SIT103/SIT772 Database Fundamentals

Week 6

Relational Algebra and Join

Dr Iynkaran Natgunanathan,
email:
iynkaran.natgunanathan@deakin.edu.au,
Phone: +61 3 924 68825.

Important Information

- Extensions and task submissions
  - we can't allow extension for more than a week via OnTrack

    - note that due dates are only for feedback, you can submit tasks even after that

    - If you submit after the due date, your tasks will be assessed any time before or at the end of the trimester.

    - please continue submitting tasks even if you missed due dates

Last Week

- Introduction to SQL: DML, DDL, TCL, DCL
- DML – `SELECT` queries

  `FROM`, `WHERE`, `ORDER BY`, `GROUP BY`, `HAVING`, `AS`, `DISTINCT`,
- Arithmetic, Comparison, Logical and Special operators
- Wildcards
- Aggregate functions
- Subqueries – nested queries

Last Week's OnTrack Tasks

- 5.1P Basic SQL

  - `SELECT` queries

- 5.2C Online Quiz 1

  - Hope you completed the Quiz 1 via the CloudDeakin

  - Submitted the screenshot of your results with score equal to or more than 16/20 via OnTrack

Questions?

Any questions/comments so far

Last week's content

Anything in general

Any OnTack tasks

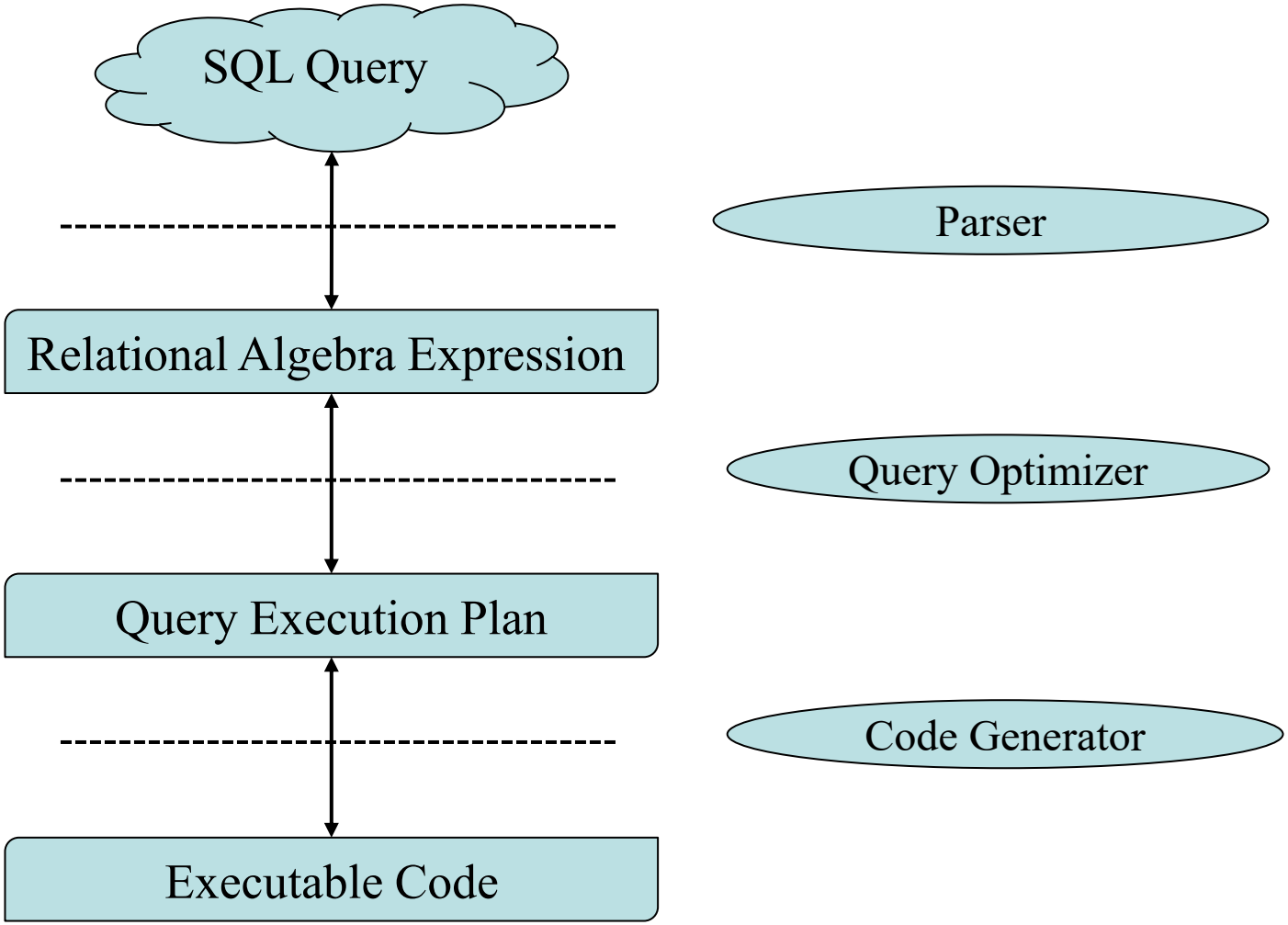
This week

- Relational Algebra
- Joins
- SQL Functions

Relational Algebra

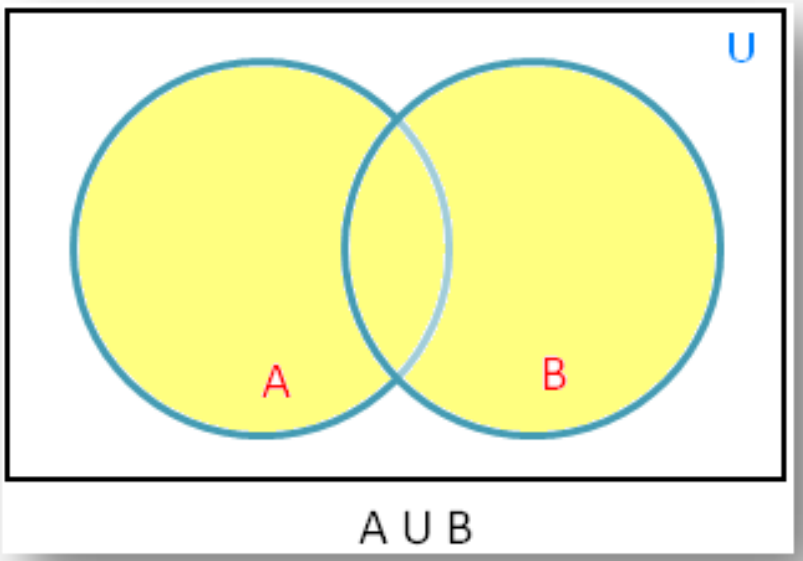
- Defines a theoretical/mathematical principles of manipulating table contents
  - procedural query language – user must specify **what** they want and how to get it
  - form the basis/foundation for relational database and SQL
- Relational operators on existing tables (relations) to produce new tables that contain required data
- Relational Operators
  - UNION, INTERSECT, DIFFERENCE, PRODUCT, SELECT, PROJECT, and JOIN

The role of Relational Algebra



Union Set Operator

- **Set Operators**
  - Union (U)



