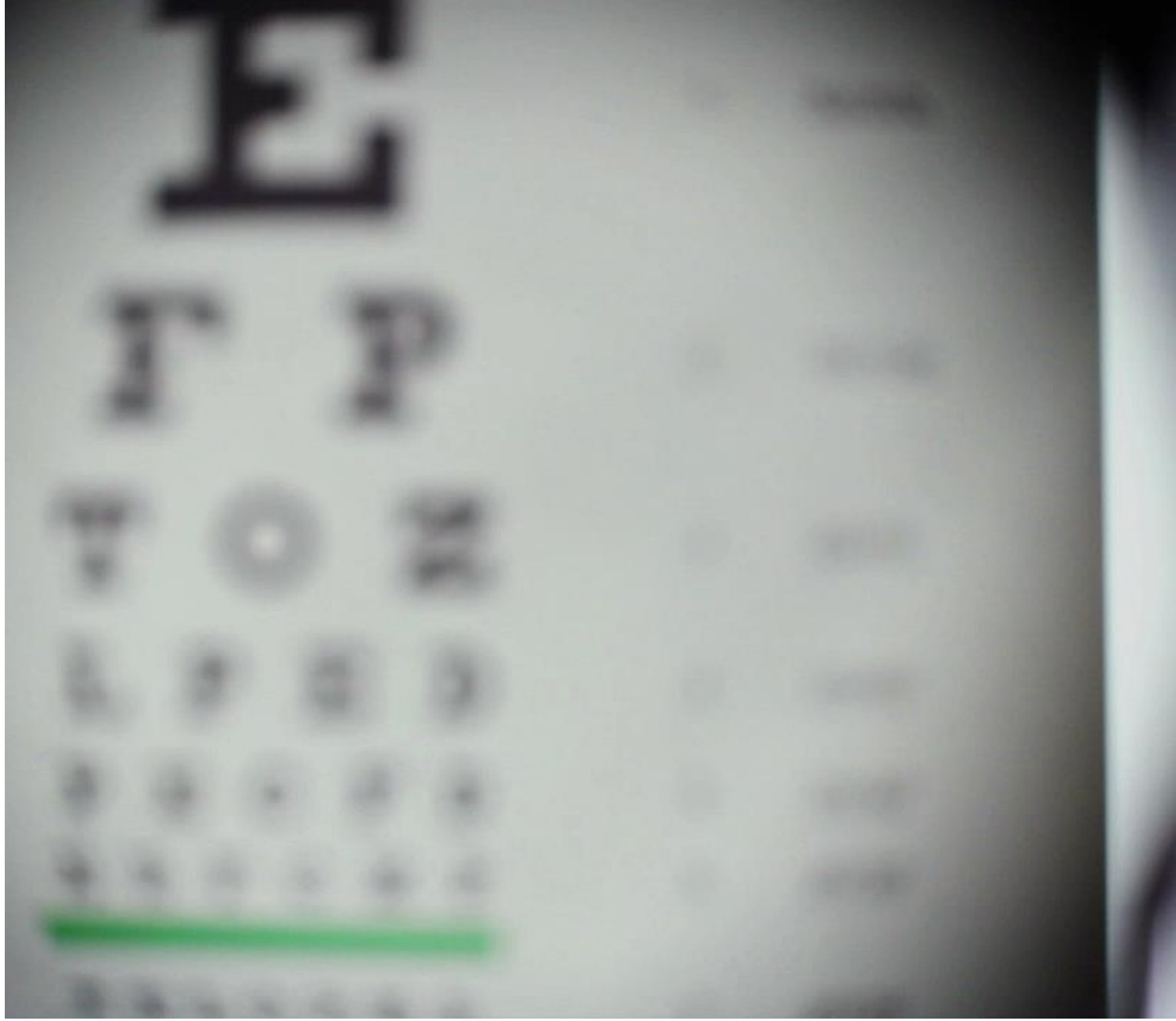
Read Chapters:

- 3: Single table queries
- 4: Multiple table (Joins)
- 7: DML (insert, update…)
- 5: Summary queries
- 6: Subqueries
- 8: Data types and functions

Homework 4, Quiz 4

HBSP Package

Exam 2



THANK YOU

BACKUP SLIDES

PART 1

Chapter 5

Summary Queries

AGGREGATE FUNCTIONS

The syntax of the aggregate functions

```
AVG([ALL|DISTINCT] expression)
SUM([ALL|DISTINCT] expression)
MIN([ALL|DISTINCT] expression)
MAX([ALL|DISTINCT] expression)
COUNT([ALL|DISTINCT] expression)
COUNT(*)
```

A summary query example

```
SELECT COUNT(*) AS number_of_invoices,
       SUM(invoice_total) AS sum_of_invoice_totals
FROM   invoices;
```

The result set

NUMBER_OF_RECORDS	Invoice Total Sum
114	214290.51

Practice:

Write a query that returns the following:

- Count of invoice records
- Sum of invoice_total
- Average invoice_total
- Lowest invoice_total
- Largest invoice_total
- Count of distinct vendors
- Should we clean up any formatting data that you returned in query?

A summary query with COUNT(*), AVG, and SUM

```
SELECT 'After 1/1/2008' AS selection_date,
    COUNT(*) AS number_of_invoices,
    ROUND(AVG(invoice_total), 2) AS avg_invoice_amt,
    SUM(invoice_total) AS total_invoice_amt
FROM invoices
WHERE invoice_date > '01-JAN-2014'
```

The result set

	SELECTION_DATE	NUMBER_OF_INVOICES	AVG_INVOICE_AMT	TOTAL_INVOICE_AMT
1	After 1/1/2014	114	1879.74	214290.51

A summary query with MIN and MAX functions

```
SELECT 'After 1/1/2008' AS selection_date,
    COUNT(*) AS number_of_invoices,
    MAX(invoice_total) AS highest_invoice_total,
    MIN(invoice_total) AS lowest_invoice_total
FROM invoices
WHERE invoice_date > '01-JAN-2014'
```

The result set

	SELECTION_DATE	NUMBER_OF_INVOICES	HIGHEST_INVOICE_TOTAL	LOWEST_INVOICE_TOTAL
1	After 1/1/2014	114	37966.19	6

A summary query example

```
SELECT    COUNT(*) AS number_of_invoices,
          SUM(invoice_total) AS sum_of_invoice_totals
FROM invoices;
```

Practice aggregating an arithmetic expression:

1. Update the above query to pull the SUM of what's due.

   NOTE: The "amount due" = (invoice_total – payment_total – credit_total)

2. Update query to only consider invoices created on/after a certain date

   Hint: Dates follow a "DD-MMM-YYYY" format (e.g. 07-Aug-2018)

A summary query with the DISTINCT keyword

```
SELECT COUNT(DISTINCT vendor_id) AS number_of_vendors,
    COUNT(vendor_id) AS number_of_invoices,
    ROUND(AVG(invoice_total),2) AS avg_invoice_amt,
    SUM(invoice_total) AS total_invoice_amt
FROM invoices
WHERE invoice_date > '01-JAN-2008'
```

The result set

	NUMBER_OF_VENDORS	NUMBER_OF_INVOICES	AVG_INVOICE_AMT	TOTAL_INVOICE_AMT
1	34	114	1879.74	214290.51

GROUPING

The syntax with GROUP BY and HAVING clauses

```
SELECT select_list
FROM table_source
[WHERE search_condition]  --row filter
[GROUP BY group_by_list]
[HAVING search_condition]  --aggregate filter
[ORDER BY order_by_list]
```

A summary query that counts the number of invoices by vendor

```
SELECT vendor_id, COUNT(*) AS invoice_qty
FROM invoices
GROUP BY vendor_id
ORDER BY vendor_id
```

The result set

	VENDOR_ID	INVOICE_QTY
1	34	2
2	37	3
3	48	1
4	72	2

(34 rows selected)

First, let's look at Excel first…

A summary query that calculates average invoice amount by vendor

```
SELECT vendor_id, vendor_name,
  ROUND(AVG(invoice_total), 2) AS average_invoice_amount
FROM invoices
GROUP BY vendor_id, vendor_name
ORDER BY average_invoice_amount DESC
```

The result set: (34 records)

	VENDOR_ID	AVERAGE_INVOICE_AMOUNT
1	110	23978.48
2	72	10963.66
3	104	7125.34
4	99	6940.25
5	119	4901.26
6	122	2575.33
7	86	2433
8	100	2184.5
9	113	1750
10	107	1600
11	103	1367.5
12	83	1077.21
13	81	936.93

