The University of Texas at Austin
McCombs School of Business

MIS 381N INTRO. TO DATABASE MANAGEMENT

SQL Essentials

`WHERE, ORDER BY, JOIN`

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	WHERE, ORDER BY FROM, JOIN
Hands-On	Exercises
Looking Forward	Homework 3 Harvard case & article

First, a little review

USING COLUMN FUNCTIONS

- **String Expression**
  - Use the string operator || to concatenate column or literal strings together
  - Select city || ‘,’ || state or Select ‘Mr. or Mrs.’ || last_name
- **Arithmetic Expression**
  - Use +, -, /, * to do arithmetic calcs on numeric columns
  - e.g. Select invoice_total – payment_total
- **Alias – give custom column names for calculated columns**
  - e.g. Select (invoice_total – payment_total) AS “Amount Owed”
  - NOTE: Include Quotes (not ticks) around column alias if your column name has spaces are includes spaces. Good rule of thumb is to always use AS and quotes

TYPES OF COLUMN FUNCTIONS

- Scalar = Operates on a single value and returns a single value
- SYSDATE – returns today's date/time. Like NOW() function in Excel
- ROUND – round decimals to whole numbers
- SUBSTR – Returns certain part of a string. Like MID() function in Excel
- TO_CHAR – convert number/date to string
- TO_DATE – convert string to a date
- MOD – returns remainder of division of two numbers

DISTINCT AND ROWNUM

- DISTINCT – finds the unique occurrence of a value
- Find all the city and states where our vendors are from
- ROWNUM – pseudo column that can be used to limit the rows that are retrieved
- Retrieve the first five rows from the vendors table

ORDER BY

WHERE

QUESTION

Is ordering and filtering required in queries?

Which commands are mandatory in a query?

IN A NUTSHELL: SQL ESSENTIALS

- WHERE is a row filter command
- ORDER BY is a column sort command
- FETCH/OFFSET is used for filtering after ORDER BY

WHERE CLAUSES

- WHERE <expression1> <operator> <expression2>
  Operator can be =, >, <, <=, >=, <>
- Boolean or logical operators AND, OR, NOT
- WHERE <expression1>
  [NOT] BETWEEN begin_expression AND end_expression
- WHERE <expression>
  [NOT] IN ({subquery|expression_1 [, expression_2]...})
- WHERE <expression> [NOT] LIKE pattern
  Wildcard symbols %, _
- IS NULL and IS NOT NULL

COMPOUND CONDITIONS

A compound condition without parentheses

```
SELECT invoice_number, invoice_date, invoice_total
FROM invoices
WHERE invoice_date > '01-MAY-2014' OR invoice_total > 500
    AND invoice_total - payment_total - credit_total > 0
ORDER BY invoice_number
```

	INVOICE_NUMBER	INVOICE_DATE	INVOICE_TOTAL
1	0-2058	08-MAY-14	37966.19
2	0-2060	08-MAY-14	23517.58
3	0-2436	07-MAY-14	10976.06

```
(91 rows selected)
```

The order of precedence for compound conditions

1. NOT
2. AND
3. OR

The same compound condition with parentheses

```
WHERE (invoice_date > '01-MAY-2014'
       OR invoice_total > 500)
    AND invoice_total - payment_total - credit_total > 0
ORDER BY invoice_number
```

Tip: Just use parenthesis to be sure

ORDER OF OPERATIONS

ORDER	CLAUSE	FUNCTION
1	from	Choose and join tables to get base data.
2	where	Filters the base data.
3	group by	Aggregates the base data.
4	having	Filters the aggregated data.
5	select	Returns the final data.
6	order by	Sorts the final data.
7	limit	Limits the returned data to a row count.

LOOKING FORWARD

Read Chapters 3 and 7

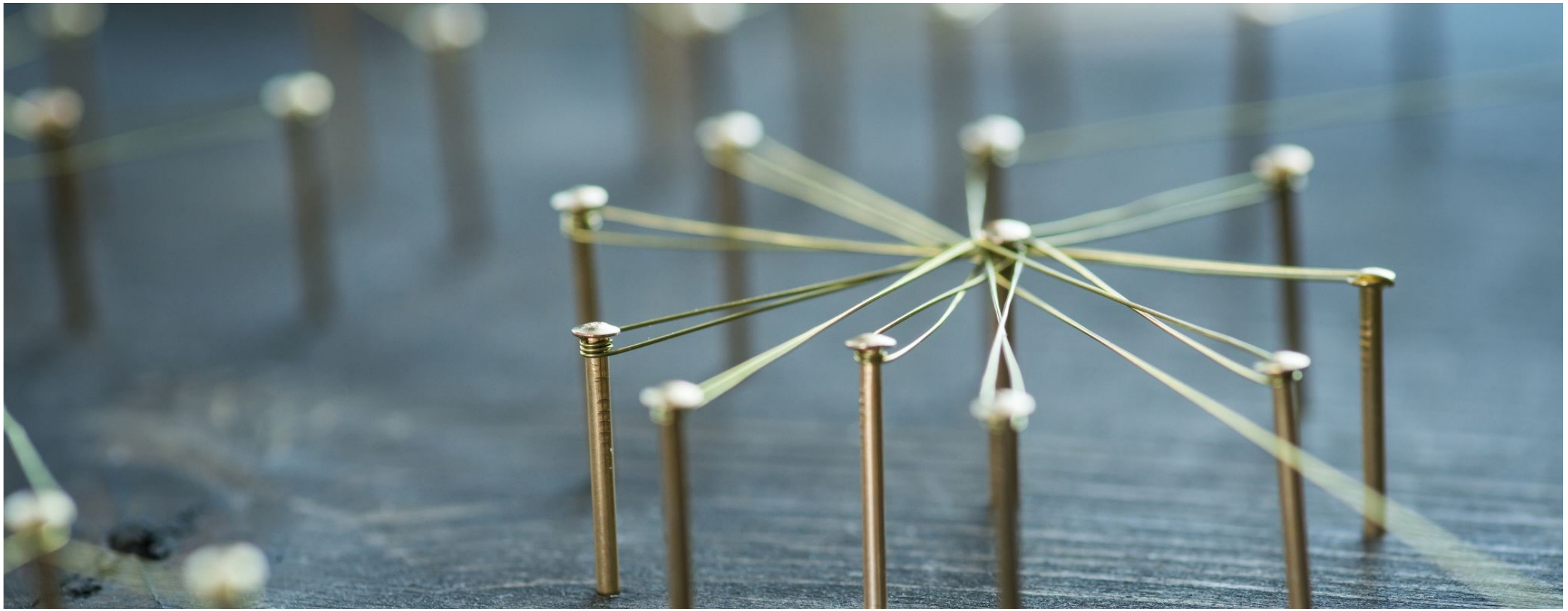
Quiz 3

Homework 3

Harvard case and article

THANK YOU

SELECT USING JOINS



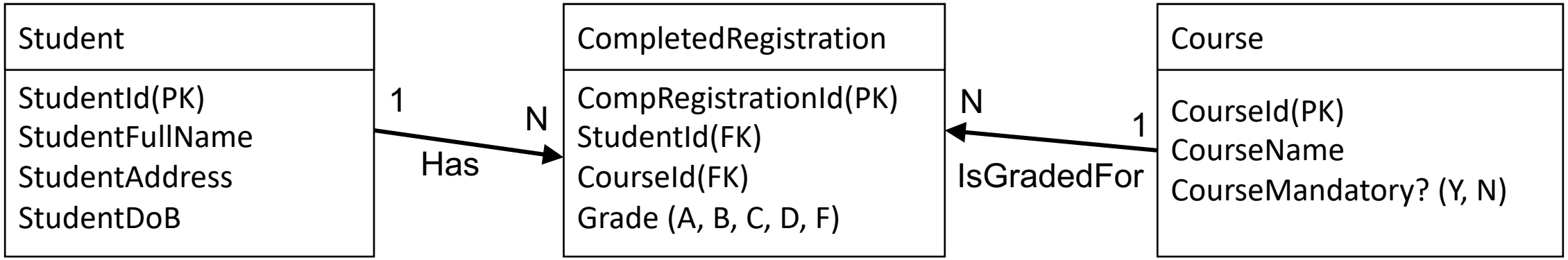
QUESTION

When could we want to join tables?