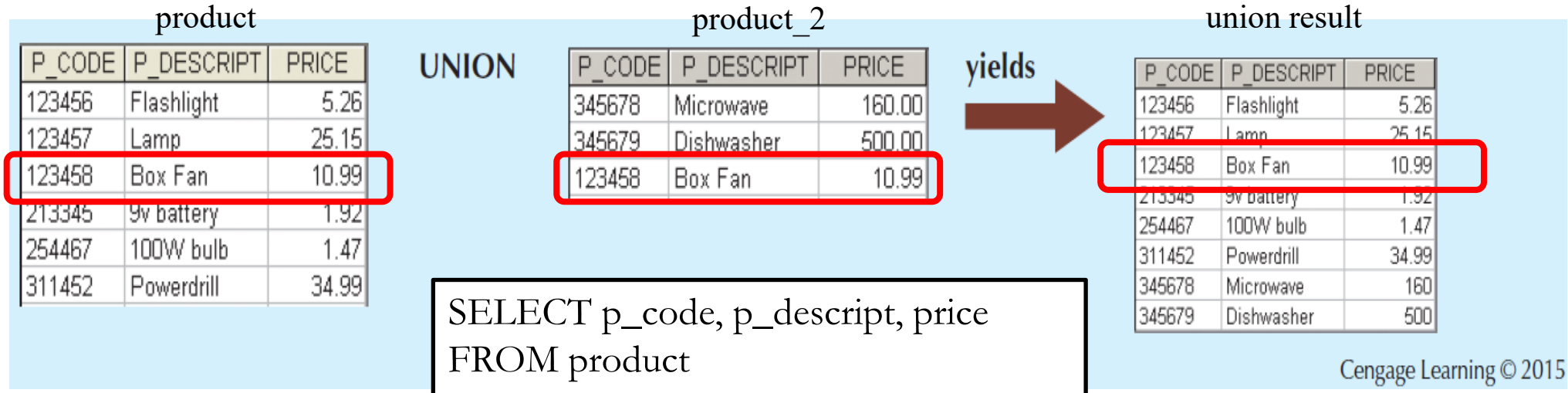
A = {1, a, 3}
B = {a, b, c}

A U B = {1, a, 3, b, c}

http://www.math-only-math.com/union-of-sets-using-venn-diagram.html

Relational UNION

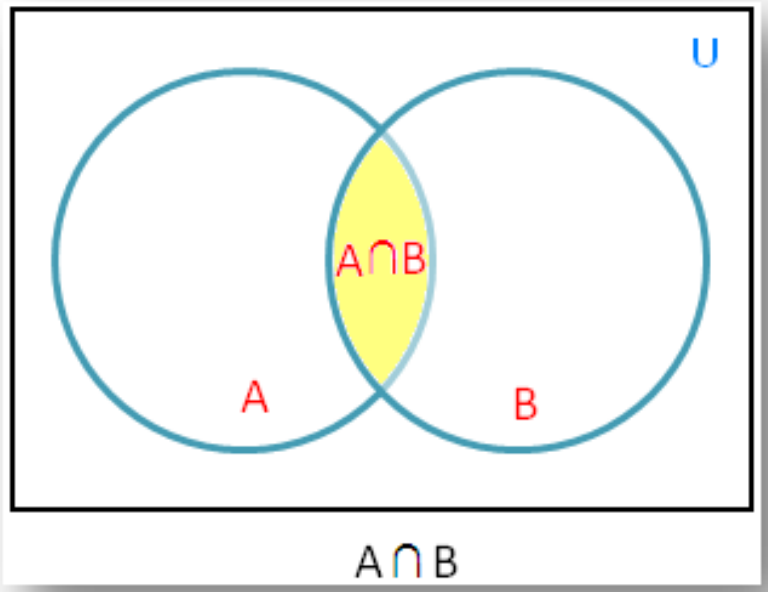
- Combines all rows from two tables, **excluding duplicate rows**
- Tables must be ***union-compatible***, i.e., tables share the same number of columns




```
SELECT p_code, p_descript, price
FROM product
UNION
SELECT p_code, p_descript, price
FROM product_2;
```

INTERSECT Set operator

- **Set Operators**
  - Intersection (∩)



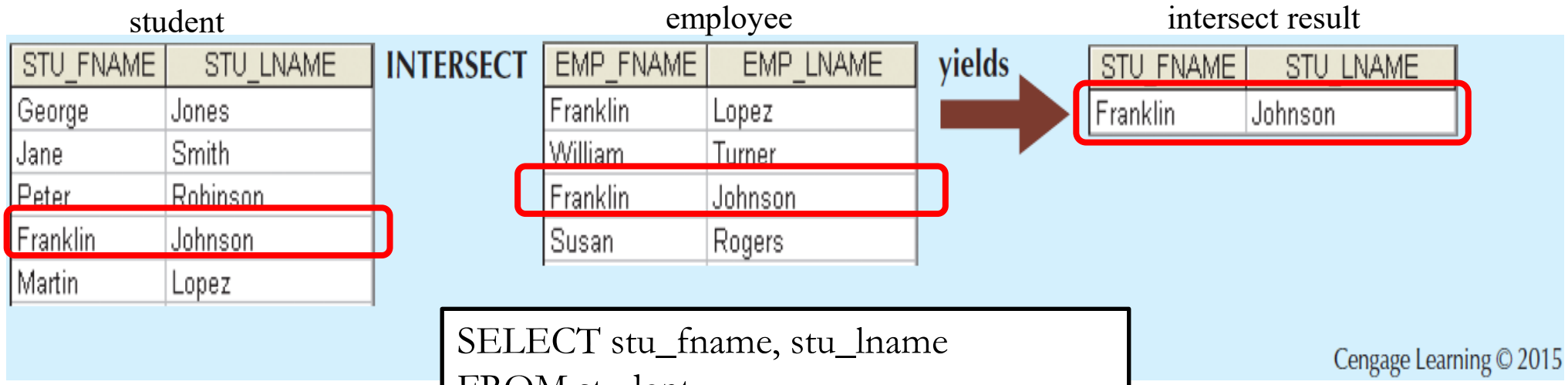
A = {1, a, 3}
B = {a, b, c}

A ∩ B = {a}

http://www.math-only-math.com/union-of-sets-using-venn-diagram.html

Relational INTERSECT

- Yields only the rows that appear in both tables
- Tables must be intersect-compatible



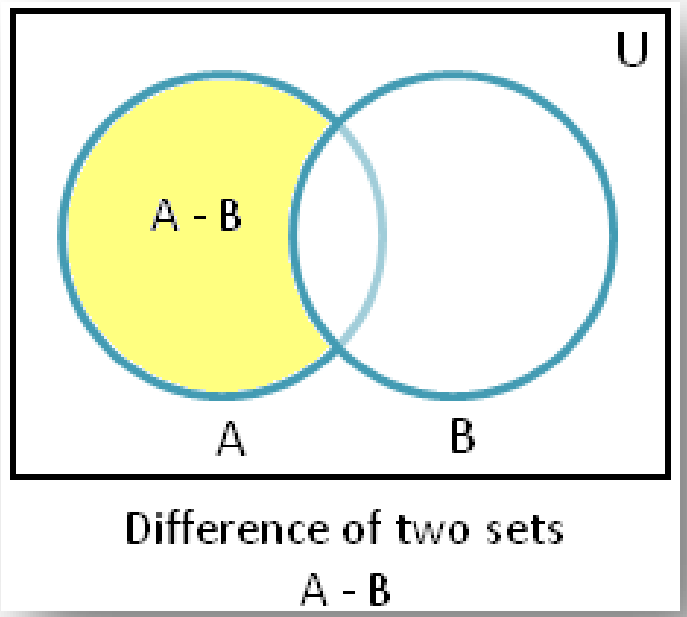

```
SELECT stu_fname, stu_lname
FROM student
INTERSECT
SELECT emp_fname, emp_lname
FROM employee;
```

