COMP810 Data Warehousing and Big Data

Semester 2 2024

Dr Victor Miranda

COMP810

Week 5 Data Warehousing

- Complex OLAP operations
  & SQL queries

JOINS IN SQL

- An SQL ‘join’ is used fetch data from two or more tables, which is set to appear as a single set of data.
- Right outer join:
  - In practice, a ‘join’ combines columns from two or more table by using common identifiers (IDs) to both tables
- Keyword: JOIN … ON

Types of 'JOIN'

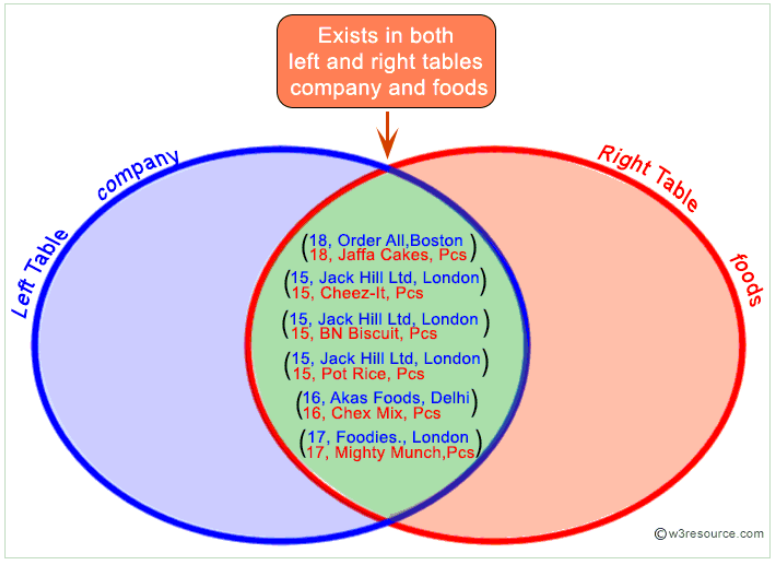
- Inner join:
  - Include all the rows from both tables
- Left outer join:
  - Include the left TABLE even if there's no match.
- Right outer join:
  - Include the right TABLE even if there's no match
- Full outer join:
  - Include the both left and right TABLES even if there's no match
- The CROSS join

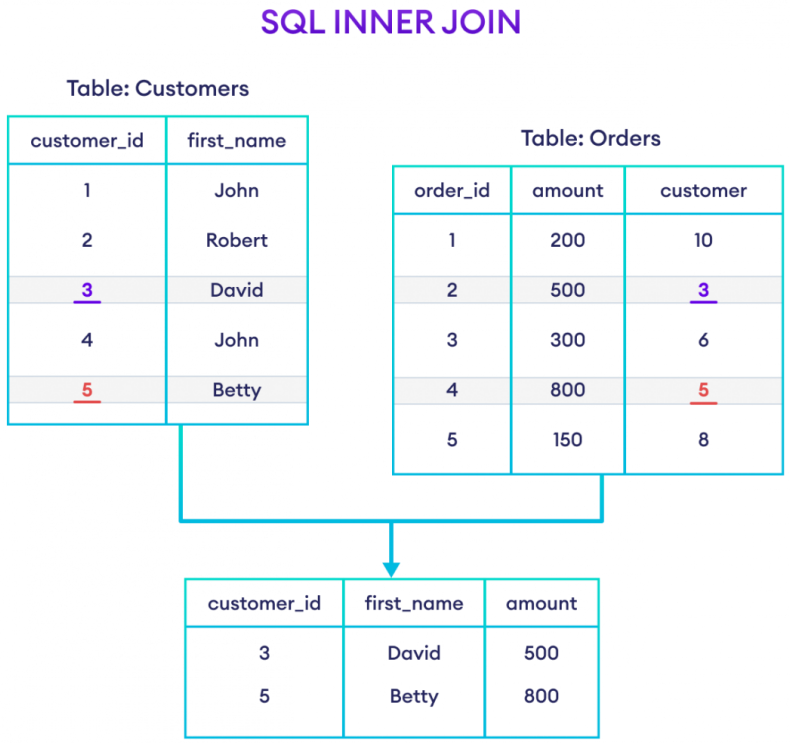
The INNER join

- The INNER join selects all COMMON rows from both tables even if there's no match.