Hint: think about vendor and their invoices

FROM **JOIN**

SOMETIMES THE DATA WE NEED IS IN MULTIPLE TABLES



- How can we retrieve StudentFullName, CourseName and Grade?
- How can we retrieve data correctly from the Student, Course and CompletedRegistration tables?
- We need to look for columns that are the same in multiple tables – the primary and foreign keys?

JOIN SYNTAX FROM CLAUSE

```
SELECT column_list
FROM table_1 INNER JOIN table_2 ON join_condition_1
```

```
SELECT StudentFullName, CourseId, Grade
FROM  CompletedRegistration INNER JOIN Student
          ON CompletedRegistration.StudentId =  Student.StudentId
```

```
SELECT StudentFullName, CourseName, Grade
FROM  CompletedRegistration INNER JOIN Student
          ON CompletedRegistration.StudentId = Student.StudentId
       INNER JOIN Course
          ON CompletedRegistration.CourseId = Course.CourseID
```

JOIN SYNTAX TABLE ALIASES

```
SELECT column_list
FROM table_1 INNER JOIN table_2 ON join_condition_1
```

```
SELECT StudentFullName, CourseId, Grade
FROM  CompletedRegistration CR INNER JOIN Student S
          ON CR.StudentId = S.StudentId
```

```
SELECT StudentFullName, CourseName, Grade
FROM  CompletedRegistration CR INNER JOIN Student S
          ON CR.StudentId = S.StudentId
       INNER JOIN Course C
          ON CR.CourseId = C.CourseID
```

JOIN SYNTAX – PRECISELY DEFINING COLUMNS

```
SELECT StudentId, StudentFullName, CourseName, Grade
FROM  CompletedRegistration CR INNER JOIN Student S
          ON CR.StudentId = S.StudentId
       INNER JOIN Course C
           ON CR.CourseId = C.CourseID
```

ORA-00918: column ambiguously defined
00918. 00000 - "column ambiguously defined"

JOIN SYNTAX – PRECISELY DEFINING COLUMNS

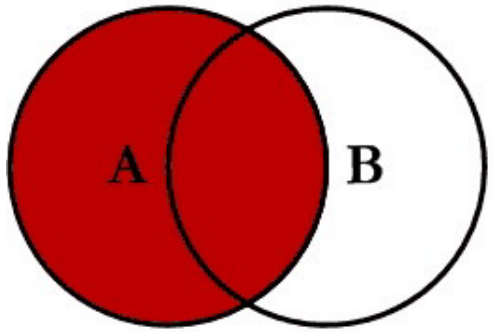
```
SELECT CR.StudentId, S.StudentFullName, C.CourseName, CR.Grade
FROM  CompletedRegistration CR INNER JOIN Student S
          ON CR.StudentId =       S.StudentId
       INNER JOIN Course C
           ON CR.CourseId = C.CourseID
```

JOIN SYNTAX – PRECISELY DEFINING COLUMNS

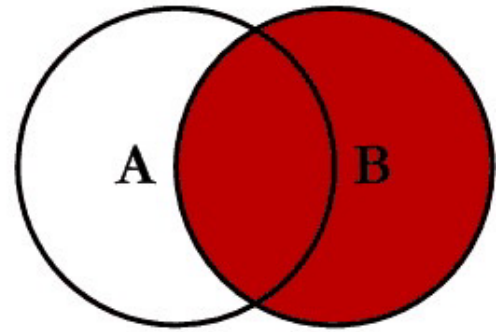
```
SELECT S.StudentFullName, C.CourseName, CR.Grade
FROM  CompletedRegistration CR, Student S, Course C
WHERE CR.StudentId = S.StudentId
                  CR.CourseId = C.CourseID
```

NOT RECOMMENDED
Correct syntax but not best practice

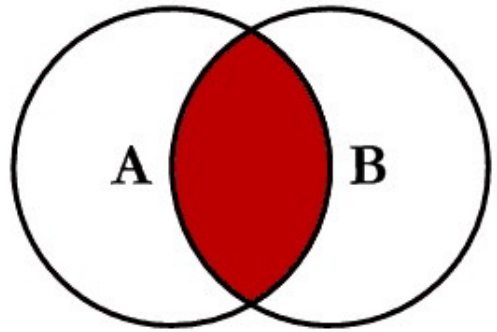
SQL JOINS



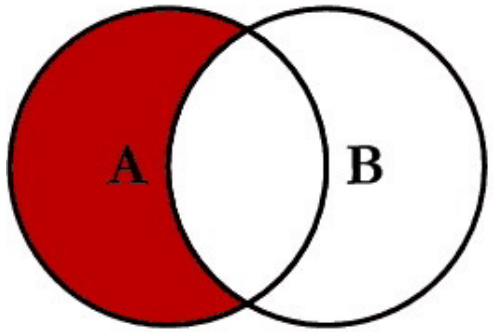

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



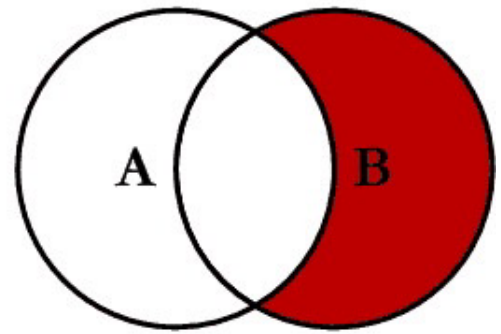

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



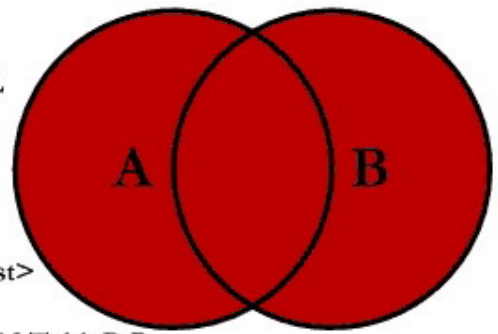

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



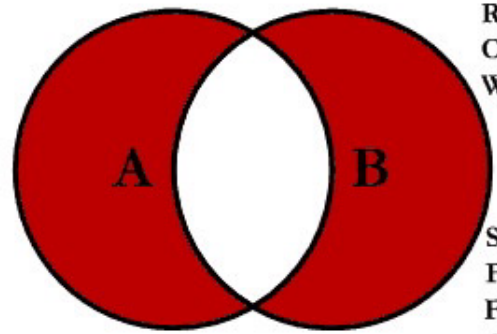

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```


important

LOOKING FORWARD

Read Chapters 3 and 7

Homework 3

Harvard case and article

THANK YOU

BACKUP SLIDES

PART 1

Chapter 3

WHERE, ORDER BY

SELECT syntax

```
SELECT column_list
FROM table_source
[WHERE filter_condition]
[ORDER BY order_by_list]
```

e.g. Sort with ORDER BY clause

```
SELECT invoice_number, invoice_date
FROM invoices
ORDER by invoice_date [DESC]
```

e.g. Select with WHERE clause

```
SELECT invoice_number, invoice_date
FROM invoices
WHERE invoice_date = '02-MAY-2014'
```