Set DIFFERENCE Operator

- **Set Operators**
  - Difference (\)
  - A\B or A-B

A = {1, a, 3}
B = {a, b, c}

A \ B = {1, 3}
B \ A = {b, c}

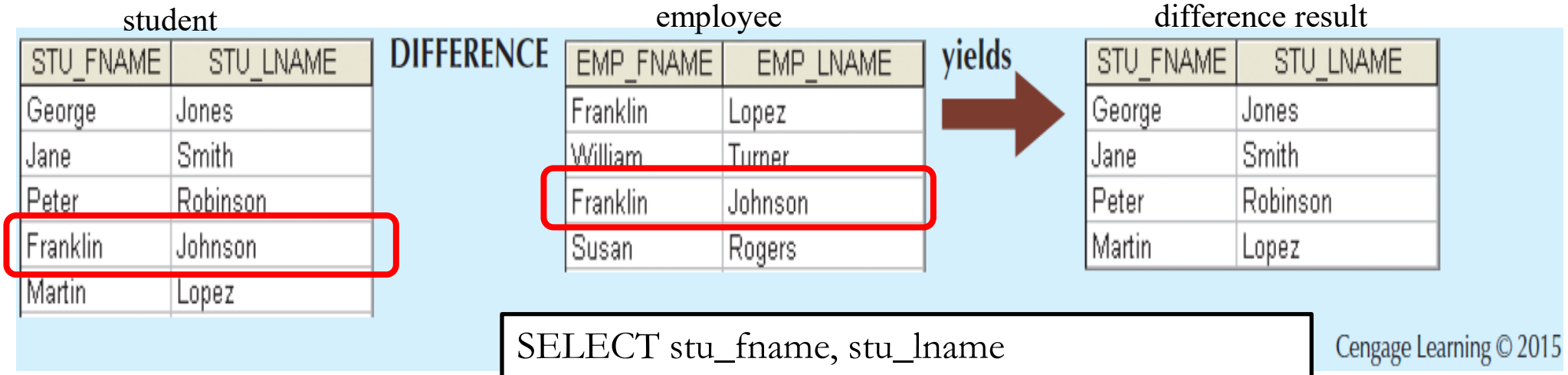


http://www.math-only-math.com/union-of-sets-using-venn-diagram.html

$$A \setminus B = A - B = \{x \in U : x \in A \text{ and } x \notin B\}$$

Relational DIFFERENCE

- Yields all rows in the 1st table not found in the 2nd table, i.e., it subtracts/removes common rows from both from the 1st table
- Tables must be difference-compatible



SQL equivalent operator →

```
SELECT stu_fname, stu_lname
FROM student
MINUS
SELECT emp_fname, emp_lname
FROM employee;
```

Not Supported by MYSQL!

Set PRODUCT Operator

- **Cartesian Product**
  - Product (X)
  - A X B

Figure Four – Cartesian Product

$$A = \{1,2,3\}$$
$$B = \{3,4,5\}$$
$$A \times B = \begin{Bmatrix} (1,3), (2,3), (3,3), \\ (1,4), (2,4), (3,4), \\ (1,5), (2,5), (3,5) \end{Bmatrix}$$

		B		
		3	4	5
A	1	(1,3)	(1,4)	(1,5)
	2	(2,3)	(2,4)	(2,5)
	3	(3,3)	(3,4)	(3,5)

A = {1, 2, 3}
B = {3, 4, 5}

A X B = {(1, 3), (1, 4), (1,5), (2,3), (2,4), (2, 5), (3, 3), (3, 4), (3, 5) }

http://hideoushumpbackfreak.com/post/2017/02/22/Just-Enough-Set-Theory-When-Sets-Collide-(Part-3-of-3)

Relational PRODUCT

- Yields all possible pairs of rows from two tables (a.k.a. **Cartesian Product** or **CROSS JOIN** in SQL)

product

P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

PRODUCT

store

STORE	AISLE	SHELF
23	W	5
24	K	9
25	Z	6

yields

product result

P_CODE	P_DESCRIPT	PRICE	STORE	AISLE	SHELF
123456	Flashlight	5.26	23	W	5
123456	Flashlight	5.26	24	K	9
123456	Flashlight	5.26	25	Z	6
123457	Lamp	25.15	23	W	5
123457	Lamp	25.15	24	K	9
123457	Lamp	25.15	25	Z	6
123458	Box Fan	10.99	23	W	5
123458	Box Fan	10.99	24	K	9
123458	Box Fan	10.99	25	Z	6
213345	9v battery	1.92	23	W	5
213345	9v battery	1.92	24	K	9
213345	9v battery	1.92	25	Z	6
311452	Powerdrill	34.99	23	W	5
311452	Powerdrill	34.99	24	K	9
311452	Powerdrill	34.99	25	Z	6
254467	100W bulb	1.47	23	W	5
254467	100W bulb	1.47	24	K	9
254467	100W bulb	1.47	25	Z	6

SQL equivalent operator

```
SELECT * FROM PRODUCT
CROSS JOIN STORE;
```

```
SELECT * FROM PRODUCT, STORE;
```

Relational SELECT operator (σ)

- Yields either all rows e.g. `SELECT * FROM product` or those rows matching a specified criterion e.g.

Original table

P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

SELECT ALL yields

SELECT * FROM PRODUCT

New table

P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

SELECT only PRICE less than $2.00 yields

SELECT * FROM PRODUCT WHERE (price < 2);

P_CODE	P_DESCRIPT	PRICE
213345	9v battery	1.92
254467	100W bulb	1.47

SELECT only P_CODE = 311452 yields

SELECT * FROM PRODUCT WHERE (p_code = '311452');

P_CODE	P_DESCRIPT	PRICE
311452	Powerdrill	34.99

Relational Algebra notation:
$\sigma_{p\_code = '311452'}$ (product)

Relational PROJECT operator (π)

- Yields all values for selected attributes
- Unary operator that yields a vertical subset of a table

Original table

P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

PROJECT PRICE yields

```
SELECT PRICE
FROM  PRODUCT;
```

New table

PRICE
5.26
25.15
10.99
1.92
1.47
34.99

PROJECT P_DESCRIPT and PRICE yields

```
SELECT P_DESCRIPT, PRICE
FROM  PRODUCT;
```

P_DESCRIPT	PRICE
Flashlight	5.26
Lamp	25.15
Box Fan	10.99
9v battery	1.92
100W bulb	1.47
Powerdrill	34.99

PROJECT P_CODE and PRICE yields

```
SELECT P_CODE, PRICE FROM
PRODUCT;
```

P_CODE	PRICE
123456	5.26
123457	25.15
123458	10.99
213345	1.92
254467	1.47
311452	34.99

Relational Algebra notation:
$\pi_{p\_code,\, price}$ (product)

The relational JOIN operator ⋈

- "Real power" behind the relational database because JOIN can **combine data** from **two or more tables linked by common attributes**