HAVING

A summary query that calculates average invoice amount by vendor

```
SELECT vendor_id,
  ROUND(AVG(invoice_total), 2) AS average_invoice_amount
FROM invoices
GROUP BY vendor_id
HAVING ROUND(AVG(invoice_total), 2) > 2000
ORDER BY average_invoice_amount DESC
```

The result set: (8 records)

	VENDOR_ID	AVERAGE_INVOICE_AMOUNT
1	110	23978.48
2	72	10963.66
3	104	7125.34
4	99	6940.25
5	119	4901.26
6	122	2575.33
7	86	2433
8	100	2184.5

Practice:

1. Write a query that returns the vendor state and the count of vendors in that state
2. Update query to only show states with more than 2 vendors
3. Add in City between state and count
4. Sort by Count DESC

A summary query with a join

```
SELECT vendor_state, vendor_city,
    COUNT(*) AS invoice_qty,
    ROUND(AVG(invoice_total),2) AS invoice_avg
FROM invoices JOIN vendors
    ON invoices.vendor_id = vendors.vendor_id
GROUP BY vendor_state, vendor_city
ORDER BY vendor_state, vendor_city
```

The result set

	VENDOR_STATE	VENDOR_CITY	INVOICE_QTY	INVOICE_AVG
1	AZ	Phoenix	1	662
2	CA	Fresno	19	1208.75
3	CA	Los Angeles	1	503.2
4	CA	Oxnard	3	188

(20 rows selected)

A summary query that limits the groups to those with two or more invoices

```
SELECT vendor_state, vendor_city,
    COUNT(*) AS invoice_qty,
    ROUND(AVG(invoice_total),2) AS invoice_avg
FROM invoices JOIN vendors
    ON invoices.vendor_id = vendors.vendor_id
GROUP BY vendor_state, vendor_city
HAVING COUNT(*) >= 2
ORDER BY vendor_state, vendor_city
```

The result set

	VENDOR_STATE	VENDOR_CITY	INVOICE_QTY	INVOICE_AVG
1	CA	Fresno	19	1208.75
2	CA	Oxnard	3	188
3	CA	Pasadena	5	196.12
4	CA	Sacramento	7	253

(12 rows selected)

HAVING vs WHERE

HAVING vs WHERE?

Summary query with filter condition in the HAVING clause

```
SELECT vendor_name, COUNT(*) AS invoice_qty,
    ROUND(AVG(invoice_total),2) AS invoice_avg
FROM vendors JOIN invoices
    ON vendors.vendor_id = invoices.vendor_id
GROUP BY vendor_name
HAVING AVG(invoice_total) > 500
ORDER BY invoice_qty DESC
```

The result set

	VENDOR_NAME	INVOICE_QTY	INVOICE_AVG
1	United Parcel Service	9	2575.33
2	Zylka Design	8	867.53
3	Malloy Lithographing Inc	5	23978.48
4	IBM	2	600.06

Summary query with filter condition in the WHERE clause

```
SELECT vendor_name, COUNT(*) AS invoice_qty,
    ROUND(AVG(invoice_total),2) AS invoice_avg
FROM vendors JOIN invoices
    ON vendors.vendor_id = invoices.vendor_id
WHERE invoice_total > 500 (row level)
GROUP BY vendor_name
ORDER BY invoice_qty DESC
```

The result set

	VENDOR_NAME	INVOICE_QTY	INVOICE_AVG
1	United Parcel Service	9	2575.33
2	Zylka Design	7	946.67
3	Malloy Lithographing Inc	5	23978.48
4	Ingram	2	1077.21

HAVING vs WHERE?

```
Select *
From invoices
where vendor_id in (
  select vendor_id
  from vendors
  where vendor_name like 'Zylka%‘
)
```

	INVOICE_ID	VENDOR_ID	INVOICE_NUMBER	INVOICE_DATE	INVOICE_TOTAL	PAYMENT_TOTAL	CREDIT_TOTAL	TERMS_ID	INVOICE_DUE_DATE	PAYMENT_DATE
1	15	121	97/553B	26-APR-14	313.55	0	0	4	09-JUL-14	(null)
2	18	121	97/553	27-APR-14	904.14	0	0	4	09-JUL-14	(null)
3	19	121	97/522	30-APR-14	1962.13	0	200	4	10-JUL-14	(null)
4	20	121	97/503	30-APR-14	639.77	639.77	0	4	11-JUN-14	05-JUN-14
5	21	121	97/488	30-APR-14	601.95	601.95	0	3	03-JUN-14	27-MAY-14
6	22	121	97/486	30-APR-14	953.1	953.1	0	2	21-MAY-14	13-MAY-14
7	23	121	97/465	01-MAY-14	565.15	565.15	0	1	14-MAY-14	05-MAY-14
8	24	121	97/222	01-MAY-14	1000.46	1000.46	0	3	03-JUN-14	25-MAY-14

A summary query with a compound condition in the HAVING clause

```
SELECT
   invoice_date,
   COUNT(*) AS invoice_qty,
   SUM(invoice_total) AS invoice_sum
FROM invoices
GROUP BY invoice_date
HAVING invoice_date
        BETWEEN '01-MAY-2014' AND '31-MAY-2014'
   AND COUNT(*) > 1
   AND SUM(invoice_total) > 100
ORDER BY invoice_date DESC
```

The result set

	INVOICE_DATE	INVOICE_QTY	INVOICE_SUM
1	31-MAY-14	3	11557.75
2	23-MAY-14	6	2761.17
3	22-MAY-14	2	442.5
4	20-MAY-14	3	308.64

(15 rows selected)

The same query with a WHERE clause

```
SELECT
   invoice_date,
   COUNT(*) AS invoice_qty,
   SUM(invoice_total) AS invoice_sum
FROM invoices
WHERE invoice_date
    BETWEEN '01-MAY-2014' AND '31-MAY-2014'
GROUP BY invoice_date
HAVING COUNT(*) > 1
   AND SUM(invoice_total) > 100
ORDER BY invoice_date DESC
```

TIP:

Key aggregate filters in HAVING because you have to.

Keep row-level filters in WHERE for readability

Practice:

- Pull all invoices after June 1 and give the average invoice_total by vendor_state. Only show states with avg > 2000.

Follow these steps:

1. Select * From table…
2. Code the JOIN (if applicable)
3. Code WHERE
4. Specify the columns and add aggregate functions AND group by
5. Code HAVING filter aggregate columns

ROLLUP / CUBE

A summary query with a final summary row

```
SELECT vendor_id, COUNT(*) AS invoice_count,
    SUM(invoice_total) AS invoice_total
FROM invoices
GROUP BY ROLLUP(vendor_id)
```

The result set

	VENDOR_ID	INVOICE_COUNT	INVOICE_TOTAL
32	121	8	6940.25
33	122	9	23177.96
34	123	47	4378.02
35	(null)	114	214290.51

(35 rows selected)

Practice:

- Pull vendor state and count of vendors like before but use a ROLLUP keyword in group by
- Try adding in an additional column like city into the select and rollup
- Try updating ROLLUP to CUBE

A summary query with a summary row at the start of the result set

```
SELECT vendor_id, COUNT(*) AS invoice_count,
    SUM(invoice_total) AS invoice_total
FROM invoices
GROUP BY CUBE(vendor_id)
```

The result set

	VENDOR_ID	INVOICE_COUNT	INVOICE_TOTAL
1	(null)	114	214290.51
2	34	2	1200.12
3	37	3	564
4	48	1	856.92

(35 rows selected)

A summary query with a summary row for each set of groups

```
SELECT vendor_state, vendor_city, COUNT(*) AS qty_vendors
FROM vendors
WHERE vendor_state IN ('IA', 'NJ')
GROUP BY CUBE(vendor_state, vendor_city)
ORDER BY vendor_state, vendor_city
```

The result set

	VENDOR_STATE	VENDOR_CITY	QTY_VENDORS
1	IA	Fairfield	1
2	IA	Washington	1
3	IA	(null)	2
4	NJ	East Brunswick	2
5	NJ	Fairfield	1
6	NJ	Washington	1
7	NJ	(null)	4
8	(null)	East Brunswick	2
9	(null)	Fairfield	2
10	(null)	Washington	2
11	(null)	(null)	6

A summary query for non-numeric columns

```
SELECT MIN(vendor_name) AS first_vendor,
    MAX(vendor_name) AS last_vendor,
    COUNT(vendor_name) AS number_of_vendors
FROM vendors
```