Practice Expressions in WHERE clause:

e.g. Arithmetic expression in WHERE clause

```
SELECT invoice_number,
       (invoice_total - payment_total) As "Money Due"
FROM invoices
WHERE (invoice_total - payment_total) = 0
```

The syntax of the WHERE clause with the IN operator

```
WHERE test_expression
  [NOT] IN ({subquery|expression_1 [, expression_2]...})
```

Examples of the IN operator

The IN operator with a list of numeric literals

```
WHERE terms_id IN (1, 3, 4)
```

The IN operator preceded by NOT

```
WHERE vendor_state NOT IN ('CA', 'NV', 'OR')
```

The IN operator with a subquery

```
WHERE vendor_id IN
    (SELECT vendor_id
     FROM invoices
     WHERE invoice_date = '01-MAY-2014')
```

The syntax of the WHERE clause with the BETWEEN operator

```
WHERE test_expression
    [NOT] BETWEEN begin_expression AND end_expression
```

Examples of the BETWEEN operator

The BETWEEN operator with literal values

```
WHERE invoice_date
    BETWEEN '01-MAY-2014' AND '31-MAY-2014'
```

The BETWEEN operator preceded by NOT

```
WHERE vendor_zip_code NOT BETWEEN 93600 AND 93799
```

The BETWEEN operator with a calculated value

```
WHERE invoice_total - payment_total - credit_total
    BETWEEN 200 AND 500
```

The BETWEEN operator with upper and lower limits

```
WHERE invoice_due_date BETWEEN SYSDATE AND (SYSDATE + 30)
```

A SELECT statement that sorts the result set after the WHERE clause

```
SELECT vendor_id, invoice_total
FROM invoices
WHERE ROWNUM <= 5
ORDER BY invoice_total DESC
```

	VENDOR_ID	INVOICE_TOTAL
1	110	26881.4
2	110	20551.18
3	34	1083.58
4	81	936.93
5	34	116.54

A SELECT statement that sorts the result set before the WHERE clause

```
SELECT vendor_id, invoice_total
FROM (SELECT * FROM invoices
      ORDER BY invoice_total DESC)
WHERE ROWNUM <= 5
```

	VENDOR_ID	INVOICE_TOTAL
1	110	37966.19
2	110	26881.4
3	110	23517.58
4	72	21842
5	110	20551.18

The syntax of the WHERE clause with comparison operators

```
WHERE expression_1 operator expression_2
```

The comparison operators

- =
- >
- <
- <=
- >=
- <>

The comparison operators

- =
- >
- <
- <=
- >=
- <>

Examples of WHERE clauses that retrieve…

Vendors located in Iowa

```
WHERE vendor_state = 'IA'
```

Invoices with a balance due (two variations)

```
WHERE (invoice_total - payment_total) > 0
WHERE invoice_total > (payment_total + credit_total)
```

Vendors with names from A to L

```
WHERE vendor_name < 'M'
```

Invoices on or before a specified date

```
WHERE invoice_date <= '31-MAY-14'
```

Invoices on or after a specified date

```
WHERE invoice_date >= '01-MAY-14'
```

Invoices with credits that don't equal zero

```
WHERE credit_total <> 0
```

A compound condition without parentheses

```
SELECT invoice_number, invoice_date, invoice_total
FROM invoices
WHERE invoice_date > '01-MAY-2014' OR invoice_total > 500
    AND invoice_total - payment_total - credit_total > 0
ORDER BY invoice_number
```

	INVOICE_NUMBER	INVOICE_DATE	INVOICE_TOTAL
1	0-2058	08-MAY-14	37966.19
2	0-2060	08-MAY-14	23517.58
3	0-2436	07-MAY-14	10976.06

(91 rows selected)

The order of precedence for compound conditions

1. NOT
2. AND
3. OR

The same compound condition with parentheses

```
WHERE (invoice_date > '01-MAY-2014'
       OR invoice_total > 500)
    AND invoice_total - payment_total - credit_total > 0
ORDER BY invoice_number
```

	INVOICE_NUMBER	INVOICE_DATE	INVOICE_TOTAL
1	0-2436	07-MAY-14	10976.06
2	109596	14-JUN-14	41.8
3	111-92R-10092	04-JUN-14	46.21

(39 rows selected)

Tip: Just use parenthesis to be sure

The syntax of the WHERE clause with the LIKE operator

```
WHERE match_expression [NOT] LIKE pattern
```

Wildcard symbols

%

_

WHERE clauses that use the LIKE operator

Example 1

```
WHERE vendor_city LIKE 'SAN%'
```

Cities that will be retrieved

"San Diego" and "Santa Ana"

Example 2

```
WHERE vendor_name LIKE 'COMPU_ER%'
```

Vendors that will be retrieved

"Compuserve" and "Computerworld"

The syntax of the WHERE clause with the Is null condition

```
WHERE expression IS [NOT] NULL
```

The contents of the Null_Sample table

```
SELECT *
FROM null_sample
```

	INVOICE_ID	INVOICE_TOTAL
1	1	125
2	2	0
3	3	(null)
4	4	2199.99
5	5	0

The expanded syntax of the ORDER BY clause

```
ORDER BY expression [ASC|DESC] [, expression [ASC|DESC]]...
```

An ORDER BY clause that sorts by one column

```
SELECT vendor_name,
    vendor_city || ', ' || vendor_state || ' ' || vendor_zip_code
AS address
FROM vendors
ORDER BY vendor_name
```

	VENDOR_NAME	ADDRESS
1	ASC Signs	Fresno, CA 93703
2	AT&T	Phoenix, AZ 85062
3	Abbey Office Furnishings	Fresno, CA 93722

An ORDER BY clause that sorts by three columns

```
SELECT vendor_name,
    vendor_city || ', ' || vendor_state || ' ' ||
        vendor_zip_code AS address
FROM vendors
ORDER BY vendor_state, vendor_city, vendor_name
```

	VENDOR_NAME	ADDRESS
1	AT&T	Phoenix, AZ 85062
2	Computer Library	Phoenix, AZ 85023
3	Wells Fargo Bank	Phoenix, AZ 85038
4	Aztek Label	Anaheim, CA 92807
5	Blue Shield of California	Anaheim, CA 92850
6	Diversified Printing & Pub	Brea, CA 92621
7	ASC Signs	Fresno, CA 93703

An ORDER BY clause that uses an alias

```
SELECT vendor_name,
    vendor_city || ', ' || vendor_state || ' ' ||
        vendor_zip_code AS address
FROM vendors
ORDER BY address, vendor_name
```