FIGURE 3.10 TWO TABLES THAT WILL BE USED IN JOIN ILLUSTRATIONS

Table name: CUSTOMER

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE
1132445	Walker	32145	231
1217782	Adares	32145	125
1312243	Rakowski	34129	167
1321242	Rodriguez	37134	125
1542311	Smithson	37134	421
1657399	Vanloo	32145	231

Table name: AGENT

AGENT_CODE	AGENT_PHONE
125	6152439887
167	6153426778
231	6152431124
333	9041234445

- Types of JOIN
  - Inner Join (Natural Join, Equi-Join, Theta Join)
  - Outer Join (Full Join, Left Join, Right Join)

Inner Join

- Returns records matching the condition(s) only from tables that are joined.
- **Natural Join**: links tables by selecting only the rows with common values in their common attribute(s)
  - implicit join based on attribute(s) with the same domain(s) and name(s)
- **Equi-Join**: links tables on the basis of an equality condition that compares specified column(s) of each table
  - explicit join based on attribute(s) sharing common values
  - uses the comparison operator '='
  - commonly used Join
  - Natural Join is a special case of Equi-Join
- **Theta Join**: links tables using an inequality comparison operator (<,>,<=,>=)

Outer Join

- Matched pairs are retained and unmatched values in the other tables are left NULL
  - **Full Outer Join:** yields all of the rows in both table, including those that do not have matching values, the columns where value do not exist are left NULL.
  - **Left Outer Join**: yields all of the rows in the first table, including those that do not have a matching value in the second table
  - **Right Outer Join**: yields all of the rows in the second table, including those that do not have matching values in the first table

JOIN Operator

- JOIN involves: (i) PRODUCT, (ii) SELECT, and (iii) PROJECT

PROJECT

Selects required columns

SELECT

Selects rows based on values of matching columns

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
1132445	Walker	32145	231	125	6152439887
1132445	Walker	32145	231	167	6153426778
1132445	Walker	32145	231	231	6152431124
1132445	Walker	32145	231	333	9041234445
1217782	Adares	32145	125	125	6152439887
1217782	Adares	32145	125	167	6153426778
1217782	Adares	32145	125	231	6152431124
1217782	Adares	32145	125	333	9041234445
1312243	Rakowski	34129	167	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1312243	Rakowski	34129	167	231	6152431124
1312243	Rakowski	34129	167	333	9041234445
1321242	Rodriguez	37134	125	125	6152439887
1321242	Rodriguez	37134	125	167	6153426778
1321242	Rodriguez	37134	125	231	6152431124
1321242	Rodriguez	37134	125	333	9041234445
1542311	Smithson	37134	421	125	6152439887
1542311	Smithson	37134	421	167	6153426778
1542311	Smithson	37134	421	231	6152431124
1542311	Smithson	37134	421	333	9041234445
1657399	Vanloo	32145	231	125	6152439887
1657399	Vanloo	32145	231	167	6153426778
1657399	Vanloo	32145	231	231	6152431124
1657399	Vanloo	32145	231	333	9041234445

Inner Join: JOIN ON

- *Syntax:* SELECT *column-list* FROM *table1* [INNER] JOIN *table2* ON *join-condition*
- Example:

```
SELECT CUSTOMER.CUS_CODE, CUSTOMER.CUS_LNAME, AGENT.PHONE FROM
CUSTOMER INNER JOIN AGENT ON CUSTOMER.AGENT_CODE = AGENT.AGENT_CODE
```

- Join-condition:
  - CUSTOMER.AGENT_CODE = AGENT.AGENT_CODE **[Natural Join]**
  - CUSTOMER.AGENT_CODE > AGENT.AGENT_CODE **[Theta Join]**
  - CUSTOMER.A_CODE = AGENT.AGENT_CODE **[Equi-Join]**

    - Attributes' names used to join can be different but their values must be comparable

- WHERE and other clauses can be used to restrict rows, example:

```
SELECT CUSTOMER.CUS_CODE, CUSTOMER.CUS_LNAME, AGENT.PHONE FROM
CUSTOMER INNER JOIN AGENT ON CUSTOMER.AGENT_CODE = AGENT.AGENT_CODE
WHERE AGENT.AGENT_CODE = 123 ORDER BY CUSTOMER.CUS_LNAME;
```

JOIN ON Example

PRODUCT

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-17	8	5	109.99	0.00	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-17	32	15	14.99	0.05	21344
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-17	18	12	17.49	0.00	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-18	15	8	39.95	0.00	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-18	23	5	43.99	0.00	23119
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-17	8	5	109.92	0.05	24288
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-17	6	5	99.87	0.05	24288
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-18	12	5	38.95	0.05	25595
23109-HB	Claw hammer	20-Jan-18	23	10	9.95	0.10	21225
23114-AA	Sledge hammer, 12 lb.	02-Jan-18	8	5	14.40	0.05	
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-17	43	20	4.99	0.00	21344
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-18	11	5	256.99	0.05	24288
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-18	188	75	5.87	0.00	
SM-18277	1.25-in. metal screw, 25	01-Mar-18	172	75	6.99	0.00	21225
SW-23116	2.5-in. wd. screw, 50	24-Feb-18	237	100	8.45	0.00	21231
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-18	18	5	119.95	0.10	25595

VENDOR

V_CODE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE	V_STATE	V_ORDER
21225	Bryson, Inc.	Smithson	615	223-3234	TN	Y
21226	SuperLoo, Inc.	Flushing	904	215-8995	FL	N
21231	D&E Supply	Singh	615	228-3245	TN	Y
21344	Gomez Bros.	Ortega	615	889-2546	KY	N
22567	Dome Supply	Smith	901	678-1419	GA	N
23119	Randsets Ltd.	Anderson	901	678-3998	GA	Y
24004	Brackman Bros.	Browning	615	228-1410	TN	N
24288	ORDVA, Inc.	Hakford	615	898-1234	TN	Y
25443	B&K, Inc.	Smith	904	227-0093	FL	N
25501	Damal Supplies	Smythe	615	890-3529	TN	N
25595	Rubicon Systems	Orton	904	456-0092	FL	Y

```
SELECT * FROM PRODUCT JOIN VENDOR ON PRODUCT.V_CODE = VENDOR.V_CODE
```

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	PRODUCT.V_CODE	VENDOR.V_CODE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE	V_STATE	V_ORDER
23109-HB	Claw hammer	20-Jan-18	23	10	9.95	0.10	21225	21225	Bryson, Inc.	Smithson	615	223-3234	TN	Y
SM-18277	1.25-in. metal screw, 25	01-Mar-18	172	75	6.99	0.00	21225	21225	Bryson, Inc.	Smithson	615	223-3234	TN	Y
SW-23116	2.5-in. wd. screw, 50	24-Feb-18	237	100	8.45	0.00	21231	21231	D&E Supply	Singh	615	228-3245	TN	Y
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-17	32	15	14.99	0.05	21344	21344	Gomez Bros.	Ortega	615	889-2546	KY	N
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-17	18	12	17.49	0.00	21344	21344	Gomez Bros.	Ortega	615	889-2546	KY	N
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-17	43	20	4.99	0.00	21344	21344	Gomez Bros.	Ortega	615	889-2546	KY	N
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-18	15	8	39.95	0.00	23119	23119	Randsets Ltd.	Anderson	901	678-3998	GA	Y
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-18	23	5	43.99	0.00	23119	23119	Randsets Ltd.	Anderson	901	678-3998	GA	Y
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-17	8	5	109.92	0.05	24288	24288	ORDVA, Inc.	Hakford	615	898-1234	TN	Y
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-17	6	5	99.87	0.05	24288	24288	ORDVA, Inc.	Hakford	615	898-1234	TN	Y
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-18	11	5	256.99	0.05	24288	24288	ORDVA, Inc.	Hakford	615	898-1234	TN	Y
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-17	8	5	109.99	0.00	25595	25595	Rubicon Systems	Orton	904	456-0092	FL	Y
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-18	12	5	38.95	0.05	25595	25595	Rubicon Systems	Orton	904	456-0092	FL	Y
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-18	18	5	119.95	0.10	25595	25595	Rubicon Systems	Orton	904	456-0092	FL	Y