The result set

	FIRST_VENDOR	LAST_VENDOR	NUMBER_OF_VENDORS
1	ASC Signs	Zylka Design	122

Practice:

1. Pull the Last Name in vendor_contacts closest to A (e.g., Adams). Pull the Last Name closest to Z (e.g., Zilker)
2. *Pop Quiz – Do you think we can pull the AVG of last_name?*

PART 2

Chapter 6

Subqueries

4 ways to use a subquery in a SELECT statement

- In a WHERE clause as a search condition
- In a HAVING clause as a search condition
- In the FROM clause as a table specification
- In the SELECT clause as a column specification

Subquery in WHERE

Basic use case: Use a Subquery in the WHERE clause

Part 1: Write subquery

```
SELECT AVG(invoice_total)
FROM invoices
```

Value returned by query

```
1879.7413
```

Part 2. Use query as a subquery in a WHERE clause

```
SELECT *
FROM invoices
WHERE invoice_total >
    (SELECT AVG(invoice_total)
     FROM invoices)
ORDER BY invoice_total
```

The result set

	INVOICE_NUMBER	INVOICE_DATE	INVOICE_TOTAL
1	989319-487	18-APR-14	1927.54
2	97/522	30-APR-14	1962.13
3	989319-417	26-APR-14	2051.59
4	989319-427	25-APR-14	2115.81
5	989319-477	19-APR-14	2184.11

A query that uses an inner join

```
SELECT invoice_number, invoice_date, invoice_total
FROM invoices JOIN vendors
    ON invoices.vendor_id = vendors.vendor_id
WHERE vendor_state = 'CA'
ORDER BY invoice_date
```

The result set

	INVOICE_NUMBER	INVOICE_DATE	INVOICE_TOTAL
1	QP58872	25-FEB-14	116.54
2	Q545443	14-MAR-14	1083.58
3	MABO1489	16-APR-14	936.93
4	97/553B	26-APR-14	313.55

(40 rows)

The same query but with a subquery

```
SELECT invoice_number, invoice_date, invoice_total
FROM invoices
WHERE vendor_id IN
    (SELECT vendor_id
    FROM vendors
    WHERE vendor_state = 'CA')
ORDER BY invoice_date
```

Advantages of joins

- A join can include columns from both tables.
- A join is more intuitive when it uses an existing relationship.

Advantages of subqueries

- A subquery can pass an aggregate value to the outer query.
- A subquery is more intuitive when it uses an ad hoc relationship.
- Long, complex queries can be easier to code with subqueries.

Build first query

```
SELECT vendor_id
FROM vendors
where vendor_name like 'B%'
```

Subquery Results

VENDOR_ID
8
11
17
37
47
51
67
84
99

Final Query w/ Subquery

```
SELECT *
from invoices
WHERE vendor_id IN (
   SELECT vendor_id
   FROM vendors
   where vendor_name like 'B%')
```

Same as the following

```
SELECT *
from invoices
WHERE vendor_id IN
(8,11,17,37,47,51,67,84,99)
```

A procedure for building complex queries

- State the problem to be solved in English.
- Use pseudocode to outline your first query.
- If necessary, use pseudocode to outline each subquery.
- Code the subqueries and test them individually
- Combine inner queries with outer queries and test the final query.

ANY, SOME, ALL

Keywords

The syntax of a subquery in WHERE

```
Select column_list
From table
WHERE expression comparison_operator
 [SOME|ANY|ALL] (subquery)
```

Example

```
SELECT *
from invoices
WHERE vendor_id IN (
  SELECT vendor_id
  FROM vendors
  where vendor_name like 'B%')
```

The syntax of a WHERE clause that uses an IN phrase with a subquery

```
WHERE test_expression [NOT] IN (subquery)
```

A query that returns vendors without invoices

```
SELECT vendor_id, vendor_name, vendor_state
FROM vendors
WHERE vendor_id NOT IN
     (SELECT DISTINCT vendor_id
     FROM invoices)
ORDER BY vendor_id
```

How would you do this another way?

```
SELECT v.vendor_id, vendor_name, vendor_state
FROM vendors v LEFT JOIN invoices i
    ON v.vendor_id = i.vendor_id
WHERE i.vendor_id IS NULL
ORDER BY v.vendor_id
```

The result of the subquery

	VENDOR_ID
1	34
2	37
3	48
4	72
5	80
6	81

(34 rows)

The result set

	VENDOR_ID	VENDOR_NAME	VENDOR_STATE
32	33	Nielson	OH
33	35	Cal State Termite	CA
34	36	Graylift	CA
35	38	Venture Communications Int'l	NY
36	39	Custom Printing Company	MO
37	40	Nat Assoc of College Stores	OH

(88 rows)

How the ALL keyword works

Condition	Equivalent expression
`x > ALL (1, 2)`	`x > 2`
`x < ALL (1, 2)`	`x < 1`
`x = ALL (1, 2)`	`(x = 1) AND (x = 2)`
`x <> ALL (1, 2)`	`(x <> 1) AND (x <> 2)`

A procedure for building complex queries

- State the problem to be solved in English.
- Use pseudocode to outline the query.
- If necessary, use pseudocode to outline each subquery.
- Code the subqueries and test them individually
- Combine inner queries with outer queries and test the final query.

```
select vendor_id
from vendors
where vendor_name = 'IBM'
```

```
Select invoice_total
from invoices
where vendor_id in (
      select vendor_id
      from vendors
      where vendor_name = 'IBM')
```

```
select *
from invoices
where invoice_total > ALL (
    Select invoice_total from invoices
    where vendor_id in (
        select vendor_id
        from vendors
        where vendor_name = 'IBM')
    );
```

Query that uses join and ALL

```
SELECT vendor_name, invoice_number, invoice_total
FROM invoices i JOIN vendors v
    ON i.vendor_id = v.vendor_id
WHERE invoice_total > ALL
    (SELECT invoice_total
    FROM invoices
    WHERE vendor_id = 34)
ORDER BY vendor_name
```

The result of the subquery

	INVOICE_TOTAL
1	116.54
2	1083.58

The result set

	VENDOR_NAME	INVOICE_NUMBER	INVOICE_TOTAL
1	Bertelsmann Industry Svcs. Inc	509786	6940.25
2	Cahners Publishing Company	587056	2184.5
3	Computerworld	367447	2433
4	Data Reproductions Corp	40318	21842

(25 rows)

How the ANY and SOME keywords work

Condition	Equivalent expression
`x > ANY (1, 2)`	`x > 1`
`x < ANY (1, 2)`	`x < 2`
`x = ANY (1, 2)`	`(x = 1) OR (x = 2)`
`x <> ANY (1, 2)`	`(x <> 1) OR (x <> 2)`

How the ANY and SOME keywords work

Condition	Equivalent expression
`x > ANY (1, 2)`	`x > 1`
`x < ANY (1, 2)`	`x < 2`
`x = ANY (1, 2)`	`(x = 1) OR (x = 2)`
`x <> ANY (1, 2)`	`(x <> 1) OR (x <> 2)`

Practice:

- Pull all the invoices that had invoice totals bigger than ANY of IBM's invoices

Subquery in FROM

When is it useful to select FROM a query?

- **When you want to aggregate an aggregate value**
- e.g. You want to find the average count of vendors that sent us more than 1 invoice.
  - First you need to find the count of invoices for each vendor
  - Then you need to filter out vendors with a count of 1
  - Then you can find the average of the counts remaining
- Example that won't work

```
SELECT VENDOR_ID, avg(count(invoice_id))
FROM INVOICES
GROUP BY VENDOR_ID….
```

```
ORA-00935: group function is nested too deeply
00935. 00000 -  "group function is nested too deeply"
```

A query that uses an inline view

```
SELECT i.vendor_id,
    MAX(invoice_date) AS last_invoice_date,
    AVG(invoice_total) AS average_invoice_total
FROM invoices i JOIN
    (
    SELECT vendor_id,
        AVG(invoice_total) AS average_invoice_total
    FROM invoices
    HAVING AVG(invoice_total) > 4900
    GROUP BY vendor_id
    ) v
    ON i.vendor_id = v.vendor_id
GROUP BY i.vendor_id
ORDER BY MAX(invoice_date) DESC
```

NOTE: You technically don't need a subquery to do this. Just review this as a syntax example of how to join a table to query

The result of the subquery (an inline view)

	VENDOR_ID	AVERAGE_INVOICE_TOTAL
1	72	10963.655
2	99	6940.25
3	104	7125.34
4	110	23978.482
5	119	4901.26

The result set

	VENDOR_ID	LAST_INVOICE_DATE	AVERAGE_INVOICE_TOTAL
1	72	18-JUL-14	10963.655
2	119	04-JUN-14	4901.26
3	104	03-JUN-14	7125.34
4	99	31-MAY-14	6940.25
5	110	08-MAY-14	23978.482

COMPLEX QUERIES

Problem to solve:

Retrieve all vendor_ids, their most recent invoice_date, and average invoice_total for all vendors that have AVG invoice total great than $4900

How?