	VENDOR_NAME	ADDRESS
1	Aztek Label	Anaheim, CA 92807
2	Blue Shield of California	Anaheim, CA 92850
3	Malloy Lithographing Inc	Ann Arbor, MI 48106
4	Data Reproductions Corp	Auburn Hills, MI 48326

An ORDER BY clause that uses an expression

```
SELECT vendor_name,
    vendor_city || ', ' || vendor_state || ' ' ||
        vendor_zip_code AS address
FROM vendors
ORDER BY vendor_contact_last_name
    || vendor_contact_first_name
```

	VENDOR_NAME	ADDRESS
1	Dristas Groom & McCormick	Fresno, CA 93720
2	Internal Revenue Service	Fresno, CA 93888
3	US Postal Service	Madison, WI 53707
4	Yale Industrial Trucks-Fresno	Fresno, CA 93706

An ORDER BY clause that uses column positions

```
SELECT vendor_name,
    vendor_city || ', ' || vendor_state || ' ' ||
        vendor_zip_code AS address
FROM vendors
ORDER BY 2, 1
```

	VENDOR_NAME	ADDRESS
1	Aztek Label	Anaheim, CA 92807
2	Blue Shield of California	Anaheim, CA 92850
3	Malloy Lithographing Inc	Ann Arbor, MI 48106
4	Data Reproductions Corp	Auburn Hills, MI 48326

The syntax of the row limiting clause (12c and later)

```
[ OFFSET offset { ROW | ROWS } ]
[ FETCH { FIRST | NEXT } [ { rowcount | percent PERCENT }
{ ROW | ROWS } { ONLY | WITH TIES } ]
```

A FETCH clause that retrieves the first five rows

```
SELECT vendor_id, invoice_total
FROM invoices
ORDER BY invoice_total DESC
FETCH FIRST 5 ROWS ONLY
```

	VENDOR_ID	INVOICE_TOTAL
1	110	37966.19
2	110	26881.4
3	110	23517.58
4	72	21842
5	110	20551.18

An OFFSET clause that starts with the third row and fetches three rows

```
SELECT invoice_id, vendor_id, invoice_total
FROM invoices
ORDER BY invoice_id
OFFSET 2 ROWS FETCH NEXT 3 ROWS ONLY
```

	INVOICE_ID	VENDOR_ID	INVOICE_TOTAL
1	3	110	20551.18
2	4	110	26881.4
3	5	81	936.93

An OFFSET clause that starts with the 101st row

```
SELECT invoice_id, vendor_id, invoice_total
FROM invoices
ORDER BY invoice_id
OFFSET 100 ROWS FETCH NEXT 1000 ROWS ONLY
```

	INVOICE_ID	VENDOR_ID	INVOICE_TOTAL
1	101	103	1367.5
2	102	48	856.92
3	103	95	19.67
4	104	114	290

A SELECT statement that retrieves rows with zero values

```
SELECT *
FROM null_sample
WHERE invoice_total = 0
```

	INVOICE_ID	INVOICE_TOTAL
1	2	0
2	5	0

A SELECT statement that retrieves rows with non-zero values

```
SELECT *
FROM null_sample
WHERE invoice_total <> 0
```

	INVOICE_ID	INVOICE_TOTAL
1	1	125
2	4	2199.99

A SELECT statement that retrieves rows with null values

```
SELECT *
FROM null_sample
WHERE invoice_total IS NULL
```

	INVOICE_ID	INVOICE_TOTAL
1	3	(null)

A SELECT statement that retrieves rows without null values

```
SELECT *
FROM null_sample
WHERE invoice_total IS NOT NULL
```

	INVOICE_ID	INVOICE_TOTAL
1	1	125
2	2	0
3	4	2199.99
4	5	0

The default sequence for an ascending sort

- Special characters
- Capital letters
- Lowercase letters
- Null values

Notes

- This causes problems when sorting mixed-case columns.
- Chapter 8 provides the solutions.

PART 2

Chapter 4

Select Using Joins – Select from Multiple Tables

The explicit syntax for an inner join (2 tables only)

```
SELECT column_list
FROM table_1 INNER JOIN table_2
        ON join_condition_1
```

NOTE: The "join_condition" defines what columns connect the tables (e.g. table1.pk = table1.fk)

A SELECT statement that joins two tables

```
SELECT invoice_number, vendor_name, invoice_total, payment_total
FROM vendors INNER JOIN invoices
        ON vendors.vendor_id = invoices.vendor_id
ORDER BY invoice_number
```

The result set

	INVOICE_NUMBER	VENDOR_NAME	INVOICE_TOTAL	PAYMENT_TOTAL
1	0-2058	Malloy Lithographing Inc	37966.19	37966.19
2	0-2060	Malloy Lithographing Inc	23517.58	21221.63
3	0-2436	Malloy Lithographing Inc	10976.06	0
4	1-200-5164	Federal Express Corporation	63.4	63.4
5	1-202-2978	Federal Express Corporation	33	33
6	10843	Yesmed, Inc	4901.26	4901.26
7	109596	Coffee Break Service	41.8	0

Inner join syntax

```
Select column_list
FROM table1 inner join table2
 ON table1.column = table2.column
```

e.g. Inner join 2 tables – pull all columns

```
SELECT *
FROM invoices inner join vendors
   ON invoices.vendor_id = vendors.vendor_id
```

Ven_id	Name	…
1	USPS	…
…	…	…
34	IBM	…
37	BCBS	…

Inv_id	Ven_id	Date	…
1	34	1/1	…
2	34	1/2	…
…	…	…	…
			…

Using a table alias

```
SELECT column_list
FROM table_1 n1 INNER JOIN table_2 n2
        ON n1.column_name operator n2.column_name
```

A SELECT statement that joins two tables

```
SELECT invoice_number, vendor_name, invoice_total, payment_total
FROM vendors v INNER JOIN invoices i
    ON v.vendor_id = i.vendor_id
ORDER BY invoice_number
```

ORA-00918: column ambiguously defined
00918. 00000 - "column ambiguously defined"

```
SELECT invoice_number, vendor_name, invoice_total, payment_total, v.vendor_id
FROM vendors v INNER JOIN invoices i
    ON v.vendor_id = i.vendor_id
ORDER BY invoice_number
```

e.g. Formatted code w/ clear aliases

```
SELECT    i.invoice_number,
          v.vendor_name,
          i.invoice_total,
          i.payment_total,
          v.vendor_id
FROM      vendors v INNER JOIN invoices i
            ON v.vendor_id = i.vendor_id
ORDER BY  i.invoice_number
```

Practice: Try combine expressions with inner join with aliases for all tables

```
SELECT   i.invoice_number,
         v.vendor_name,
         i.invoice_due_date,
         (invoice_total - payment_total - credit_total) AS "balance_due"
FROM vendors v JOIN invoices i
         ON v.vendor_id = i.vendor_id
WHERE (invoice_total - payment_total - credit_total) > 0
ORDER BY invoice_due_date DESC
```

The result set

	INVOICE_NUMBER	VENDOR_NAME	INVOICE_DUE_DATE	BALANCE_DUE
1	40318	Data Reproductions Corp	20-JUL-14	21842
2	39104	Data Reproductions Corp	20-JUL-14	85.31
3	0-2436	Malloy Lithographing Inc	17-JUL-14	10976.06