NATURAL JOIN

- Natural join returns all rows with matching values in the matching columns and **eliminates duplicate columns**
  - Determines the common attribute(s) by looking **for attributes with identical names and compatible data types**
  - Selects only the rows with common values in the common attribute(s)
  - If there are **no common attributes, returns the relational product of the two tables**,
- Unlike the Cartesian product, which concatenates each row of the first table with every row of the second, natural join considers only those pairs of rows with the same value on those attributes that appear in the schemas of both relations.
- Syntax:

```
SELECT column-list FROM table1 NATURAL JOIN table2
```

NATURAL JOIN (2)

FIGURE 3.10 TWO TABLES THAT WILL BE USED IN JOIN ILLUSTRATIONS

Table name: CUSTOMER

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE
1132445	Walker	32145	231
1217782	Adares	32145	125
1312243	Rakowski	34129	167
1321242	Rodriguez	37134	125
1542311	Smithson	37134	421
1657399	Vanloo	32145	231

Table name: AGENT

AGENT_CODE	AGENT_PHONE
125	6152439887
167	6153426778
231	6152431124
333	9041234445

```
SELECT * FROM Customer NATURAL JOIN Agent
```

FIGURE 3.13 NATURAL JOIN, STEP 3: PROJECT

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	6152439887
1321242	Rodriguez	37134	125	6152439887
1312243	Rakowski	34129	167	6153426778
1132445	Walker	32145	231	6152431124
1657399	Vanloo	32145	231	6152431124

JOIN USING

- Another way of joining tables using attribute with the same name
- Like NATURAL JOIN, eliminates duplicate columns
- Unlike NATURAL JOIN, if there are no common attributes, it gives an error
- Syntax:

SELECT *column-list* FROM *table1* JOIN *table 2* USING (*common-attribute)*

```
SELECT P_CODE, P_DESCRIPT, V_CODE, V_NAME, V_AREACODE,
V_PHONE FROM PRODUCT JOIN VENDOR USING (V_CODE);
```

THETA JOIN

- Like equi join but for any other comparison operator such as >, >=, <, <=

Car

CarModel	CarPrice
CarA	20,000
CarB	30,000
CarC	50,000

Boat

BoatModel	BoatPrice
Boat1	10,000
Boat2	40,000
Boat3	60,000

$$Car \bowtie_{CarPrice \geq BoatPrice} Boat$$

CarModel	CarPrice	BoatModel	BoatPrice
CarA	20,000	Boat1	10,000
CarB	30,000	Boat1	10,000
CarC	50,000	Boat1	10,000
CarC	50,000	Boat2	40,000

```
SELECT * FROM Car JOIN Boat ON CarPrice >= BoatPrice
```

FULL OUTER JOIN

- Returns not only the rows matching the join condition but it also returns all of the rows with unmatched values in the table on either side.

SELECT *column-list* FROM *table1* FULL [OUTER] JOIN *table2* ON *join-condition*

```
SELECT P_CODE, VENDOR.V_CODE, V_NAME
FROM VENDOR FULL JOIN PRODUCT ON
VENDOR. V_CODE = PRODUCT.V_CODE;
```

Not Supported by MYSQL!

P_CODE	V_CODE	V_NAME
	21226	SuperLoo, Inc.
	22567	Dome Supply
	24004	Brackman Bros.
	25443	B&K, Inc.
	25501	Damal Supplies
11QER/31	25595	Rubicon Systems
13-Q2/P2	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
1546-QQ2	23119	Randsets Ltd.
1558-QW1	23119	Randsets Ltd.
2232/QTY	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
2238/QPD	25595	Rubicon Systems
23109-HB	21225	Bryson, Inc.
23114-AA		
54778-2T	21344	Gomez Bros.
89-WRE-Q	24288	ORDVA, Inc.
PVC23DRT		
SM-18277	21225	Bryson, Inc.
SW-23116	21231	D&E Supply
WR3/TT3	25595	Rubicon Systems

LEFT OUTER JOIN

- Returns not only the rows matching the join condition but it also returns all of the rows in the left table with unmatched values in the right table.

SELECT *column-list* FROM *table1* LEFT [OUTER] JOIN *table2* ON *join-condition*

```
SELECT P_CODE, VENDOR.V_CODE, V_NAME
FROM VENDOR LEFT JOIN PRODUCT ON
VENDOR. V_CODE = PRODUCT.V_CODE;
```

P_CODE	V_CODE	V_NAME
23109-HB	21225	Bryson, Inc.
SM-18277	21225	Bryson, Inc.
	21226	SuperLoo, Inc.
SW-23116	21231	D&E Supply
13-Q2/P2	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
54778-2T	21344	Gomez Bros.
	22567	Dome Supply
1546-QQ2	23119	Randsets Ltd.
1558-QW1	23119	Randsets Ltd.
	24004	Brackman Bros.
2232/QTY	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
89-WRE-Q	24288	ORDVA, Inc.
	25443	B&K, Inc.
	25501	Damal Supplies
11QER/31	25595	Rubicon Systems
2238/QPD	25595	Rubicon Systems
WR3/TT3	25595	Rubicon Systems

RIGHT OUTER JOIN

- Returns not only the rows matching the join condition but it also returns all of the rows in the right table with unmatched values in the left table.

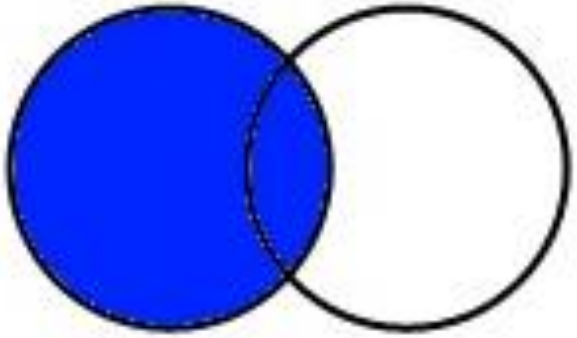
SELECT *column-list* FROM *table1* RIGHT [OUTER] JOIN *table2* ON *join-condition*

```
SELECT P_CODE, VENDOR.V_CODE, V_NAME
FROM VENDOR RIGHT JOIN PRODUCT ON
VENDOR. V_CODE = PRODUCT.V_CODE;
```