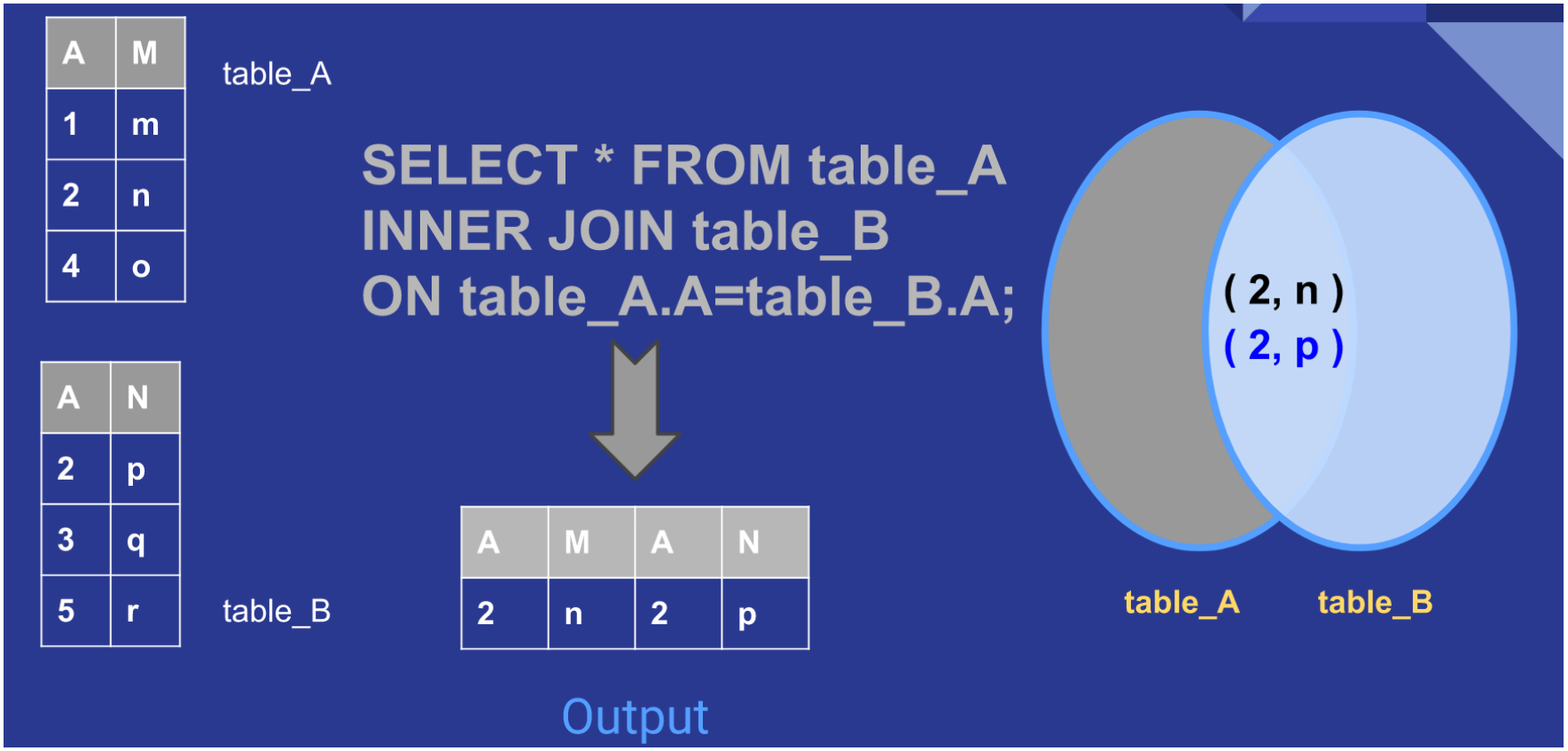
The INNER join

- The INNER join selects all COMMON rows from both tables even if there's no match.





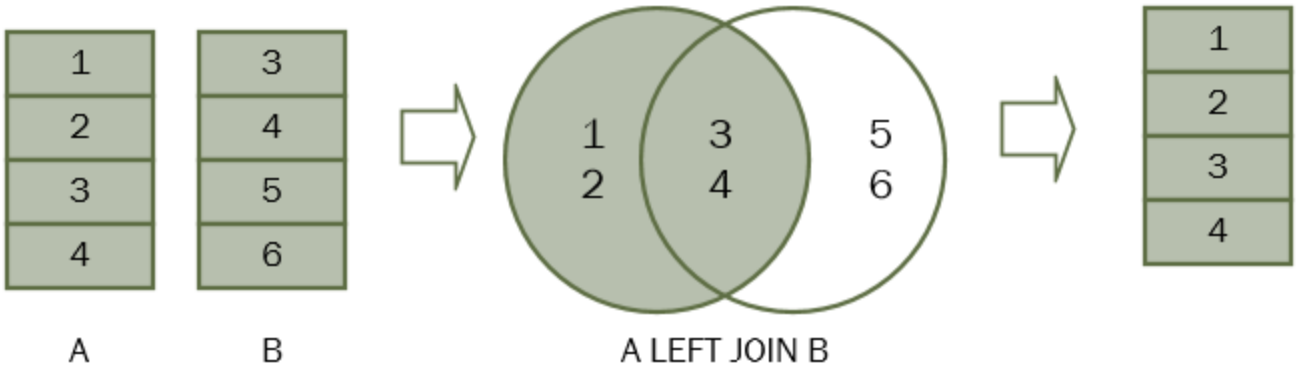
The INNER join - EXAMPLE



The LEFT join or LEFT OUTER join

- The LEFT JOIN joins two tables and fetches rows based on 'conditions' (matching/linking both tables). The is an 'order': Left table and right table.