(40 rows selected)

Best Practice: Keep filtering out of FROM

An inner join with two conditions – Not best practice to include filters in FROM

```
SELECT invoice_number, invoice_date,
    invoice_total, line_item_amt
FROM invoices i JOIN invoice_line_items li
    ON (i.invoice_id = li.invoice_id) AND
       (i.invoice_total > li.line_item_amt)
ORDER BY invoice_number
```

The same join with one condition in a WHERE clause – Best practice

```
SELECT invoice_number, invoice_date,
    invoice_total, line_item_amt
FROM invoices i JOIN invoice_line_items li
    ON i.invoice_id = li.invoice_id
WHERE i.invoice_total > li.line_item_amt
ORDER BY invoice_number
```

	INVOICE_NUMBER	INVOICE_DATE	INVOICE_TOTAL	LINE_ITEM_AMT
1	97/522	30-APR-14	1962.13	765.13
2	97/522	30-APR-14	1962.13	1197
3	I77271-001	05-JUN-14	662	75.6
4	I77271-001	05-JUN-14	662	58.4

(6 rows selected)

A SELECT statement that joins four tables

```
SELECT vendor_name, invoice_number, invoice_date,
    line_item_amt, account_description
FROM vendors v
    JOIN invoices i ON v.vendor_id = i.vendor_id
    JOIN invoice_line_items li
        ON i.invoice_id = li.invoice_id
    JOIN general_ledger_accounts gl
        ON li.account_number = gl.account_number
WHERE (invoice_total - payment_total - credit_total) > 0
ORDER BY vendor_name, line_item_amt DESC
```

The result set

	VENDOR_NAME	INVOICE_NUMBER	INVOICE_DATE	LINE_ITEM_AMT	ACCOUNT_DESCRIPTION
1	Abbey Office Furnishings	203339-13	02-MAY-14	17.5	Office Supplies
2	Blue Cross	547481328	20-MAY-14	224	Group Insurance
3	Blue Cross	547480102	19-MAY-14	224	Group Insurance
4	Blue Cross	547479217	17-MAY-14	116	Group Insurance
5	Cardinal Business Media, Inc.	134116	01-JUN-14	90.36	Card Deck Advertising
6	Coffee Break Service	109596	14-JUN-14	41.8	Meals
7	Compuserve	21-4748363	09-MAY-14	9.95	Books, Dues, and Subscriptions
8	Computerworld	367447	31-MAY-14	2433	Card Deck Advertising

(44 rows selected)

Joining more than 2 tables

```
SELECT column_list
FROM table_1 n1 INNER JOIN table_2 n2 ON n1.column_name operator n2.column_name
                INNER JOIN table_3 n3 ON n2.column_name = n3.column_name
```

e.g. Joining 2+ tables with aliases

```
SELECT i.invoice_number, v.vendor_name, i.invoice_total, i.payment_total
FROM vendors v INNER JOIN invoices i ON v.vendor_id = i.vendor_id
               INNER JOIN invoice_line_items ili ON i.invoice_id = ili.invoice_id
ORDER BY invoice_number
```

The implicit syntax for an inner join

```
SELECT select_list
FROM table_1, table_2 [, table_3]...
WHERE table_1.column_name operator table_2.column_name
  [AND table_2.column_name operator table_3.column_name]...
```

Implicit syntax that joins two tables – simpler but puts the join in WHERE

```
SELECT invoice_number, vendor_name
FROM vendors v, invoices i
WHERE v.vendor_id = i.vendor_id
ORDER BY invoice_number
```

The result set

	INVOICE_NUMBER	VENDOR_NAME
1	0-2058	Malloy Lithographing Inc
2	0-2060	Malloy Lithographing Inc
3	0-2436	Malloy Lithographing Inc
4	1-200-5164	Federal Express Corporation
5	1-202-2978	Federal Express Corporation

(114 rows selected)

Implicit syntax that joins four tables

```
SELECT vendor_name, invoice_number, invoice_date,
    line_item_amt, account_description
FROM  vendors v, invoices i, invoice_line_items  li,
    general_ledger_accounts gl
WHERE v.vendor_id = i.vendor_id
  AND i.invoice_id = li.invoice_id
  AND li.account_number = gl.account_number
  AND (invoice_total - payment_total - credit_total) > 0
ORDER BY vendor_name, line_item_amt DESC
```

The result set

	VENDOR_NAME	INVOICE_NUMBER	INVOICE_DATE	LINE_ITEM_AMT	ACCOUNT_DESCRIPTION
1	Abbey Office Furnishings	203339-13	02-MAY-14	17.5	Office Supplies
2	Blue Cross	547481328	20-MAY-14	224	Group Insurance
3	Blue Cross	547480102	19-MAY-14	224	Group Insurance
4	Blue Cross	547479217	17-MAY-14	116	Group Insurance
5	Cardinal Business Media, Inc.	134116	01-JUN-14	90.36	Card Deck Advertising

(44 rows selected)

But…Best practice

- Joins in FROM
- Filters in WHERE

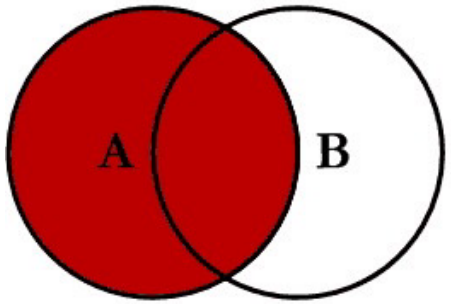
The explicit syntax for an outer join

```
SELECT select_list
FROM table_1
    {LEFT|RIGHT|FULL} [OUTER] JOIN table_2
        ON join_condition_1
   [{LEFT|RIGHT|FULL} [OUTER] JOIN table_3
        ON join_condition_2]...
```

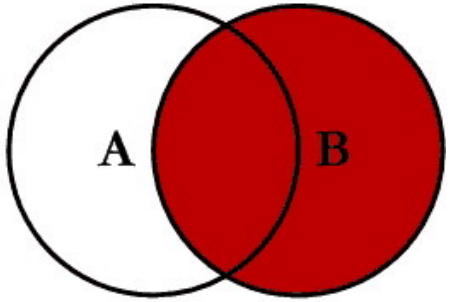
What outer joins do

Join	Keeps unmatched rows from
Left	The left table
Right	The right table
Full	Both tables