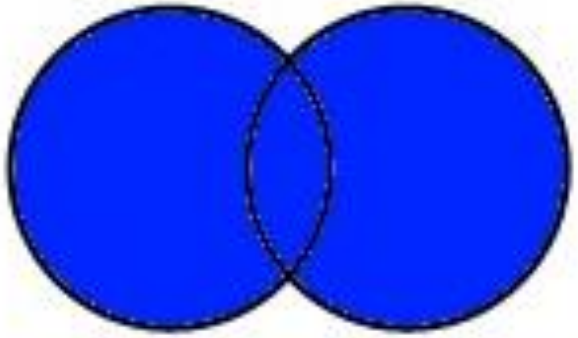
P_CODE	V_CODE	V_NAME
23114-AA		
PVC23DRT		
23109-HB	21225	Bryson, Inc.
SM-18277	21225	Bryson, Inc.
SW-23116	21231	D&E Supply
13-Q2/P2	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
54778-2T	21344	Gomez Bros.
1546-QQ2	23119	Randsets Ltd.
1558-QW1	23119	Randsets Ltd.
2232/QTY	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
89-WRE-Q	24288	ORDVA, Inc.
11QER/31	25595	Rubicon Systems
2238/QPD	25595	Rubicon Systems
WR3/TT3	25595	Rubicon Systems

Pictorial representation of JOINs

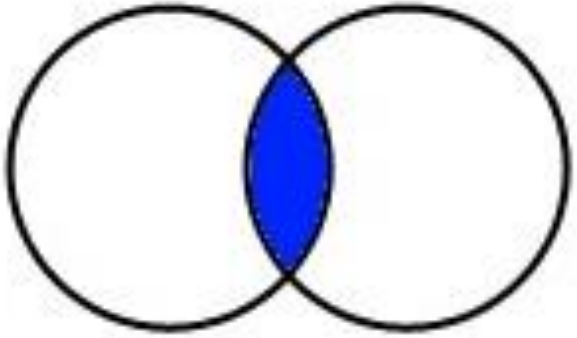
LEFT JOIN



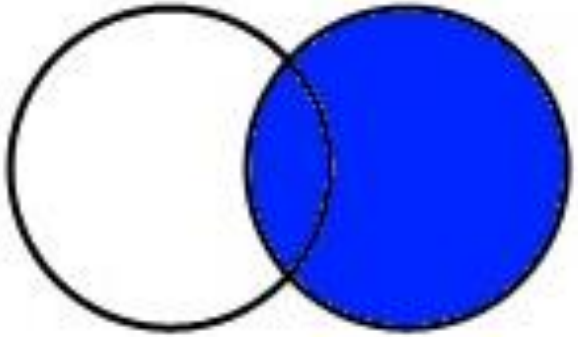
FULL OUTER JOIN



INNER JOIN



RIGHT JOIN



Join using WHERE clause

- Joining two tables

```
SELECT column-list FROM table1 JOIN table2 ON join-condition
```

```
SELECT column-list FROM table1 , table2 WHERE join-condition [AND other-conditions]
```

```
SELECT CUSTOMER.CUS_CODE, CUSTOMER.CUS_LNAME, AGENT.PHONE FROM CUSTOMER JOIN
AGENT ON CUSTOMER.AGENT_CODE = AGENT.AGENT_CODE
```

```
SELECT CUSTOMER.CUS_CODE, CUSTOMER.CUS_LNAME, AGENT.PHONE FROM
CUSTOMER, AGENT WHERE CUSTOMER.AGENT_CODE = AGENT.AGENT_CODE
```

- Joining more than two tables

```
SELECT column-list FROM table1 JOIN table2 ON join-condition JOIN table3 ON
join-condition
```

```
SELECT column-list FROM table1 , table2 , table3 WHERE join-conditions
```

Joining tables with Aliases

```
SELECT column-list FROM table1 t1 JOIN table2 t2 ON join-condition
```

```
SELECT column-list FROM table1 t1 , table2 t2 WHERE join-condition
```

```
SELECT C.CUS_CODE, C.CUS_LNAME, A.PHONE FROM CUSTOMER C
JOIN AGENT A ON C.AGENT_CODE = A.AGENT_CODE
```

```
SELECT C.CUS_CODE, C.CUS_LNAME, A.PHONE FROM CUSTOMER
C, AGENT A WHERE C.AGENT_CODE = A.AGENT_CODE
```

Recursive (self) JOIN

- Table alias is useful when a table must be joined to itself
- Unary relationships: Employees' managers are also employees

EMP_NUM	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIRE_DATE	EMP_AREACODE	EMP_PHONE	EMP_MGR
100	Mr.	Kolmycz	George	D	15-Jun-42	15-Mar-85	615	324-5456	
101	Ms.	Lewis	Rhonda	G	19-Mar-65	25-Apr-86	615	324-4472	100
102	Mr.	Vandam	Rhett		14-Nov-58	20-Dec-90	901	675-8993	100
103	Ms.	Jones	Anne	M	16-Oct-74	28-Aug-94	615	898-3456	100
104	Mr.	Lange	John	P	08-Nov-71	20-Oct-94	901	504-4430	105
105	Mr.	Williams	Robert	D	14-Mar-75	08-Nov-98	615	890-3220	
106	Mrs.	Smith	Jeanine	K	12-Feb-68	05-Jan-89	615	324-7883	105
107	Mr.	Diante	Jorge	D	21-Aug-74	02-Jul-94	615	890-4567	105
108	Mr.	Wiesenbach	Paul	R	14-Feb-66	18-Nov-92	615	897-4358	
109	Mr.	Smith	George	K	18-Jun-61	14-Apr-89	901	504-3339	108
110	Mrs.	Genkazi	Leighla	W	19-May-70	01-Dec-90	901	569-0093	108
111	Mr.	Washington	Rupert	E	03-Jan-66	21-Jun-93	615	890-4925	105
112	Mr.	Johnson	Edward	E	14-May-61	01-Dec-83	615	898-4387	100
113	Ms.	Smythe	Melanie	P	15-Sep-70	11-May-99	615	324-9006	105
114	Ms.	Brandon	Marie	G	02-Nov-56	15-Nov-79	901	882-0845	108
115	Mrs.	Saranda	Hermine	R	25-Jul-72	23-Apr-93	615	324-5505	105
116	Mr.	Smith	George	A	08-Nov-65	10-Dec-88	615	890-2984	108

Recursive (self) JOIN (2)

- Find employees' number and last name with their manager's number and last name

```
SELECT E.EMP_NUM, E.EMP_LNAME, E.EMP_MGR, M.EMP_LNAME FROM EMP E
JOIN EMP M ON E.EMP_MGR = M.EMP_NUM;
```

```
SELECT E.EMP_NUM, E.EMP_LNAME, E.EMP_MGR, M.EMP_LNAME FROM EMP E,
EMP M WHERE E.EMP_MGR = M.EMP_NUM;
```

EMP_NUM	E.EMP_LNAME	EMP_MGR	M.EMP_LNAME
112	Johnson	100	Kolmycz
103	Jones	100	Kolmycz
102	Vandam	100	Kolmycz
101	Lewis	100	Kolmycz
115	Saranda	105	Williams
113	Smythe	105	Williams
111	Washington	105	Williams
107	Diante	105	Williams
106	Smith	105	Williams
104	Lange	105	Williams
116	Smith	108	Wiesenbach
114	Brandon	108	Wiesenbach
110	Genkazi	108	Wiesenbach
109	Smith	108	Wiesenbach