- The unmatched tables from the RIGHT table will also be available as NULLs if there is no match.



The LEFT join - EXAMPLE

A	M
1	m
2	n
4	o

table_A

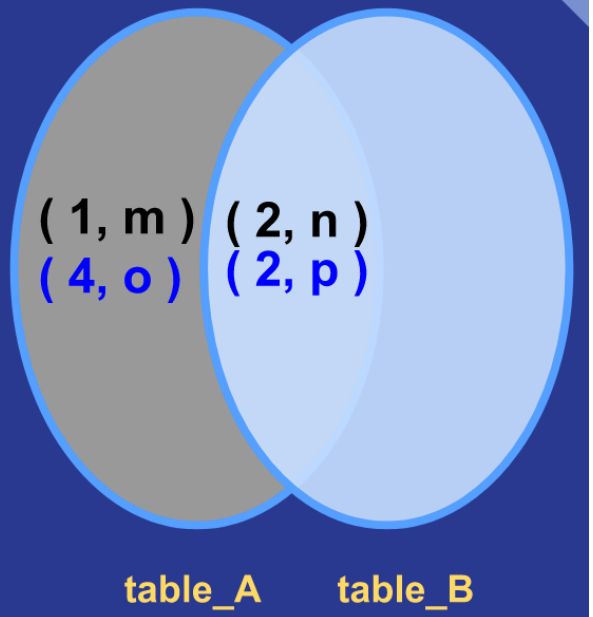
A	N
2	p
3	q
5	r

table_B

```
SELECT * FROM table_A
LEFT JOIN table_B
ON table_A.A=table_B.A;
```

A	M	A	N
2	n	2	p
1	m	null	null
4	o	null	null

Output



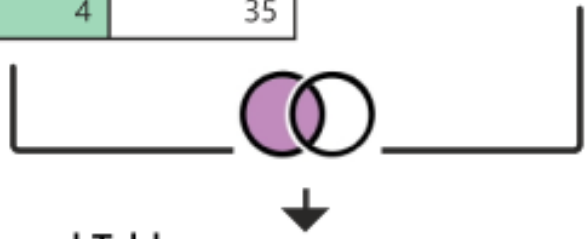
The LEFT join – EXAMPLE II

Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	4	35

Right Table

ID	Country
1	USA
2	Canada
3	Panama



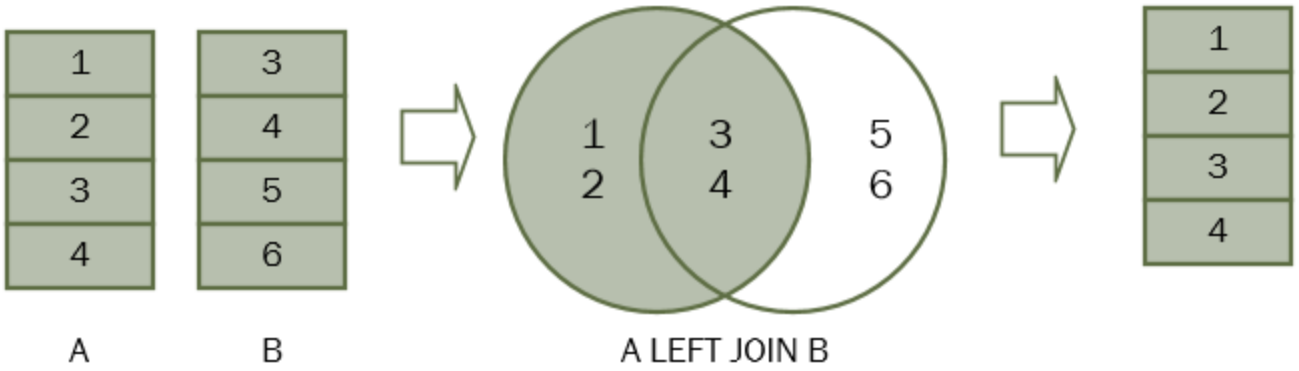
Merged Table

Date	CountryID	Units	Country
1/1/2020	1	40	USA
1/2/2020	1	25	USA
1/3/2020	3	30	Panama
1/4/2020	4	35	*null*