Join a **table** to a **select statement**

INVOICE_ID	VENDOR_ID	INVOICE_NUMBER	INVOICE_DATE	INVOICE_TOTAL
1	34	QP58872	25-FEB-14	116.54
2	34	Q545443	14-MAR-14	1083.58
3	110	P-0608	11-APR-14	20551.18
4	110	P-0259	16-APR-14	26881.4
5	81	MABO1489	16-APR-14	936.93
6	122	989319-497	17-APR-14	2312.2
7	82	C73-24	17-APR-14	600
8	122	989319-487	18-APR-14	1927.54
9	122	989319-477	19-APR-14	2184.11
10	122	989319-467	24-APR-14	2318.03
11	122	989319-457	24-APR-14	3813.33
12	122	989319-447	24-APR-14	3689.99
13	122	989319-437	24-APR-14	2765.36
14	122	989319-427	25-APR-14	2115.81

VENDOR_ID	AVERAGE_INVOICE_TOTAL
72	10963.655
99	6940.25
104	7125.34
110	23978.482
119	4901.26

When a *in line* subquery is needed (i.e. in FROM)

- Pull the vendor in each state with highest invoice total (Vendor_Name, State, Invoice Total)
- What do we need to first?

STEP 1 – Code first subquery

```
SELECT      v.vendor_id,
            v.vendor_name,
            v.vendor_state,
            SUM(i.invoice_total) AS sum_of_invoices
FROM invoices i JOIN vendors v ON i.vendor_id = v.vendor_id
GROUP BY v.vendor_id, v.vendor_name, v.vendor_state
order by vendor_state
```

VENDOR_ID	VENDOR_NAME	VENDOR_STATE	SUM_OF_INVOICES
96	Wells Fargo Bank	AZ	662
94	Abbey Office Furnishings	CA	17.5
99	Bertelsmann Industry Svcs. Inc	CA	6940.25
37	Blue Cross	CA	564
102	Coffee Break Service	CA	41.8
86	Computerworld	CA	2433
104	Digital Dreamworks	CA	7125.34
105	Dristas Groom & McCormick	CA	220
89	Evans Executone Inc	CA	95
106	Ford Motor Credit Company	CA	503.2
107	Franchise Tax Board	CA	1600
48	Fresno County Tax Collector	CA	856.92
108	Gostanian General Building	CA	450
34	IBM	CA	1200.12

STEP 2 – Move subquery into outer query

```
SELECT *
FROM (SELECT      v.vendor_id,
                  v.vendor_name,
                  v.vendor_state,
                  SUM(i.invoice_total) AS sum_of_invoices
FROM invoices i JOIN vendors v ON i.vendor_id = v.vendor_id
GROUP BY v.vendor_id, v.vendor_name, v.vendor_state
order by vendor_state)
```

VENDOR_ID	VENDOR_NAME	VENDOR_STATE	SUM_OF_INVOICES
96	Wells Fargo Bank	AZ	662
94	Abbey Office Furnishings	CA	17.5
99	Bertelsmann Industry Svcs. Inc	CA	6940.25
37	Blue Cross	CA	564
102	Coffee Break Service	CA	41.8
86	Computerworld	CA	2433
104	Digital Dreamworks	CA	7125.34
105	Dristas Groom & McCormick	CA	220
89	Evans Executone Inc	CA	95
106	Ford Motor Credit Company	CA	503.2
107	Franchise Tax Board	CA	1600
48	Fresno County Tax Collector	CA	856.92
108	Gostanian General Building	CA	450

When a *in line* subquery is needed (i.e. in FROM)

- Pull the vendor in each state with highest invoice total (Vendor_Name, State, Invoice Total)
- What do we need to first?

STEP 1 – Code first subquery

```
SELECT      v.vendor_id,
            v.vendor_name,
            v.vendor_state,
            SUM(i.invoice_total) AS sum_of_invoices
FROM invoices i JOIN vendors v ON i.vendor_id = v.vendor_id
GROUP BY v.vendor_id, v.vendor_name, v.vendor_state
order by vendor_state
```

VENDOR_ID	VENDOR_NAME	VENDOR_STATE	SUM_OF_INVOICES
96	Wells Fargo Bank	AZ	662
94	Abbey Office Furnishings	CA	17.5
99	Bertelsmann Industry Svcs. Inc	CA	6940.25
37	Blue Cross	CA	564
102	Coffee Break Service	CA	41.8
86	Computerworld	CA	2433
104	Digital Dreamworks	CA	7125.34
105	Dristas Groom & McCormick	CA	220
89	Evans Executone Inc	CA	95
106	Ford Motor Credit Company	CA	503.2
107	Franchise Tax Board	CA	1600
48	Fresno County Tax Collector	CA	856.92
108	Gostanian General Building	CA	450
34	IBM	CA	1200.12

STEP 2 – Move subquery into outer query

```
SELECT vendor_state, max(sum_of_invoices) as invoice_total
FROM (SELECT      v.vendor_id,
                  v.vendor_name,
                  v.vendor_state,
                  SUM(i.invoice_total) AS sum_of_invoices
FROM invoices i JOIN vendors v ON i.vendor_id = v.vendor_id
GROUP BY v.vendor_id, v.vendor_name, v.vendor_state
order by vendor_state)
Group by vendor_state
```

VENDOR_STATE	INVOICE_TOTAL
CA	7125.34
MA	1367.5
OH	207.78
TN	4378.02
MI	119892.41
DC	600
NV	23177.96
TX	2154.42
AZ	662
PA	265.36

When a *in line* subquery is needed (i.e. in FROM)

STEP 3 – Join both query outputs together like they're tables

```
SELECT * FROM
(SELECT     v.vendor_id,
            v.vendor_name,
            v.vendor_state,
            SUM(i.invoice_total) AS sum_of_invoices
FROM invoices i JOIN vendors v ON i.vendor_id = v.vendor_id
GROUP BY v.vendor_id, v.vendor_name, v.vendor_state
order by vendor_state)

(select vendor_state, max(sum_of_invoices) as invoice_total
from
(SELECT  v.vendor_id,
            v.vendor_state,
            SUM(i.invoice_total) AS sum_of_invoices
      FROM invoices i JOIN vendors v ON i.vendor_id = v.vendor_id
      GROUP BY v.vendor_id, v.vendor_state
      order by v.vendor_state)
group by vendor_state) s
```

VENDOR_ID	VENDOR_NAME	VENDOR_STATE	SUM_OF_INVOICES	VENDOR_STATE_1	INVOICE_TOTAL
104	Digital Dreamworks	CA	7125.34	CA	7125.34
103	Dean Witter Reynolds	MA	1367.5	MA	1367.5
88	Edward Data Services	OH	207.78	OH	207.78
123	Federal Express Corporation	TN	4378.02	TN	4378.02
110	Malloy Lithographing Inc	MI	119892.41	MI	119892.41
82	Reiter's Scientific & Pro Books	DC	600	DC	600
122	United Parcel Service	NV	23177.96	NV	23177.96
83	Ingram	TX	2154.42	TX	2154.42
96	Wells Fargo Bank	AZ	662	AZ	662
80	Cardinal Business Media, Inc.	PA	265.36	PA	265.36

When a *in line* subquery is needed (i.e. in FROM)

STEP 4 – Update your final select LAST

```
SELECT select summary_1.vendor_name, summary_1.vendor_state, invoice_total FROM
(SELECT     v.vendor_id,
            v.vendor_name,
            v.vendor_state,
            SUM(i.invoice_total) AS sum_of_invoices
FROM invoices i JOIN vendors v ON i.vendor_id = v.vendor_id
GROUP BY v.vendor_id, v.vendor_name, v.vendor_state
order by vendor_state) summary_1
      INNER JOIN
(select vendor_state, max(sum_of_invoices) as invoice_total
from
(SELECT  v.vendor_id,
            v.vendor_state,
            SUM(i.invoice_total) AS sum_of_invoices
        FROM invoices i JOIN vendors v ON i.vendor_id = v.vendor_id
        GROUP BY v.vendor_id, v.vendor_state
        order by v.vendor_state)
group by vendor_state) summary_2
    ON summary_1.vendor_state = summary_2.vendor_state
    and summary_1.sum_of_invoices = summary_2.invoice_total
Order by summary_1.vendor_state
```