SQL Functions

- SQL functions are very useful tools, similar to functions in programming languages
- Some categories of SQL functions
  - Date and Time Functions
  - Numerical Functions
  - String Functions
  - Conversion Functions
- Functions always use numerical, date, or string values; a value may be part of a command or a table attribute

SQL Functions

- Date and time functions
  - All date functions take one parameter of a date or character data type and return a value (character, numeric or date type);
  - Different implementation in different DBMS
- Numeric functions
  - Can be grouped in different ways, such as algebraic, trigonometric, and logarithmic
- String functions
  - String manipulations - among the most-used functions in programming
- Conversion functions
  - Allow you to take a value of a given data type and convert it to the equivalent value in another data type

Date

DEAKIN UNIVERSITY

TABLE 7.10

SELECTED MYSQL DATE/TIME FUNCTIONS

FUNCTION	EXAMPLE(S)
Date_Format Returns a character string or a formatted string from a date value Syntax: DATE_FORMAT(date_value, fmt) fmt = format used; can be: %M: name of month %m: two-digit month number %b: abbreviated month name %d: number of day of month %W: weekday name %a: abbreviated weekday name %Y: four-digit year %y: two-digit year	Displays the product code and date the product was last received into stock for all products: SELECT P_CODE, DATE_FORMAT(P_INDATE, '%m/%d/%y') FROM PRODUCT; SELECT P_CODE, DATE_FORMAT(P_INDATE, '%M %d, %Y') FROM PRODUCT;
YEAR Returns a four-digit year Syntax: YEAR(date_value)	Lists all employees born in 1982: SELECT EMP_LNAME, EMP_FNAME, EMP_DOB, YEAR(EMP_DOB) AS YEAR FROM EMPLOYEE WHERE YEAR(EMP_DOB) = 1982;

Date (2)

SYSDATE()

Function to get system date and time

```
SELECT SYSDATE();
```

Get system time

```
SELECT TIME(SYSDATE());
```

Get system date

```
SELECT DATE(SYSDATE());
```

TABLE 7.10 (CONTINUED)

SELECTED MYSQL DATE/TIME FUNCTIONS

MONTH Returns a two-digit month code Syntax: MONTH(date_value)	Lists all employees born in November: SELECT EMP_LNAME, EMP_FNAME, EMP_DOB, MONTH(EMP_DOB) AS MONTH FROM EMPLOYEE WHERE MONTH(EMP_DOB) = 11;
DAY Returns the number of the day Syntax: DAY(date_value)	Lists all employees born on the 14th day of the month: SELECT EMP_LNAME, EMP_FNAME, EMP_DOB, DAY(EMP_DOB) AS DAY FROM EMPLOYEE WHERE DAY(EMP_DOB) = 14;
ADDDATE Adds a number of days to a date Syntax: ADDDATE(date_value, n) n = number of days **DATE_ADD** Adds a number of days, months, or years to a date. This is similar to ADDDATE except it is more robust. It allows the user to specify the date unit to add. Syntax: DATE_ADD(date, INTERVAL n unit) n = number to add unit = date unit, can be: DAY: add n days WEEK: add n weeks MONTH: add n months YEAR: add n years	List all products with the date they will have been on the shelf for 30 days. SELECT P_CODE, P_INDATE, ADDDATE(P_INDATE, 30) FROM PRODUCT ORDER BY ADDDATE(P_INDATE, 30); Lists all products with their expiration date (two years from the purchase date): SELECT P_CODE, P_INDATE, DATE_ADD(P_INDATE, INTERVAL 2 YEAR) FROM PRODUCT ORDER BY DATE_ADD(P_INDATE, INTERVAL 2 YEAR);
LAST_DAY Returns the date of the last day of the month given in a date Syntax: LAST_DAY(date_value)	Lists all employees who were hired within the last seven days of a month: SELECT EMP_LNAME, EMP_FNAME, EMP_HIRE_DATE FROM EMPLOYEE WHERE EMP_HIRE_DATE >= DATE_ADD(LAST_DAY (EMP_HIRE_DATE), INTERVAL -7 DAY);

Numeric Functions

TABLE 7.11

SELECTED NUMERIC FUNCTIONS

FUNCTION	EXAMPLE(S)
ABS Returns the absolute value of a number Syntax: ABS(numeric_value)	In Oracle, use the following: SELECT 1.95, –1.93, ABS(1.95), ABS(–1.93) FROM DUAL; In MS Access, MySQL, and MS SQL Server, use the following: SELECT 1.95, –1.93, ABS(1.95), ABS(–1.93);
ROUND Rounds a value to a specified precision (number of digits) Syntax: ROUND(numeric_value, p) p = precision	Lists the product prices rounded to one and zero decimal places: SELECT P_CODE, P_PRICE, ROUND(P_PRICE,1) AS PRICE1, ROUND(P_PRICE,0) AS PRICE0 FROM PRODUCT;
CEIL/CEILING/FLOOR Returns the smallest integer greater than or equal to a number or returns the largest integer equal to or less than a number, respectively Syntax: CEIL(numeric_value) Oracle or MySQL CEILING(numeric_value) MS SQL Server or MySQL FLOOR(numeric_value)	Lists the product price, the smallest integer greater than or equal to the product price, and the largest integer equal to or less than the product price. In Oracle or MySQL, use the following: SELECT P_PRICE, CEIL(P_PRICE), FLOOR(P_PRICE) FROM PRODUCT; In MS SQL Server or MySQL, use the following: SELECT P_PRICE, CEILING(P_PRICE), FLOOR(P_PRICE) FROM PRODUCT; MS Access does not support these functions. Note that MySQL supports both CEIL and CEILING.

String Functions

FUNCTION	EXAMPLE(S)
Concatenation **|| Oracle** **+ Access and MS SQL Server** **& Access** **CONCAT() MySQL** Concatenates data from two different character columns and returns a single column. Syntax: strg_value || strg_value strg_value + strg_value strg_value & strg_value CONCAT(strg_value, strg_value) The CONCAT function can only accept two string values so nested CONCAT functions are required when more than two values are to be concatenated.	Lists all employee names (concatenated). In Oracle, use the following: SELECT EMP_LNAME || ', ' || EMP_FNAME AS NAME FROM EMPLOYEE; In Access and MS SQL Server, use the following: SELECT EMP_LNAME + ', ' + EMP_FNAME AS NAME FROM EMPLOYEE; In MySQL, use the following: SELECT CONCAT(CONCAT(EMP_LNAME, ', '), EMP_FNAME AS NAME FROM EMPLOYEE;
UPPER Oracle, MS SQL Server, and MySQL **UCASE MySQL and Access** **LOWER Oracle, MS SQL Server, and MySQL** **LCASE MySQL and Access** Returns a string in all capital or all lowercase letters Syntax: UPPER(strg_value) UCASE(strg_value) LOWER(strg_value) LCASE(strg_value)	Lists all employee names in all capital letters (concatenated). In Oracle, use the following: SELECT UPPER(EMP_LNAME || ', ' || EMP_FNAME) AS NAME FROM EMPLOYEE; In MS SQL Server, use the following: SELECT UPPER(EMP_LNAME + ', ' + EMP_FNAME) AS NAME FROM EMPLOYEE; In Access, use the following: SELECT UCASE(EMP_LNAME & ', ' & EMP_FNAME) AS NAME FROM EMPLOYEE; In MySQL, use the following: SELECT UPPER(CONCAT(CONCAT(EMP_LNAME, ', '), EMP_FNAME AS NAME FROM EMPLOYEE;
SUBSTRING Returns a substring or part of a given string parameter Syntax: SUBSTR(strg_value, p, l) Oracle and MySQL SUBSTRING(strg_value,p,l) MS SQL Server and MySQL MID(strg_value,p,l) Access p = start position l = length of characters If the length of characters is omitted, the functions will return the remainder of the string value.	Lists the first three characters of all employee phone numbers. In Oracle or MySQL, use the following: SELECT EMP_PHONE, SUBSTR(EMP_PHONE,1,3) AS PREFIX FROM EMPLOYEE; In MS SQL Server or MySQL, use the following: SELECT EMP_PHONE, SUBSTRING(EMP_PHONE,1,3) AS PREFIX FROM EMPLOYEE; In Access, use the following: SELECT EMP_PHONE, MID(EMP_PHONE, 1,3) AS PREFIX FROM EMPLOYEE;