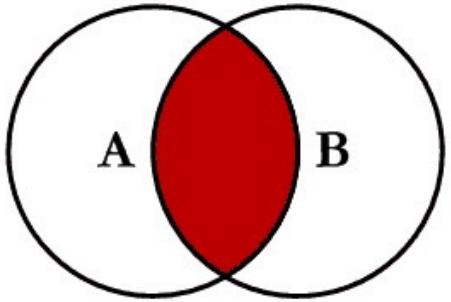
SQL JOINS



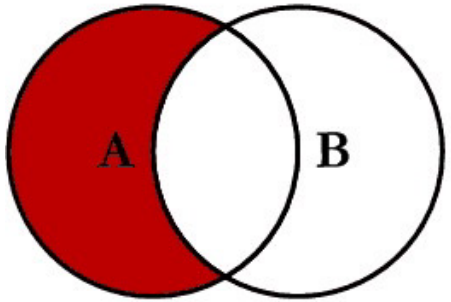

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



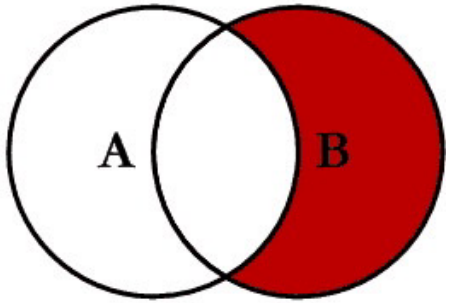

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



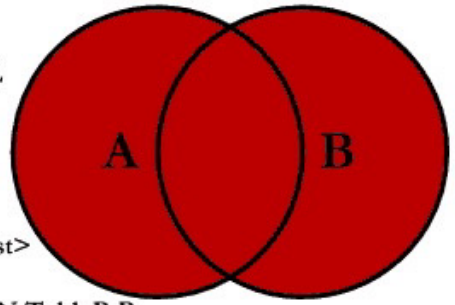

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



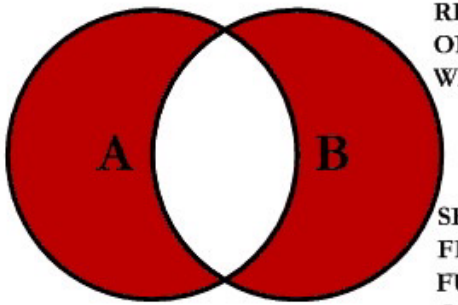

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

A SELECT statement that uses a left outer join

```
SELECT vendor_name, invoice_number, invoice_total
FROM vendors LEFT JOIN invoices
    ON vendors.vendor_id = invoices.vendor_id
ORDER BY vendor_name
```

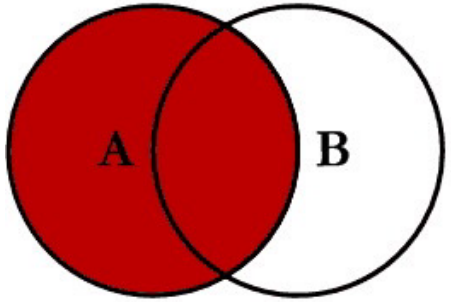
The result set

	VENDOR_NAME	INVOICE_NUMBER	INVOICE_TOTAL
1	ASC Signs	(null)	(null)
2	AT&T	(null)	(null)
3	Abbey Office Furnishings	203339-13	17.5
4	American Booksellers Assoc	(null)	(null)
5	American Express	(null)	(null)

(202 rows selected)

The Departments table

	DEPARTMENT_NUMBER	DEPARTMENT_NAME
1	1	Accounting
2	2	Payroll
3	3	Operations
4	4	Personnel
5	5	Maintenance



SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key

The Employees table

EMPLOYEE_ID	LAST_NAME	FIRST_NAME	DEPARTMENT_NUMBER
1	Smith	Cindy	2
2	Jones	Elmer	4
3	Simonian	Ralph	2
4	Hernandez	Olivia	1
5	Aaronsen	Robert	2
6	Watson	Denise	6
7	Hardy	Thomas	5
8	O'Leary	Rhea	4
9	Locario	Paulo	6

A left outer join

```
SELECT department_name AS dept_name,
    d.department_number AS dept_no,
    last_name
FROM departments d
    LEFT JOIN employees e
    ON d.department_number = e.department_number
ORDER BY department_name
```

Query results

	DEPT_NAME	DEPT_NO	LAST_NAME
1	Accounting	1	Hernandez
2	Maintenance	5	Hardy
3	Operations	3	(null)
4	Payroll	2	Simonian
5	Payroll	2	Aaronsen
6	Payroll	2	Smith
7	Personnel	4	Jones
8	Personnel	4	O'Leary

The Departments table

	DEPARTMENT_NUMBER	DEPARTMENT_NAME
1	1	Accounting
2	2	Payroll
3	3	Operations
4	4	Personnel
5	5	Maintenance

The Employees table

EMPLOYEE_ID	LAST_NAME	FIRST_NAME	DEPARTMENT_NUMBER
1	Smith	Cindy	2
2	Jones	Elmer	4
3	Simonian	Ralph	2
4	Hernandez	Olivia	1
5	Aaronsen	Robert	2
6	Watson	Denise	6
7	Hardy	Thomas	5
8	O'Leary	Rhea	4
9	Locario	Paulo	6

A right outer join – Not needed

```
SELECT department_name AS dept_name,
    e.department_number AS dept_no,
    last_name
FROM departments d
    RIGHT JOIN employees e
    ON d.department_number = e.department_number
ORDER BY department_name
```

Query results

	DEPT_NAME	DEPT_NO	LAST_NAME
1	Accounting	1	Hernandez
2	Maintenance	5	Hardy
3	Payroll	2	Smith
4	Payroll	2	Simonian
5	Payroll	2	Aaronsen
6	Personnel	4	O'Leary
7	Personnel	4	Jones
8	(null)	6	Watson
9	(null)	6	Locario

The Departments table

	DEPARTMENT_NUMBER	DEPARTMENT_NAME
1	1	Accounting
2	2	Payroll
3	3	Operations
4	4	Personnel
5	5	Maintenance

The Employees table

EMPLOYEE_ID	LAST_NAME	FIRST_NAME	DEPARTMENT_NUMBER
1	Smith	Cindy	2
2	Jones	Elmer	4
3	Simonian	Ralph	2
4	Hernandez	Olivia	1
5	Aaronsen	Robert	2
6	Watson	Denise	6
7	Hardy	Thomas	5
8	O'Leary	Rhea	4
9	Locario	Paulo	6

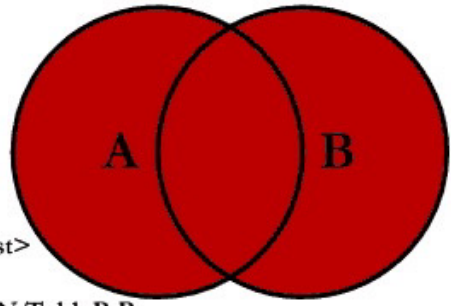
A full outer join

```
SELECT department_name AS dept_name,
    d.department_number AS d_dept_no,
    e.department_number AS e_dept_no,
    last_name
FROM departments d
    FULL JOIN employees e
    ON d.department_number = e.department_number
ORDER BY e.department_number, d.department_number
```

	DEPT_NAME	D_DEPT_NO	E_DEPT_NO	LAST_NAME
1	Accounting	1	1	Hernandez
2	Payroll	2	2	Simonian
3	Payroll	2	2	Smith
4	Payroll	2	2	Aaronsen
5	Personnel	4	4	O'Leary
6	Personnel	4	4	Jones
7	Maintenance	5	5	Hardy
8	(null)	(null)	6	Watson
9	(null)	(null)	6	Locario
10	Operations	3	(null)	(null)

A SELECT statement that uses full outer joins

```
SELECT department_name, last_name, project_number AS proj_no
FROM departments dpt
    FULL JOIN employees emp
        ON dpt.department_number = emp.department_number
    FULL JOIN projects prj
        ON emp.employee_id = prj.employee_id
ORDER BY department_name
```