VENDOR_NAME	VENDOR_STATE	INVOICE_TOTAL
Wells Fargo Bank	AZ	662
Digital Dreamworks	CA	7125.34
Reiter's Scientific & Pro Books	DC	600
Dean Witter Reynolds	MA	1367.5
Malloy Lithographing Inc	MI	119892.41
United Parcel Service	NV	23177.96
Edward Data Services	OH	207.78
Cardinal Business Media, Inc.	PA	265.36
Federal Express Corporation	TN	4378.02
Ingram	TX	2154.42

MORE ON COMPLEX QUERIES

A query that uses three subqueries

```
SELECT summary1.vendor_state, summary1.vendor_name,
       top_in_state.sum_of_invoices
FROM
    (
    SELECT v_sub.vendor_state, v_sub.vendor_name,
        SUM(i_sub.invoice_total) AS sum_of_invoices
    FROM invoices i_sub JOIN vendors v_sub
        ON i_sub.vendor_id = v_sub.vendor_id
    GROUP BY v_sub.vendor_state, v_sub.vendor_name
    ) summary1
    JOIN
        (
        SELECT summary2.vendor_state,
            MAX(summary2.sum_of_invoices) AS sum_of_invoices
        FROM
            (
            SELECT v_sub.vendor_state, v_sub.vendor_name,
                SUM(i_sub.invoice_total) AS sum_of_invoices
            FROM invoices i_sub JOIN vendors v_sub
                ON i_sub.vendor_id = v_sub.vendor_id
            GROUP BY v_sub.vendor_state, v_sub.vendor_name
            ) summary2
        GROUP BY summary2.vendor_state
        ) top_in_state
    ON summary1.vendor_state =
            top_in_state.vendor_state AND
       summary1.sum_of_invoices =
           top_in_state.sum_of_invoices
ORDER BY summary1.vendor_state
```

The result set

	VENDOR_STATE	VENDOR_NAME	SUM_OF_INVOICES
1	AZ	Wells Fargo Bank	662
2	CA	Digital Dreamworks	7125.34
3	DC	Reiter's Scientific & Pro Books	600
4	MA	Dean Witter Reynolds	1367.5
5	MI	Malloy Lithographing Inc	119892.41
6	NV	United Parcel Service	23177.96
7	OH	Edward Data Services	207.78

(10 rows)

Pseudocode for the query

```
SELECT summary1.vendor_state, summary1.vendor_name,
       top_in_state.sum_of_invoices
FROM (inline view returning
          vendor_state, vendor_name, sum_of_invoices)
          AS summary1
    JOIN (inline view returning vendor_state,
              max(sum_of_invoices))
              AS top_in_state
        ON summary1.vendor_state =
           top_in_state.vendor_state
      AND summary1.sum_of_invoices =
           top_in_state.sum_of_invoices
ORDER BY summary1.vendor_state
```

Pseudocode for the Top_In_State subquery

```
SELECT summary2.vendor_state,
       MAX(summary2.sum_of_invoices)
FROM (inline view returning vendor_state,
          vendor_name, sum_of_invoices)
          AS summary2
GROUP BY summary2.vendor_state
```

The Summary1 and Summary2 subqueries

```
SELECT v_sub.vendor_state, v_sub.vendor_name,
    SUM(i_sub.invoice_total) AS sum_of_invoices
FROM invoices i_sub JOIN vendors v_sub
    ON i_sub.vendor_id = v_sub.vendor_id
GROUP BY v_sub.vendor_state, v_sub.vendor_name
ORDER BY v_sub.vendor_state, v_sub.vendor_name
```

The result of Summary1 and Summary2

	VENDOR_STATE	VENDOR_NAME	SUM_OF_INVOICES
1	AZ	Wells Fargo Bank	662
2	CA	Abbey Office Furnishings	17.5
3	CA	Bertelsmann Industry Svcs. Inc	6940.25

(34 rows)

The result of the Top_In_State subquery

	VENDOR_STATE	SUM_OF_INVOICES
1	CA	7125.34
2	MA	1367.5
3	OH	207.78

(10 rows)

CORRELATED SUBQUERIES

Out of scope for class

Problem to solve:

Pull vendors with an above average invoice total

A query that uses a correlated subquery

```
SELECT vendor_id, invoice_number, invoice_total
FROM invoices inv_main
WHERE invoice_total >
     (SELECT AVG(invoice_total)
     FROM invoices inv_sub
     WHERE inv_sub.vendor_id = inv_main.vendor_id)
ORDER BY vendor_id, invoice_total
```

INSTEAD…try the SYNTAX to use an IN-LINE JOIN INSTEAD

```
SELECT *
FROM table1 alias_1 inner join
 (sub-query) alias_2
  on alias_1.key = alias_2.key
```

The value returned by the subquery for vendor 95

```
28.50166...
```

The result set

	VENDOR_ID	INVOICE_NUMBER	INVOICE_TOTAL
6	83	31359783	1575
7	95	111-92R-10095	32.7
8	95	111-92R-10093	39.77
9	95	111-92R-10092	46.21
10	110	P-0259	26881.4

(36 rows) **But, correlated can be complicated!**

A query that uses a correlated subquery

```
SELECT vendor_name,
    (SELECT MAX(invoice_date) FROM invoices
    WHERE invoices.vendor_id =
        vendors.vendor_id) AS latest_inv
FROM vendors
ORDER BY latest_inv
```

The result set

	VENDOR_NAME	LATEST_INV
1	IBM	14-MAR-14
2	Wang Laboratories, Inc.	16-APR-14
3	Reiter's Scientific & Pro Books	17-APR-14
4	United Parcel Service	26-APR-14
5	Wakefield Co	26-APR-14
6	Zylka Design	01-MAY-14
7	Abbey Office Furnishings	02-MAY-14

(122 rows)

The same query restated using a join

```
SELECT vendor_name,
    MAX(invoice_date) AS latest_inv
FROM vendors v
    LEFT JOIN invoices i
        ON v.vendor_id = i.vendor_id
GROUP BY vendor_name
ORDER BY latest_inv
```

The same result set

	VENDOR_NAME	LATEST_INV
1	IBM	14-MAR-14
2	Wang Laboratories, Inc.	16-APR-14
3	Reiter's Scientific & Pro Books	17-APR-14
4	United Parcel Service	26-APR-14
5	Wakefield Co	26-APR-14
6	Zylka Design	01-MAY-14
7	Abbey Office Furnishings	02-MAY-14

(122 rows)

Problem to solve:

Pull vendors without invoices

The syntax of a subquery with EXISTS

```
WHERE [NOT] EXISTS (subquery)
```

A query that returns vendors without invoices

```
SELECT vendor_id, vendor_name, vendor_state
FROM vendors
WHERE NOT EXISTS
    (SELECT *
    FROM invoices
    WHERE invoices.vendor_id = vendors.vendor_id)
```

The result set

	VENDOR_ID	VENDOR_NAME	VENDOR_STATE
53	33	Nielson	OH
54	35	Cal State Termite	CA
55	36	Graylift	CA
56	38	Venture Communications Int'l	NY
57	39	Custom Printing Company	MO
58	40	Nat Assoc of College Stores	OH

(88 rows)

PART 3

Chapter 8

Data types and functions

Built-in data type categories

- **Character** – (i.e. strings)
- **Numeric** – (integers and decimals)
- **Temporal** – (e.g. dates, times)
- Large object (LOB) – (e.g. Text, images, sounds, video)
- Rowid – (address for each row in a database)

ANSI type	Oracle equivalent
CHARACTER (n)	CHAR (n)
CHAR (n)	
CHARACTER VARYING (n)	VARCHAR2 (n)
CHAR VARYING (n)	
NATIONAL CHARACTER (n)	NCHAR (n)
NATIONAL CHAR (n)	
NCHAR (n)	
NATIONAL CHARACTER VARYING (n)	NVARCHAR2 (n)
NATIONAL CHAR VARYING (n)	
NCHAR VARYING (n)	
NUMERIC (p,s)	NUMBER (p,s)
DECIMAL (p,s)	
INTEGER	NUMBER (38)
INT	
SMALLINT	
FLOAT	FLOAT (126)
DOUBLE PRECISION	FLOAT (126)
REAL	FLOAT (63)

1 byte = 256 characters

ABC
123

2-3 bytes = 65,000 characters

七
米

p = total number of digits stored
s = number of decimal digits that can be stored