Conversion Functions

FUNCTION	EXAMPLE(S)
Numeric or Date to Character: **TO_CHAR Oracle** **CAST Oracle, MS SQL Server, MySQL** **CONVERT MS SQL Server, MySQL** **CSTR Access** Returns a character string from a numeric or date value. Syntax: TO_CHAR(value-to-convert, fmt) fmt = format used; can be: 9 = displays a digit 0 = displays a leading zero , = displays the comma . = displays the decimal point $= displays the dollar sign B = leading blank S = leading sign MI = trailing minus sign CAST (value-to-convert AS char(length)) Note that Oracle and MS SQL Server can use CAST to convert the numeric data into fixed length or variable length character data type. MySQL cannot CAST into variable length character data, only fixed length. MS SQL Server: CONVERT(varchar(length), value-to-convert) MySQL: CONVERT(value-to-convert, char(length)) The primary difference between CAST and CONVERT is that CONVERT can also be used to change the character set of the data. CSTR(value-to-convert)	Lists all product prices, product received date, and percent discount using formatted values. TO_CHAR: SELECT P_CODE, TO_CHAR(P_PRICE,'999.99') AS PRICE, TO_CHAR(P_INDATE, 'MM/DD/YYYY') AS INDATE, TO_CHAR(P_DISCOUNT,'0.99') AS DISC FROM PRODUCT; CAST in Oracle and MS SQL Server: SELECT P_CODE, CAST(P_PRICE AS VARCHAR(8)) AS PRICE, CAST(P_INDATE AS VARCHAR(20)) AS INDATE, CAST(P_DISCOUNT AS VARCHAR(4)) AS DISC FROM PRODUCT; CAST in MySQL: SELECT P_CODE, CAST(P_PRICE AS CHAR(8)) AS PRICE, CAST(P_INDATE AS CHAR(20)) AS INDATE, CAST(P_DISCOUNT AS CHAR(4)) AS DISC FROM PRODUCT; CONVERT in MS SQL Server: SELECT P_CODE, CONVERT(VARCHAR(8), P_PRICE) AS PRICE, CONVERT(VARCHAR(20), P_INDATE) AS INDATE, CONVERT(VARCHAR(4), P_DISC) AS DISC FROM PRODUCT; CONVERT in MySQL: SELECT P_CODE, CONVERT(P_PRICE, CHAR(8)) AS PRICE, CONVERT(P_INDATE, CHAR(20)) AS INDATE, CONVERT(P_DISC, CHAR(4)) AS DISC FROM PRODUCT; CSTR in Access: SELECT P_CODE, CSTR(P_PRICE) AS PRICE, CSTR(P_INDATE) AS INDATE, CSTR(P_DISC) AS DISCOUNT FROM PRODUCT;

Conversion Functions (2)

FUNCTION	EXAMPLE(S)
String to Number: **TO_NUMBER Oracle** **CAST Oracle, MS SQL Server, MySQL** **CONVERT MS SQL Server, MySQL** **CINT Access** **CDEC Access** Returns a number from a character string Syntax: Oracle: TO_NUMBER(char_value, fmt) fmt = format used; can be: 9 = indicates a digit B = leading blank S = leading sign MI = trailing minus sign CAST (value-to-convert as numeric-data type) Note that in addition to the INTEGER and DECIMAL(I,d) data types, Oracle supports NUMBER and MS SQL Server supports NUMERIC. MS SQL Server: CONVERT(value-to-convert, decimal(I,d)) MySQL: CONVERT(value-to-convert, decimal(I,d)) Other than the data type to be converted into, these functions operate the same as described above. CINT in Access returns the number in the integer data type, while CDEC returns decimal data type.	Converts text strings to numeric values when importing data to a table from another source in text format; for example, the query shown here uses the TO_NUMBER function to convert text formatted to Oracle default numeric values using the format masks given. TO_NUMBER: SELECT TO_NUMBER('–123.99', 'S999.99'), TO_NUMBER('99.78–','B999.99MI') FROM DUAL; CAST: SELECT CAST('–123.99' AS DECIMAL(8,2)), CAST('–99.78' AS DECIMAL(8,2)); The CAST function does not support the trailing sign on the character string. CINT and CDEC: SELECT CINT('–123'), CDEC('–123.99');

An Example of SQL Functions

- Suppose we have an Invoice table as follows:

  INVOICE (**INV_NUMBER**, INV_DATE, CUST_CODE)

- Listing invoice numbers and invoice dates is simply:

  `SELECT` INV_NUMBER, INV_DATE `FROM` INVOICE;

INV_NUMBER	INV_DATE
1001	16-Jan-18
1002	16-Jan-18
1003	16-Jan-18
1004	17-Jan-18
1005	17-Jan-18
1006	17-Jan-18
1007	17-Jan-18
1008	17-Jan-18

BUT, how old is each invoice?

An Example of SQL Functions

- List invoice numbers, dates and ages (in days) of all invoices.

```
SELECT INV_NUMBER, INV_DATE,
DATEDIFF(SYSDATE(), INV_DATE) AS "Age in Days"
FROM INVOICE;
```

INV_NUMBER	INV_DATE	Age in Days
1001	16-Jan-18	197
1002	16-Jan-18	197
1003	16-Jan-18	197
1004	17-Jan-18	196
1005	17-Jan-18	196
1006	17-Jan-18	196
1007	17-Jan-18	196
1008	17-Jan-18	196

Summary

- **Relational Algebra**
  - UNION
  - INTERSECT
  - DIFFERENCE/MINUS
  - PRODUCT
  - SELECT
  - PROJECT
- **Joining multiple tables**
  - CROSS JOIN
  - INNER JOIN
  - NATURAL JOIN
  - FULL OUTER JOIN
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
- **SQL Functions**
  - Date/time functions
  - Conversion functions
  - Numeric functions
  - String functions

This Week's OnTrack Task

- 6.1P SELECT with JOIN

  - `SELECT` queries from multiple tables with `JOIN`

Next Week

- DDL – Creating and altering tables
- More DML - Inserting and Update records

Thank you

See you next week

Any questions/comments?

SQL Demo

Let's see some examples

Readings and References:

- Chapters 3 and 7

Database Systems : Design, Implementation, & Management 13TH EDITION, by Carlos Coronel, Steven Morris