	DEPARTMENT_NAME	LAST_NAME	PROJ_NO
1	Accounting	Hernandez	P1011
2	Maintenance	Hardy	(null)
3	Operations	(null)	(null)
4	Payroll	Simonian	P1012
5	Payroll	Aaronsen	P1012
6	Payroll	Smith	P1012
7	Personnel	Jones	(null)
8	Personnel	O'Leary	P1011
9	(null)	Locario	P1013
10	(null)	(null)	P1014
11	(null)	Watson	P1013

Let's practice FULL OUTER join:

1. Join Orders, Order_Items, Products, and Categories.
2. Return Order_ID, Product_Name, Category_Name
3. Test: What changes when you use INNER versus OUTER join? Try both

The syntax for a union

```
    SELECT_statement_1
UNION [ALL]
    SELECT_statement_2
[UNION [ALL]
    SELECT_statement_3]...
[ORDER BY order_by_list]
```

Rules for a union

- The number of columns must be the same in all SELECTs.
- The column data types must be compatible.
- The column names are taken from the first SELECT statement.

e.g. A union with data from just Invoices table

```
    SELECT 'Active' AS source, invoice_number, invoice_date,
        invoice_total
    FROM invoices
    WHERE (invoice_total - payment_total - credit_total) > 0
UNION
    SELECT 'Paid' AS source, invoice_number, invoice_date,
        invoice_total
    FROM invoices
    WHERE (invoice_total - payment_total - credit_total)
          <= 0
ORDER BY invoice_total DESC
```

The result set

	SOURCE	INVOICE_NUMBER	INVOICE_DATE	INVOICE_TOTAL
1	Paid	0-2058	08-MAY-14	37966.19
2	Paid	P-0259	16-APR-14	26881.4
3	Paid	0-2060	08-MAY-14	23517.58
4	Active	40318	18-JUL-14	21842
5	Active	P-0608	11-APR-14	20551.18
6	Active	0-2436	07-MAY-14	10976.06

The syntax for MINUS and INTERSECT operations

```
    SELECT_statement_1
{MINUS|INTERSECT}
    SELECT_statement_2
[ORDER BY order_by_list]
```

Self Joins: A way of finding in common records on a table

A self-join that returns vendors from cities in common with other vendors

```
SELECT DISTINCT v1.vendor_name, v1.vendor_city,
    v1.vendor_state
FROM vendors v1 JOIN vendors v2
    ON (v1.vendor_city = v2.vendor_city) AND
       (v1.vendor_state = v2.vendor_state) AND
       (v1.vendor_id <> v2.vendor_id)
ORDER BY v1.vendor_state, v1.vendor_city
```

The result set

	VENDOR_NAME	VENDOR_CITY	VENDOR_STATE
1	AT&T	Phoenix	AZ
2	Computer Library	Phoenix	AZ
3	Wells Fargo Bank	Phoenix	AZ
4	Aztek Label	Anaheim	CA
5	Blue Shield of California	Anaheim	CA
6	ASC Signs	Fresno	CA
7	Abbey Office Furnishings	Fresno	CA
8	BFI Industries	Fresno	CA

(84 rows selected)

JUST FYI:

- Not going to practice this
- We'll learn a different way to do this later

A SELECT statement that uses left outer joins

```
SELECT department_name, last_name, project_number AS proj_no
FROM departments d
    LEFT JOIN employees e
        ON d.department_number = e.department_number
    LEFT JOIN projects p
        ON e.employee_id = p.employee_id
ORDER BY department_name, last_name, project_number
```

	DEPARTMENT_NAME	LAST_NAME	PROJ_NO
1	Accounting	Hernandez	P1011
2	Maintenance	Hardy	(null)
3	Operations	(null)	(null)
4	Payroll	Aaronsen	P1012
5	Payroll	Simonian	P1012
6	Payroll	Smith	P1012
7	Personnel	Jones	(null)
8	Personnel	O'Leary	P1011

A SELECT statement that uses full outer joins

```
SELECT department_name, last_name, project_number AS proj_no
FROM departments dpt
    FULL JOIN employees emp
        ON dpt.department_number = emp.department_number
    FULL JOIN projects prj
        ON emp.employee_id = prj.employee_id
ORDER BY department_name
```

	DEPARTMENT_NAME	LAST_NAME	PROJ_NO
1	Accounting	Hernandez	P1011
2	Maintenance	Hardy	(null)
3	Operations	(null)	(null)
4	Payroll	Simonian	P1012
5	Payroll	Aaronsen	P1012
6	Payroll	Smith	P1012
7	Personnel	Jones	(null)
8	Personnel	O'Leary	P1011
9	(null)	Locario	P1013
10	(null)	(null)	P1014
11	(null)	Watson	P1013

A SELECT statement with an outer and inner join

```
SELECT department_name AS dept_name,
       last_name, project_number
FROM departments dpt
    JOIN employees emp
        ON dpt.department_number = emp.department_number
    LEFT JOIN projects prj
        ON emp.employee_id = prj.employee_id
ORDER BY department_name
```

The result set

	DEPT_NAME	LAST_NAME	PROJECT_NUMBER
1	Accounting	Hernandez	P1011
2	Maintenance	Hardy	(null)
3	Payroll	Simonian	P1012
4	Payroll	Smith	P1012
5	Payroll	Aaronsen	P1012
6	Personnel	Jones	(null)
7	Personnel	O'Leary	P1011

(7 rows selected)

Terms to know

- **Outer join**
- **Left outer join**
- **Right outer join**
- Equi-join
- Natural join
- Cross join

The syntax for a join with the USING keyword

```
SELECT select_list
FROM table_1
    [{LEFT|RIGHT|FULL} [OUTER]] JOIN table_2
        USING(join_column_1[, join_column_2]...)
   [[{LEFT|RIGHT|FULL} [OUTER]] JOIN table_3
        USING (join_column_2[, join_column_2]...)]...
```

A SELECT statement with the USING keyword

```
SELECT invoice_number, vendor_name
FROM vendors
    JOIN invoices USING (vendor_id)
ORDER BY invoice_number
```

The result set

	INVOICE_NUMBER	VENDOR_NAME
1	0-2058	Malloy Lithographing Inc
2	0-2060	Malloy Lithographing Inc
3	0-2436	Malloy Lithographing Inc
4	1-200-5164	Federal Express Corporation

(114 rows selected)

JUST FYI:

- Removes the need of the “ON column1 = column2” syntax
- Only works if joining columns have same name
- Best to just learn full syntax and stick to using “ON” keyword.

The syntax for a join with the NATURAL keyword

```
SELECT select_list
FROM table_1
    NATURAL JOIN table_2
    [NATURAL JOIN table_3]...
```

A SELECT statement with the NATURAL keyword

```
SELECT invoice_number, vendor_name
FROM vendors
    NATURAL JOIN invoices
ORDER BY invoice_number
```

The result set

	INVOICE_NUMBER	VENDOR_NAME
1	0-2058	Malloy Lithographing Inc
2	0-2060	Malloy Lithographing Inc
3	0-2436	Malloy Lithographing Inc
4	1-200-5164	Federal Express Corporation

(114 rows selected)

JUST FYI:

- Simpler/lazy way to join
- Only works if there is a single column in common between the two joined tables.
- Best to just learn full syntax and stick to using “INNER” and “ON” keywords.