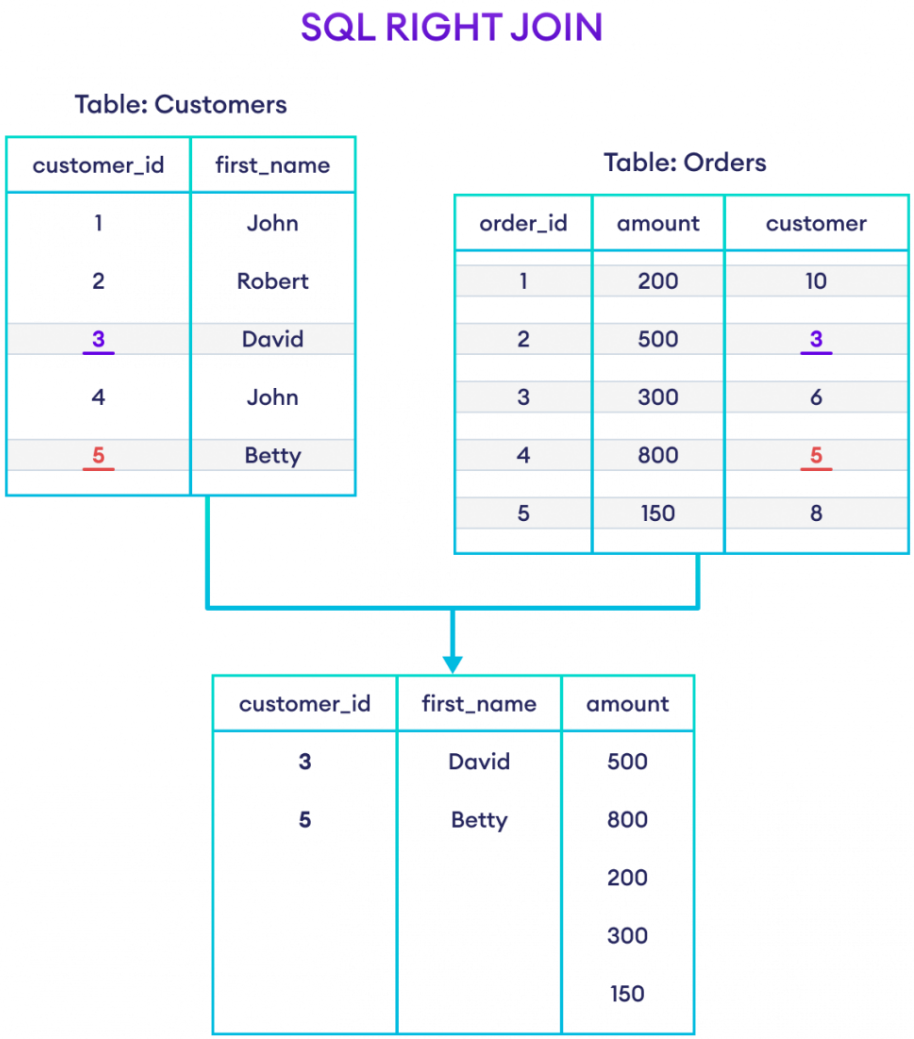
The RIGHT join

- The RIGHT JOIN joins two tables and fetches rows based on 'conditions' (matching/linking both tables). The is an 'order': Left table and right table.

- The unmatched tables from the LEFT table will also be available as NULLs if there is no match.



The RIGHT join – EXAMPLE



The RIGHT join – EXAMPLE II

A	M
1	m
2	n
4	o

table_A

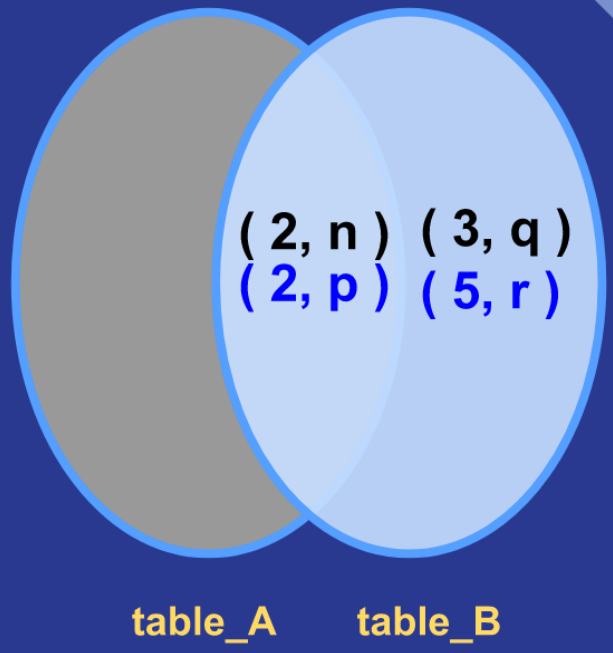
A	N
2	p
3	q
5	r

table_B

```
SELECT * FROM table_A
RIGHT JOIN table_B
ON table_A.A=table_B.A;
```