The date/time data types

```
DATE
TIMESTAMP[(fsp)]
TIMESTAMP [(fsp)] WITH TIME ZONE
TIMESTAMP [(fsp)] WITH LOCAL TIME ZONE
INTERVAL YEAR [(yp)] TO MONTH
INTERVAL DAY [(dp)] TO SECOND [(fsp)]
```

Terms to know

- Temporal data types
- Date/time data type

The large object data types

- CLOB – characters
- NCLOB - Unicode characters
- BLOB – unstructured data (image, sound, video)
- BFILE – pointer to a large file outside database

Oracle functions for converting data

```
TO_CHAR(expr[, format])
TO_NUMBER(expr[, format])
TO_DATE(expr[, format])
```

NOTE: format will use a default if not specified

Examples that use TO functions

```
TO_CHAR(1975.5)
TO_CHAR(1975.5, '$99,999.99')

TO_CHAR(SYSDATE)
TO_CHAR(SYSDATE, 'DD-MON-YYYY HH24:MI:SS')

TO_NUMBER('1975.5')
TO_NUMBER('$1,975.5', '$99,999.99')

TO_DATE('15-APR-14')
TO_CHAR(TO_DATE('15-APR-14'), 'DD-MON-YYYY HH24:MI:SS')
```

Practice:

1. Select SYSDATE from ***dual*** and format with the following but convert to string with the following formats
   - OCT-2019
   - October 23, 2019
   - 10/23/19 06:10:01
   - WED, 10-23-2019

2. Select invoice_total from invoices and format as char, $999.99

FYI other option: CAST functions

```
SELECT   invoice_id,
         invoice_date,
         invoice_total,
         CAST(invoice_date AS VARCHAR2(9)),
         CAST(invoice_total AS NUMBER(9))
FROM     invoices
```

Same query using TO_ functions

```
SELECT   invoice_id,
         invoice_date,
         invoice_total,
         to_char(invoice_date),
         to_number(invoice_total)
FROM     invoices
```

NOTE:

- ANSI-standard so you can use on other DBMS
- TO_ functions gives more control
- Expect that if you are using different DBMSs (MySQL, SQL Server, DB2) that you ma have to use different functions to convert data. Googling never ends!

Working with **Numbers**

Number format elements

```
9          = digits
. or D     = decimals
G or ,     = group separator
0          = leading/trailing 0
$          = $ sign
L          = local currency
U
C          = currency symbol
S          = - or + sign
MI         = - for negatives
PR         = brackets for neg.
FM         = remove trail/lead 0
EEEE
```

Examples

Value	Format	Output
1975.5	(none specified)	1975.5
1975.5	999	###
1975.5	9999	1976
1975.5	9,999.9	1,975.5
1975.5	9G999D9	1,975.5
1975.5	99,999.99	1,975.50
1975.5	09,999.990	01,975.500
1975.5	$99,999.99	$1,975.50
1975.5	L9,999.99	$1,975.50
1975.5	U9,999.99	$1,975.50
1975.5	C9,999.99	USD1,975.50
1975.5	S9,999.99	+1,975.50
-1975.5	9,999.99S	1,975.50-
-1975.5	9,999.99MI	1,975.50-
-1975.5	9,999.99PR	<1,975.50>
1975.5	9,999.99PR	1,975.50
01975.50	FM9,999.99	1,975.5
1975.5	9.99EEEE	1.98E+03

Practice: Experiment with format of currency fields on invoices, make a negative

Some common numeric functions

```
ROUND(number[, length])
TRUNC(number[, length])
CEIL(number)
FLOOR(number)
ABS(number)
SIGN(number)
MOD(number, number_divisor)
POWER(number, number_exponent)
SQRT(number)
```

Example	Result
ROUND(12.5)	13
ROUND(12.4999, 0)	12
ROUND(12.4999, 1)	12.5
ROUND(12.4944, 2)	12.49
ROUND(1264.99, -2)	1300
TRUNC(12.5)	12
TRUNC(12.4999, 1)	12.4
TRUNC(12.4944, 2)	12.49
TRUNC(1264.99, -2)	1200
CEIL(1.25)	2
CEIL(-1.25)	-1
FLOOR(1.25)	1
FLOOR(-1.25)	-2

Example	Result
ABS(1.25)	1.25
ABS(-1.25)	1.25
SIGN(1.25)	1
SIGN(0)	0
SIGN(-1.25)	-1
MOD(10, 10)	0
MOD(10, 9)	1
POWER(2, 2)	4
POWER(2, 2.5)	5.65685…
SQRT(4)	2
SQRT(5)	2.23606…

The Float_Sample table

	FLOAT_ID	FLOAT_VALUE
1	1	0.999999999999999
2	2	1.0
3	3	1.000000000000001
4	4	1234.56789012345
5	5	999.04440209348
6	6	24.04849

A statement that searches for an exact value

```
SELECT *
FROM float_sample
WHERE float_value = 1
```

The result set

	FLOAT_ID	FLOAT_VALUE
1	2	1.0

A statement that searches for a range of values

```
SELECT *
FROM float_sample
WHERE float_value BETWEEN 0.99 AND 1.01
```

The result set

	FLOAT_ID	FLOAT_VALUE
1	1	0.999999999999999
2	2	1.0
3	3	1.000000000000001

A statement that searches for rounded values

```
SELECT *
FROM float_sample
WHERE ROUND(float_value, 2) = 1
```

The result set

	FLOAT_ID	FLOAT_VALUE
1	1	0.999999999999999
2	2	1.0
3	3	1.000000000000001

Working with **Dates**

Common date/time format elements

Element	Description	Element	Description
AD	Anno Domini	DAY	Name of day padded with spaces
BC	Before Christ	DY	Abbreviated name of day
CC	Century	DDD	Day of year (1-366)
YEAR	Year spelled out	DD	Day of month (01-31)
YYYY	Four-digit year	D	Day of week (1-7)
YY	Two-digit year	HH	Hour of day (01-12)
RR	Two-digit round year	HH24	Hour of day (01-24)
Q	Quarter of year (1-4)	MI	Minute (00-59)
MONTH	Name of month padded with spaces	SS	Second (00-59)
MON	Abbreviated name of month	SSSSS	Seconds past midnight (0-86399)
MM	Month (01-12)	FF[1-9]	Fractional seconds
WW	Week of year (1-52)	PM	Post Meridian
W	Week of month (1-5)	AM	Ante Meridian

Practice:

Make something like this based on ***invoices***

INVOICE_ID	INVOICE_DATE	INVOICE_TOTAL
1	TUE - FEB 25, 2014	$116.54
2	FRI - MAR 14, 2014	$1,083.58
3	FRI - APR 11, 2014	$20,551.18
4	WED - APR 16, 2014	$26,881.40
5	WED - APR 16, 2014	$936.93
6	THU - APR 17, 2014	$2,312.20
7	THU - APR 17, 2014	$600.00
8	FRI - APR 18, 2014	$1,927.54
9	SAT - APR 19, 2014	$2,184.11
10	THU - APR 24, 2014	$2,318.03

Date Examples

(none specified)	19-AUG-14
DD-MON-YYYY	19-AUG-2014
DD-Mon-YY	19-Aug-14
MM/DD/YY	08/19/14
YYYY-MM-DD	2014-08-19
Dy Mon DD, YY	Tue Aug 19, 14
Month DD, YYYY B.C.	August 19, 2014 A.D.