How to code a cross join with the explicit syntax

The explicit syntax for a cross join

```
SELECT select_list
FROM table_1 CROSS JOIN table_2
```

A cross join that uses the explicit syntax

```
SELECT departments.department_number, department_name,
    employee_id, last_name
FROM departments CROSS JOIN employees
ORDER BY departments.department_number
```

The result set

	DEPARTMENT_NUMBER	DEPARTMENT_NAME	EMPLOYEE_ID	LAST_NAME
1	1	Accounting	4	Hernandez
2	1	Accounting	3	Simonian
3	1	Accounting	9	Locario
4	1	Accounting	8	O'Leary
5	1	Accounting	7	Hardy
6	1	Accounting	6	Watson
7	1	Accounting	5	Aaronsen

(45 rows selected)

How to code a cross join with the implicit syntax

The implicit syntax for a cross join

```
SELECT select_list
FROM table_1, table_2
```

A cross join that uses the implicit syntax

```
SELECT departments.department_number, department_name,
    employee_id, last_name
FROM departments, employees
ORDER BY departments.department_number
```

The result set

	DEPARTMENT_NUMBER	DEPARTMENT_NAME	EMPLOYEE_ID	LAST_NAME
1	1	Accounting	4	Hernandez
2	1	Accounting	3	Simonian
3	1	Accounting	9	Locario
4	1	Accounting	8	O'Leary
5	1	Accounting	7	Hardy
6	1	Accounting	6	Watson
7	1	Accounting	5	Aaronsen

(45 rows selected)

JUST FYI:

- Cross joins typically won't happen if you avoid use of implicit joins
- Just be aware of what a Cartesian join can look like so you can know how to correct it

The syntax for a union

```
    SELECT_statement_1
UNION [ALL]
    SELECT_statement_2
[UNION [ALL]
    SELECT_statement_3]...
[ORDER BY order_by_list]
```

Rules for a union

- The number of columns must be the same in all SELECTs.
- The column data types must be compatible.
- The column names are taken from the first SELECT statement.

A union with data from two different tables

```
    SELECT 'Active' AS source, invoice_number, invoice_date,
        invoice_total
    FROM active_invoices
    WHERE invoice_date >= '01-JUN-2014'
UNION
    SELECT 'Paid' AS source, invoice_number, invoice_date,
        invoice_total
    FROM paid_invoices
    WHERE invoice_date >= '01-JUN-2014'
ORDER BY invoice_total DESC
```

The result set

	SOURCE	INVOICE_NUMBER	INVOICE_DATE	INVOICE_TOTAL
1	Active	40318	18-JUL-14	21842
2	Paid	P02-3772	03-JUN-14	7125.34
3	Paid	10843	04-JUN-14	4901.26
4	Paid	77290	04-JUN-14	1750
5	Paid	RTR-72-3662-X	04-JUN-14	1600
6	Paid	75C-90227	06-JUN-14	1367.5
7	Paid	P02-88D77S7	06-JUN-14	856.92
8	Active	I77271-001	05-JUN-14	662
9	Active	9982771	03-JUN-14	503.2

```
(22 rows selected)
```

A union with data from just the Invoices table

```
    SELECT 'Active' AS source, invoice_number, invoice_date,
        invoice_total
    FROM invoices
    WHERE (invoice_total - payment_total - credit_total) > 0
UNION
    SELECT 'Paid' AS source, invoice_number, invoice_date,
        invoice_total
    FROM invoices
    WHERE (invoice_total - payment_total - credit_total)
          <= 0
ORDER BY invoice_total DESC
```

The result set

	SOURCE	INVOICE_NUMBER	INVOICE_DATE	INVOICE_TOTAL
1	Paid	0-2058	08-MAY-14	37966.19
2	Paid	P-0259	16-APR-14	26881.4
3	Paid	0-2060	08-MAY-14	23517.58
4	Active	40318	18-JUL-14	21842
5	Active	P-0608	11-APR-14	20551.18
6	Active	0-2436	07-MAY-14	10976.06

(114 rows selected)

A union with payment data from the same tables

```
    SELECT invoice_number, vendor_name,
        '33% Payment' AS payment_type,
        invoice_total AS total,
        (invoice_total * 0.333) AS payment
    FROM invoices JOIN vendors
        ON invoices.vendor_id = vendors.vendor_id
    WHERE invoice_total > 10000
UNION
    SELECT invoice_number, vendor_name,
        '50% Payment' AS payment_type,
        invoice_total AS total,
        (invoice_total * 0.5) AS payment
    FROM invoices JOIN vendors
        ON invoices.vendor_id = vendors.vendor_id
    WHERE invoice_total BETWEEN 500 AND 10000
```

The union (continued)

```
UNION
    SELECT invoice_number, vendor_name,
        'Full amount' AS payment_type,
        invoice_total AS Total, invoice_total AS Payment
    FROM invoices JOIN vendors
        ON invoices.vendor_id = vendors.vendor_id
    WHERE invoice_total < 500
ORDER BY payment_type, vendor_name, invoice_number
```

The result set

	INVOICE_NUMBER	VENDOR_NAME	PAYMENT_TYPE	TOTAL	PAYMENT
1	40318	Data Reproductions Corp	33% Payment	21842	7273.386
2	0-2058	Malloy Lithographing Inc	33% Payment	37966.19	12642.74127
3	0-2060	Malloy Lithographing Inc	33% Payment	23517.58	7831.35414
4	0-2436	Malloy Lithographing Inc	33% Payment	10976.06	3655.02798
5	P-0259	Malloy Lithographing Inc	33% Payment	26881.4	8951.5062
6	P-0608	Malloy Lithographing Inc	33% Payment	20551.18	6843.54294
7	509786	Bertelsmann Industry Svcs. Inc	50% Payment	6940.25	3470.125

(114 rows selected)

The syntax for MINUS and INTERSECT operations

```
    SELECT_statement_1
{MINUS|INTERSECT}
    SELECT_statement_2
[ORDER BY order_by_list]
```

The Customers table

CUSTOMER_LAST_NAME	CUSTOMER_FIRST_NAME
Anders	Maria
Trujillo	Ana
Moreno	Antonio
Hardy	Thomas
Berglund	Christina
Moos	Hanna

(24 rows selected)

The Employees table

EMPLOYEE_ID	LAST_NAME	FIRST_NAME	DEPARTMENT_NUMBER
1	Smith	Cindy	2
2	Jones	Elmer	4
3	Simonian	Ralph	2
4	Hernandez	Olivia	1
5	Aaronsen	Robert	2
6	Watson	Denise	6
7	Hardy	Thomas	5
8	O'Leary	Rhea	4
9	Locario	Paulo	6

(9 rows selected)

A query that excludes rows from the first query if they also occur in the second query

```
    SELECT customer_first_name, customer_last_name
    FROM customers
MINUS
    SELECT first_name, last_name
    FROM employees
ORDER BY customer_last_name
```

The result set

	CUSTOMER_FIRST_NAME	CUSTOMER_LAST_NAME
1	Maria	Anders
2	Christina	Berglund
3	Art	Braunschweiger
4	Donna	Chelan

```
(23 rows selected)
```

A query that only includes rows that occur in both queries

```
    SELECT customer_first_name, customer_last_name
    FROM customers
INTERSECT
    SELECT first_name, last_name
    FROM employees
```

The result set

	CUSTOMER_FIRST_NAME	CUSTOMER_LAST_NAME
1	Thomas	Hardy

```
(1 rows selected)
```