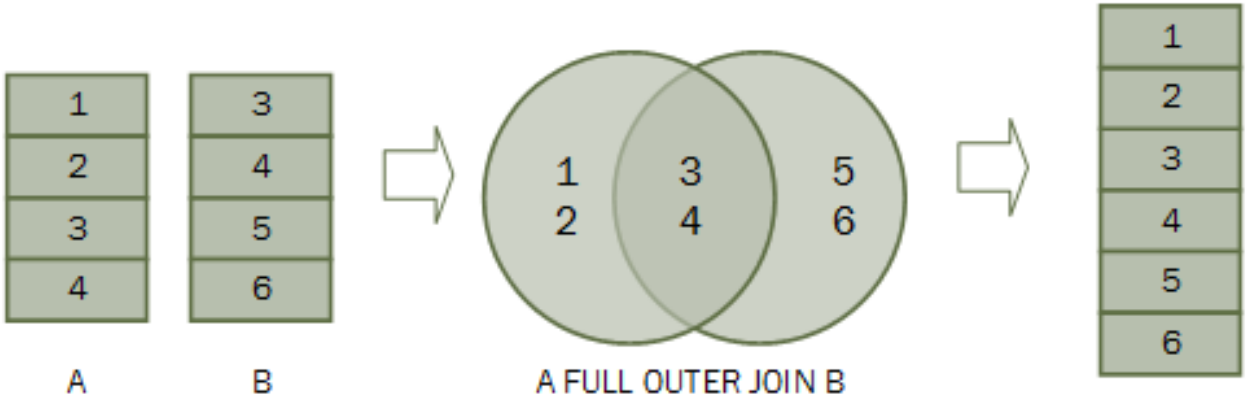
A	M	A	N
2	n	2	p
null	null	3	q
null	null	5	r

Output



The FULL join

- The FULL JOIN returns all records when there is a march in left (Table 1) **or** right (Table 2) records.



The FULL join

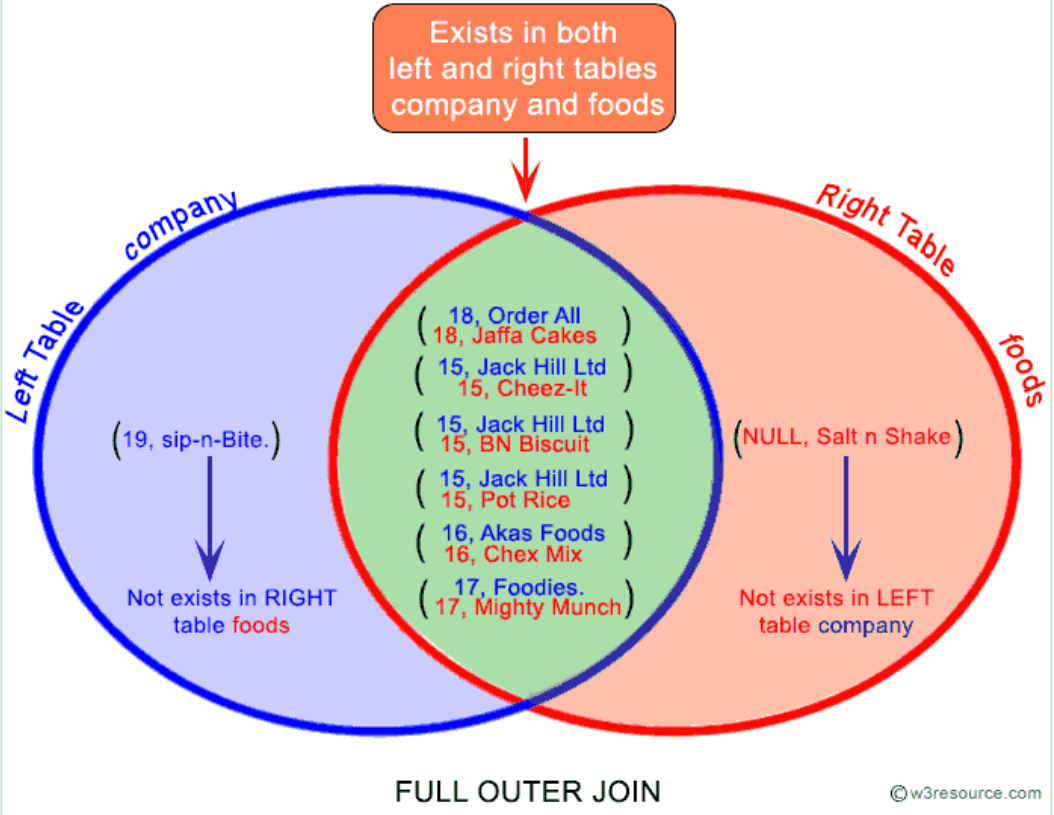
- The FULL JOIN returns all records when there is a march in left (Table 1) **or** right (Table 2) records.

```
SELECT a.company_id as "a.ComID",
a.company_name as "C_Name",
b.company_id as "b.ComID"
FROM   company a
LEFT OUTER JOIN foods b
ON a.company_id = b.company_id;
```