Time Examples

HH:MI	04:20
HH24:MI:SS	16:20:36
HH:MI AM	04:20 PM
HH:MI A.M.	04:20 P.M.
HH:MI:SS.FF5	04:20:36.12345
HH:MI:SS.FF4	04:20:36.1234
YYYY-MM-DD HH:MI:SS AM	2014-08-19 04:20:36 PM

Examples that parse a date/time value

Example	Result
TO_CHAR(SYSDATE,'DD-MON-RR HH:MI:SS')	19-AUG-14 04:20:36 PM
TO_CHAR(SYSDATE, 'YEAR')	TWO THOUSAND FOURTEEN
TO_CHAR(SYSDATE, 'YEAR')	Two Thousand Fourteen
TO_CHAR(SYSDATE, 'YYYY')	2014
TO_CHAR(SYSDATE, 'YY')	14
TO_CHAR(SYSDATE, 'MONTH')	AUGUST
TO_CHAR(SYSDATE, 'MON')	AUG
TO_CHAR(SYSDATE, 'MM')	08
TO_CHAR(SYSDATE, 'DD')	19
TO_CHAR(SYSDATE, 'DAY')	TUESDAY
TO_CHAR(SYSDATE, 'DY')	TUES
TO_CHAR(SYSDATE, 'HH24')	16
TO_CHAR(SYSDATE, 'HH')	04

Example	Result
TO_CHAR(SYSDATE, 'MI')	20
TO_CHAR(SYSDATE, 'SS')	36
TO_CHAR(SYSDATE, 'CC')	21
TO_CHAR(SYSDATE, 'Q')	3
TO_CHAR(SYSDATE, 'WW')	34
TO_CHAR(SYSDATE, 'W')	3
TO_CHAR(SYSDATE, 'DDD')	232
TO_CHAR(SYSDATE, 'D')	3

Example	Result
TO_NUMBER(TO_CHAR(SYSDATE, 'HH24'))	16
TO_NUMBER(TO_CHAR(SYSDATE, 'HH'))	4
TO_NUMBER(TO_CHAR(SYSDATE, 'SS'))	36

Some common date/time functions

```
SYSDATE
CURRENT_DATE
ROUND(date[, date_format])
TRUNC(date[, date_format])
MONTHS_BETWEEN(date1, date2)
ADD_MONTHS(date, integer_months)
LAST_DAY(date)
NEXT_DAY(date, day_of_week)
```

Two operators for working with dates

```
+
-
```

Examples that use the date/time functions

Example	Result
SYSDATE	19-AUG-14 04:20:36 PM
ROUND(SYSDATE)	20-AUG-14 12:00:00 AM
TRUNC(SYSDATE, 'MI')	19-AUG-14 04:20:00 PM
MONTHS_BETWEEN('01-SEP-14','01-AUG-14')	1
MONTHS_BETWEEN('15-SEP-14','01-AUG-14')	1.451...
ADD_MONTHS('19-AUG-14', -1)	19-JUL-14
ADD_MONTHS('19-AUG-14', 11)	19-JUL-15
LAST_DAY('15-FEB-14')	29-FEB-14
NEXT_DAY('15-AUG-14', 'FRIDAY')	22-AUG-14
NEXT_DAY('15-AUG-14', 'THURS')	21-AUG-14
SYSDATE - 1	18-AUG-14
SYSDATE + 7	26-AUG-14
SYSDATE - TO_DATE('01-JAN-14')	231
TO_DATE('01-JAN-14') - SYSDATE	-231

The Date_Sample table

	DATE_ID	START_DATE
1	1	01-MAR-79
2	2	28-FEB-99
3	3	31-OCT-03
4	4	28-FEB-05
5	5	28-FEB-06
6	6	01-MAR-06

A SELECT statement that fails to return a row

```
SELECT *
FROM date_sample
WHERE start_date = '28-FEB-06'
```

A SELECT statement that searches for a range

```
SELECT *
FROM date_sample
WHERE start_date >= '28-FEB-06'
    AND start_date < '01-MAR-06'
```

The result set

	DATE_ID	START_DATE
1	5	28-FEB-06

A statement with TRUNC to remove time values

```
SELECT *
FROM date_sample
WHERE TRUNC(start_date) = '28-FEB-06'
```

The result set

	DATE_ID	START_DATE
1	5	28-FEB-06

The Date_Sample table

	DATE_ID	START_DATE
1	1	01-MAR-79
2	2	28-FEB-99
3	3	31-OCT-03
4	4	28-FEB-05
5	5	28-FEB-06
6	6	01-MAR-06

A SELECT statement that fails to return a row

```
SELECT *
FROM date_sample
WHERE start_date = TO_DATE('10:00:00', 'HH24:MI:SS')
```

A statement that ignores the date component

```
SELECT *
FROM date_sample
WHERE TO_CHAR(start_date, 'HH24:MI:SS') = '10:00:00'
```

The result set

	DATE_ID	START_DATE
1	4	28-FEB-05

Another statement that fails to return a row

```
SELECT * FROM date_sample
WHERE start_date >= TO_DATE('09:00:00', 'HH24:MI:SS')
  AND start_date  < TO_DATE('12:59:59', 'HH24:MI:SS')
```

Another statement that ignores the date

```
SELECT * FROM date_sample
WHERE TO_CHAR(start_date, 'HH24:MI:SS') >= '09:00:00'
  AND TO_CHAR(start_date, 'HH24:MI:SS')  < '12:59:59'
```

The result set

	DATE_ID	START_DATE
1	4	28-FEB-05
2	6	01-MAR-06

Working with
Characters/Strings

Some common character functions

```
LTRIM(string[, trim_string])
RTRIM(string[, trim_string])
TRIM(string) Removes leading/trailing spaces
TRIM([trim_char FROM ]string)

LPAD(string, length[, pad_string])
RPAD(string, length[, pad_string])

LOWER(string)
UPPER(string)
INITCAP(string)
```

Practice:

1. Select invoice_date with MONTH format and then use TRIM to remove spaces
2. Left pad invoice_total with 15 periods '.'
3. Select vendor_name in ALL CAPS
4. Select vendor_state in lowercase

Character function examples

Example	Result
LTRIM(' John Smith ')	'John Smith '
RTRIM(' John Smith ')	' John Smith'
TRIM(' John Smith ')	'John Smith'
LTRIM('$0019.99', '$0')	'19.99'
TRIM('$' FROM '$0019.99')	'0019.99'
LPAD('$19.99', 15)	' $19.99'
LPAD('$2150.78', 15)	' $2150.78'
LPAD('$2150.78', 15, '.')	'.......$2150.78'
RPAD('John', 15)	'John '
RPAD('John', 15, '.')	'John.........'
LOWER('CA')	'ca'
UPPER('ca')	'CA'
INITCAP('john smith')	'John Smith'
INITCAP('JOHN SMITH')	'John Smith'

More character functions

```
SUBSTR(string, start[, length])
LENGTH(string)
INSTR(string, find [,start])
REPLACE(string, find, replace)
```

Practice:

1. Select the vendor_name and vendor_name length
2. Select vendor_phone twice. Once with no formatting and a 2nd time in the following format 999-999-9999. Try doing it two ways (i.e. one with REPLACE and another with SUBSTR)

Character function examples (continued)

Example	Result
`SUBSTR('(559) 555-1212', 1, 5)`	`'(559)'`
`SUBSTR('(559) 555-1212', 7, 3)`	`'555'`
`SUBSTR('(559) 555-1212', 7)`	`'555-1212'`
`INSTR('(559) 555-1212', ' ')`	`6`
`INSTR('559-555-1212', '-')`	`4`
`INSTR('559-555-1212', '-', 5)`	`8`
`INSTR('559-555-1212', '1212')`	`9`
`LENGTH('(559) 555-1212')`	`14`
`LENGTH('  (559) 555-1212  ')`	`18`
`REPLACE('559-555-1212', '-', '.')`	`'559.555.1212'`
`REPLACE('559-555-1212', '-', '')`	`'5595551212'`

The String_Sample table

	ID	NAME
1	1	Lizbeth Darien
2	2	Darnell O'Sullivan
3	17	Lance Pinos-Potter
4	20	Jean Paul Renard
5	3	Alisha von Strump

A SELECT statement that parses a string

```
SELECT SUBSTR(name, 1, (INSTR(name, ' ') - 1))
           AS first_name,
       SUBSTR(name, (INSTR(name, ' ') + 1))
           AS last_name
FROM string_sample
```

The result set

	FIRST_NAME	LAST_NAME
1	Lizbeth	Darien
2	Darnell	O'Sullivan
3	Lance	Pinos-Potter
4	Jean	Paul Renard
5	Alisha	von Strump

Practice:

The product_name in ***products*** starts with the brand and is followed by the actual product's name.

Parse the field into two new fields:

1. *Brand* which contains the first word in the product_name
2. *Instrument_Name* which is any text that comes after the brand.