A.ComID	C_Name	B.ComID
16	Akas Foods	16
15	Jack Hill Ltd	15
15	Jack Hill Ltd	15
17	Foodies.	17
15	Jack Hill Ltd	15
18	Order All	18
19	sip-n-Bite.	-

```
SELECT a.company_id as "a.ComID",
a.company_name as "C_name",
b.company_id as "b.ComID"
FROM   company a
RIGHT OUTER JOIN foods b
ON a.company_id = b.company_id;
```

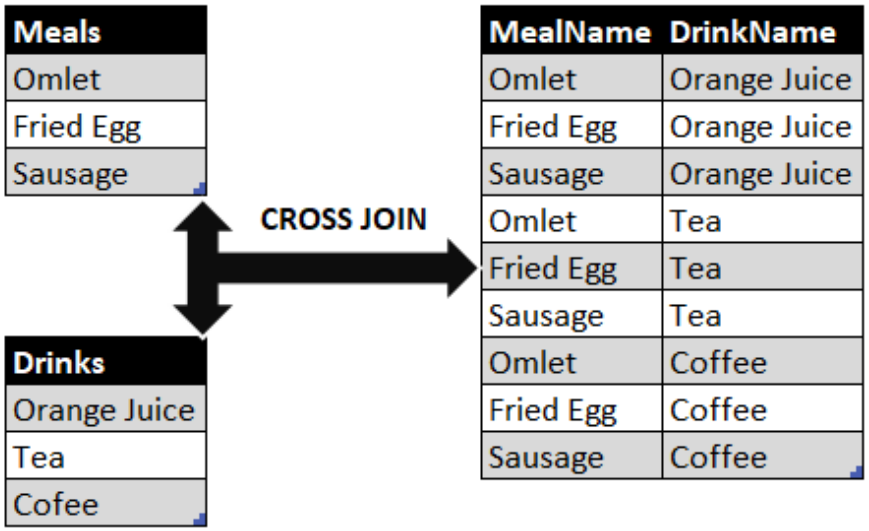
A.ComID	C_name	B.ComID
16	Akas Foods	16
15	Jack Hill Ltd	15
15	Jack Hill Ltd	15
17	Foodies.	17
15	Jack Hill Ltd	15
18	Order All	18
-	-	-



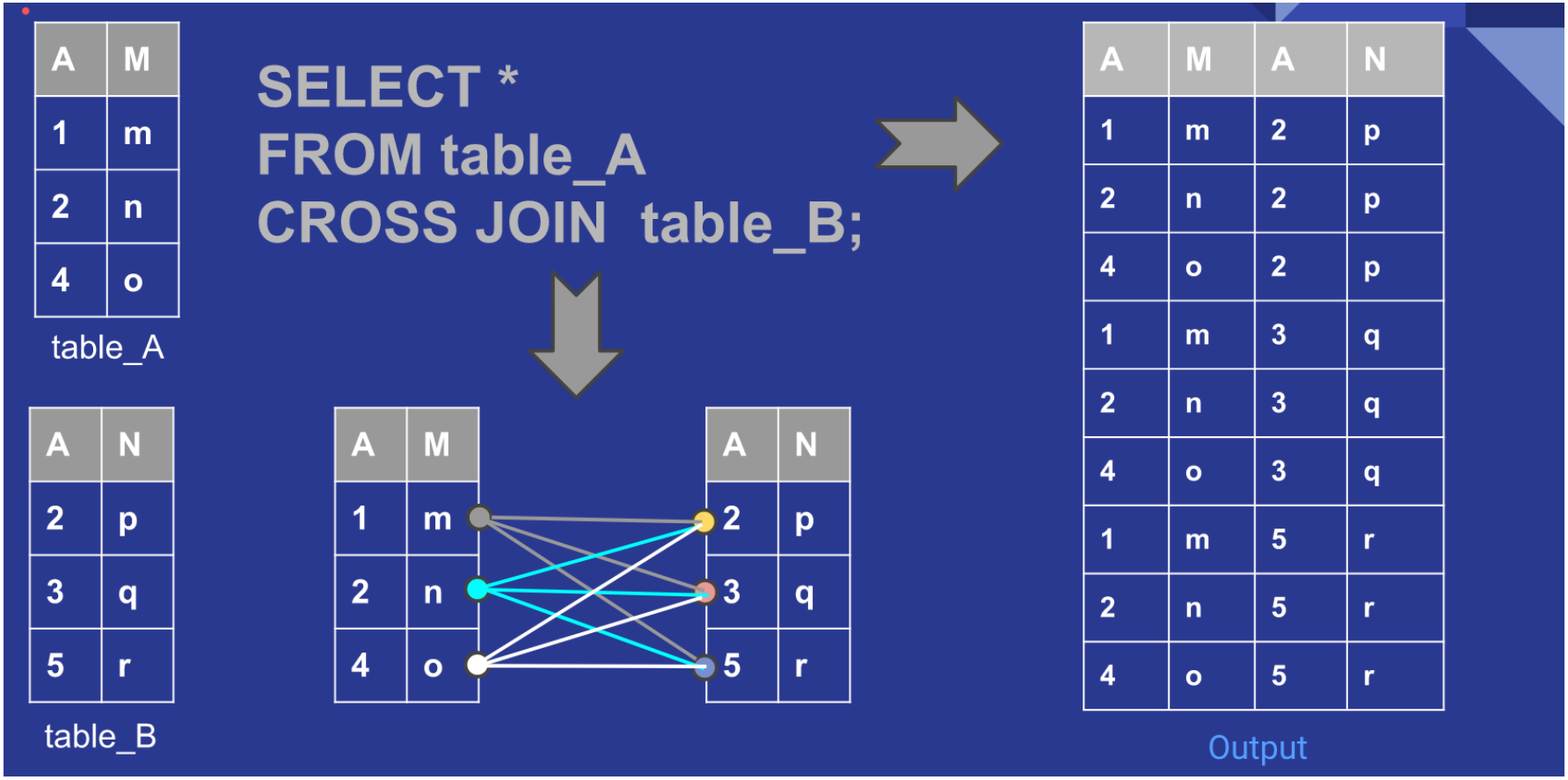
The CROSS join

- The CROSS JOIN produces crossed-set, where all entries from the first table (Table 1) are 'crossed' with all entries from the second table (also called Cartesian product).

- If there is a WHERE clause, this call resembles an INNER JOIN



The CROSS join - EXAMPLE



More examples

SQL– Left/Right Join

Example:

Schema:

People(person_fname, person_lname, person_id, person_state, person_city)

Movies(movie_id, movie_title, director_id, studio_id)

Location(movie_id, city, state)

Query:

```
Select movie_title, city, state
From Movies Left Join Locations
        On Movies.movie_id = Locations.movie_id

Select movie_title, person_fname, person_lname
From Movies Right Join People
        On Movies.director_id = Person.person_id
```

SQL– Left/Right Join

Example:

Schema:

People(person_fname, person_lname, person_id, person_state, person_city)

Movies(movie_id, movie_title, director_id, studio_id)

Location(movie_id, city, state)

Query:

```
Select movie_title, city, state
From Movies Left Join Locations
        On Movies.movie_id = Locations.movie_id
```

Includes all matched movie titles

```
Select movie_title, person_fname, person_lname
From Movies Right Join People
        On Movies.director_id = Person.person_id
```

Includes all people matching to directors

SQL Inner Joins

```
SELECT S.name, E.classid
  FROM Students S (INNER) JOIN Enrolled E
  ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150

Note the previous version of this query (with no join keyword) is an “Implicit join”

SQL Inner Joins

```
SELECT S.name, E.classid
  FROM Students S (INNER) JOIN Enrolled E
  ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150

Unmatched keys

What kind of Join is this?

```
SELECT S.name, E.classid
  FROM Students S ?? Enrolled E
  ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
Brown	NULL

SQL Joins

```
SELECT S.name, E.classid
  FROM Students S LEFT OUTER JOIN Enrolled E
  ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
Brown	NULL

What kind of Join is this?

```
SELECT S.name, E.classid
  FROM Students S ?? Enrolled E
  ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
NULL	English10

SQL Joins

```
SELECT S.name, E.classid
  FROM Students S RIGHT OUTER JOIN Enrolled E
  ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
NULL	English10

SQL Joins

```
SELECT S.name, E.classid
  FROM Students S ? JOIN Enrolled E
  ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
NULL	English10
Brown	NULL

SQL Joins

```
SELECT S.name, E.classid
  FROM Students S FULL OUTER JOIN Enrolled E
  ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
NULL	English10
Brown	NULL

What kind of Join is this?

```
SELECT S.name, E.classid
  FROM Students S ?? Enrolled E
  ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Smith	French150

SQL Joins

```
SELECT S.name, E.classid
  FROM Students S LEFT SEMI JOIN Enrolled E
  ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Smith	French150

What kind of Join is this?

```
SELECT *
  FROM Students S ?? Enrolled E
```

S

S.name	S.sid
Jones	11111
Smith	22222

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150

S.name	S.sid	E.sid	E.classid
Jones	11111	11111	History105
Jones	11111	11111	DataScience194
Jones	11111	22222	French150
Smith	22222	11111	History105
Smith	22222	11111	DataScience194
Smith	22222	22222	French150