Sorting char/varchar columns like #s

A table sorted by a character column

```
SELECT * FROM string_sample
ORDER BY id
```

The result set (questionable results)

	ID	NAME
1	1	Lizbeth Darien
2	17	Lance Pinos-Potter
3	2	Darnell O'Sullivan
4	20	Jean Paul Renard
5	3	Alisha von Strump

In this example table, "ID" is CHAR so it won't sort like a number

Solution: Convert character column to number

```
SELECT * FROM string_sample
ORDER BY TO_NUMBER(id)
```

The result set

	ID	NAME
1	1	Lizbeth Darien
2	2	Darnell O'Sullivan
3	3	Alisha von Strump
4	17	Lance Pinos-Potter
5	20	Jean Paul Renard

Another option: Pad character column with leading 0 to make sortable

```
SELECT LPAD(id, 2, '0') AS lpad_id, name
FROM string_sample
ORDER BY lpad_id
```

The result set

	LPAD_ID	NAME
1	01	Lizbeth Darien
2	02	Darnell O'Sullivan
3	03	Alisha von Strump
4	17	Lance Pinos-Potter
5	20	Jean Paul Renard

Handling Conditional with **CASE**

The syntax of the simple CASE expression

```
CASE input_expression
     WHEN when_expression_1 THEN result_expression_1
    [WHEN when_expression_2 THEN result_expression_2]...
    [ELSE else_result_expression]
END
```

The syntax of the searched CASE expression

```
CASE
     WHEN conditional_expression_1 THEN result_expression_1
    [WHEN conditional_expression_2
          THEN result_expression_2]...
    [ELSE else_result_expression]
END
```

The syntax of the simple CASE expression

```
CASE input_expression
     WHEN when_expression_1 THEN result_expression_1
    [WHEN when_expression_2 THEN result_expression_2]...
    [ELSE else_result_expression]
END
```

Category Names

1 = 'Guitars'
2 = 'Bass'
3 = 'Drums'
4 = 'Keyboard‘

A statement that uses a simple CASE expression

```
SELECT invoice_number, terms_id,
    CASE terms_id
        WHEN 1 THEN 'Net due 10 days'
        WHEN 2 THEN 'Net due 20 days'
        WHEN 3 THEN 'Net due 30 days'
        WHEN 4 THEN 'Net due 60 days'
        WHEN 5 THEN 'Net due 90 days'
    END AS terms
FROM invoices
```

The result set

	INVOICE_NUMBER	TERMS_ID	TERMS
1	QP58872	1	Net due 10 days
2	Q545443	1	Net due 10 days
3	P-0608	5	Net due 90 days

Sort by category, price

PRODUCT_NAME	LIST_PRICE	CATEGORY_NAME
Hofner Icon	499.99	Bass
Fender Precision	799.99	Bass
Ludwig 5-piece Drum Set with Cymbals	699.99	Drums
Tama 5-Piece Drum Set with Cymbals	799.99	Drums
Washburn D10S	299	Guitars
Rodriguez Caballero 11	415	Guitars
Yamaha FG700S	489.99	Guitars
Fender Stratocaster	699	Guitars
Gibson Les Paul	1199	Guitars
Gibson SG	2517	Guitars

The syntax of the searched CASE expression

```
CASE
    WHEN conditional_expression_1 THEN result_expression_1
   [WHEN conditional_expression_2
        THEN result_expression_2]...
   [ELSE else_result_expression]
END
```

A statement that uses a searched CASE

```
SELECT invoice_number, invoice_total, invoice_date,
       invoice_due_date,
    CASE
        WHEN (SYSDATE - invoice_due_date) > 30
            THEN 'Over 30 days past due'
        WHEN (SYSDATE - invoice_due_date) > 0
            THEN '1 to 30 days past due'
        ELSE 'Current'
    END AS status
FROM invoices
WHERE invoice_total - payment_total - credit_total > 0
```

Product_Grade

>= 1000 = 'Professional'

>= 500 = 'Intermediate'

Everything else = 'Beginner'

The result set

	INVOICE_NUMBER	INVOICE_TOTAL	INVOICE_DATE	INVOICE_DUE_DATE	STATUS
37	547481328	224	20-MAY-08	25-JUN-08	Over 30 days past due
38	40318	21842	18-JUL-08	20-JUL-08	Over 30 days past due
39	31361833	579.42	23-MAY-08	09-JUN-08	Over 30 days past due
40	456789	8344.5	01-AUG-14	31-AUG-14	Current

Handling NULL values

The COALESCE, NVL, and NVL2 syntax

```
COALESCE(expression1 [, expression2][, expression3]...)
NVL(expression, null_replacement)
NVL2(expression, not_null_replacement, null_replacement)
```

A statement that uses the COALESCE function

```
SELECT payment_date, invoice_due_date,
       COALESCE(payment_date, invoice_due_date,
           TO_DATE('01-JAN-1900'))
       AS payment_date_2
FROM invoices
```

The result set

	PAYMENT_DATE	INVOICE_DUE_DATE	PAYMENT_DATE_2
1	11-APR-08	22-APR-08	11-APR-08
2	14-MAY-08	23-MAY-08	14-MAY-08
3	(null)	30-JUN-08	30-JUN-08
4	12-MAY-08	16-MAY-08	12-MAY-08
5	13-MAY-08	16-MAY-08	13-MAY-08
6	(null)	26-JUN-08	26-JUN-08

The COALESCE, NVL, and NVL2 syntax

```
COALESCE(expression1 [, expression2][, expression3]...)
NVL(expression, null_replacement)
NVL2(expression, not_null_replacement, null_replacement)
```

A SELECT statement that uses the NVL function

```
SELECT payment_date,
       NVL(TO_CHAR(payment_date), 'Unpaid') AS payment_date_2
FROM invoices
```

The result set

	PAYMENT_DATE	PAYMENT_DATE_2
1	11-APR-08	11-APR-08
2	14-MAY-08	14-MAY-08
3	(null)	Unpaid
4	12-MAY-08	12-MAY-08

A SELECT statement that uses the NVL2 function

```
SELECT payment_date,
       NVL2(payment_date, 'Paid', 'Unpaid') AS payment_date_2
FROM invoices
```

The result set

	PAYMENT_DATE	PAYMENT_DATE_2
1	11-APR-08	Paid
2	14-MAY-08	Paid
3	(null)	Unpaid
4	12-MAY-08	Paid

Practice:

1. Pull a report that checks vendor contact is up-to-date like so.

VENDOR_ID	VENDOR_NAME	CONTACT_CHECK
81	Wang Laboratories, Inc.	(800) 555-0344
82	Reiter's Scientific & Pro Books	(202) 555-5561
83	Ingram	Update contact
84	Boucher Communications Inc	(215) 555-8000
85	Champion Printing Company	(800) 555-1957
86	Computerworld	(617) 555-0700
87	DMV Renewal	Update contact
88	Edward Data Services	(513) 555-3043
89	Evans Executone Inc	Update contact
90	Wakefield Co	(559) 555-4744

2. Update to show not null values as ‘Okay’ & sort

VENDOR_ID	VENDOR_NAME	CONTACT_CHECK
40	Nat Assoc of College Stores	Update contact
58	Fresno Rack & Shelving Inc	Update contact
60	The Mailers Guide Co	Update contact
64	Texaco	Update contact
65	The Drawing Board	Update contact
66	Ascom Hasler Mailing Systems	Update contact
87	DMV Renewal	Update contact
72	Data Reproductions Corp	Okay
73	Executive Office Products	Okay
74	Leslie Company	Okay
75	Retirement Plan Consultants	Okay
76	Simon Direct Inc	Okay
77	State Board Of Equalization	Okay
78	The Freeport Center	Okay

Rank & Row Numbers

(Using Window Functions)

A query that uses RANK and DENSE_RANK

```
SELECT RANK() OVER (ORDER BY invoice_total) AS rank,
       DENSE_RANK() OVER (ORDER BY invoice_total)
           AS dense_rank,
       invoice_total, invoice_number
FROM invoices
```

The result set

	RANK	DENSE_RANK	INVOICE_TOTAL	INVOICE_NUMBER
1	1	1	6	25022117
2	1	1	6	24863706
3	1	1	6	24780512
4	4	2	9.95	21-4748363
5	4	2	9.95	21-4923721
6	6	3	10	4-342-8069

Practice:

Pull a list of products and their rank by list_price descending.

Which is the 2nd ranked product by price?

PRODUCT_NAME	LIST_PRICE	RANK
Gibson SG	2517	1
Gibson Les Paul	1199	2
Tama 5-Piece Drum Se...	799.99	3
Fender Precision	799.99	3
Ludwig 5-piece Drum ...	699.99	5
Fender Stratocaster	699	6
Hofner Icon	499.99	7
Yamaha FG700S	489.99	8
Rodriguez Caballero 11	415	9
Washburn D10S	299	10

A query that uses the ROW_NUMBER function

```
SELECT ROW_NUMBER() OVER(ORDER BY vendor_name)
       AS row_number, vendor_name
FROM vendors
```

The result set

	ROW_NUMBER	VENDOR_NAME
1	1	ASC Signs
2	2	AT&T
3	3	Abbey Office Furnishings
4	4	American Booksellers Assoc
5	5	American Express