SQL Joins

```
SELECT *
  FROM Students S CROSS JOIN Enrolled E
```

S

S.name	S.sid
Jones	11111
Smith	22222

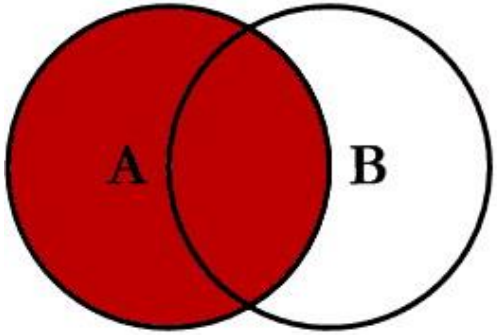
E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150

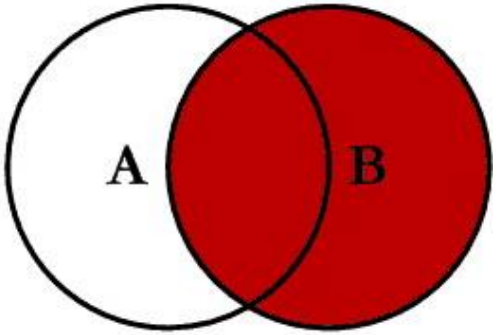
S.name	S.sid	E.sid	E.classid
Jones	11111	11111	History105
Jones	11111	11111	DataScience194
Jones	11111	22222	French150
Smith	22222	11111	History105
Smith	22222	11111	DataScience194
Smith	22222	22222	French150

Summary

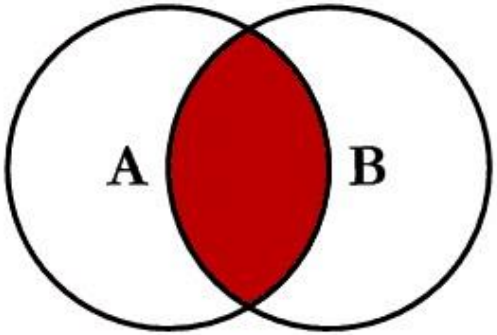
SQL JOINS



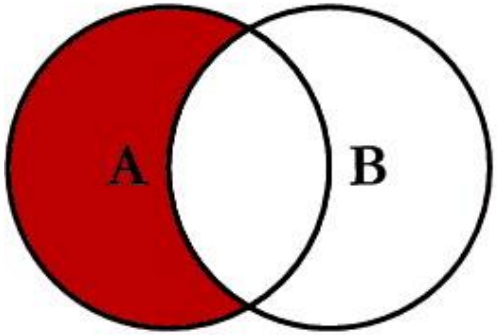

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



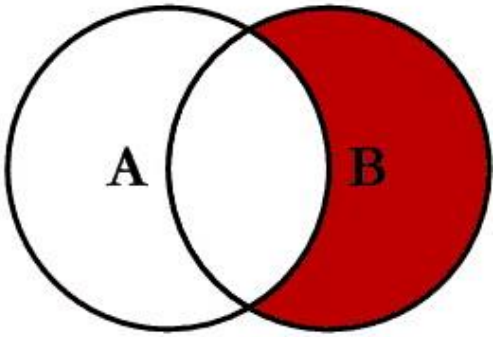

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



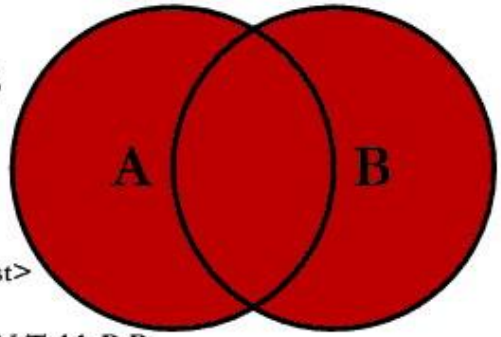

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



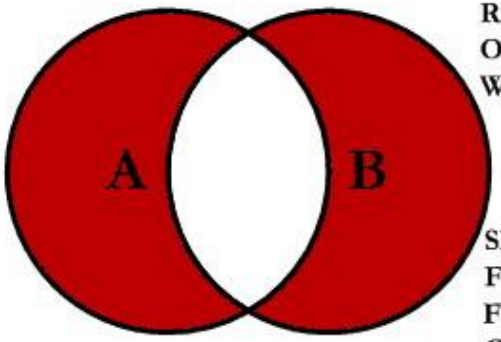

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
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
```

The UNION operator

The UNION operator

It is used to 'combine' the resulting set of two or more SELECT operations

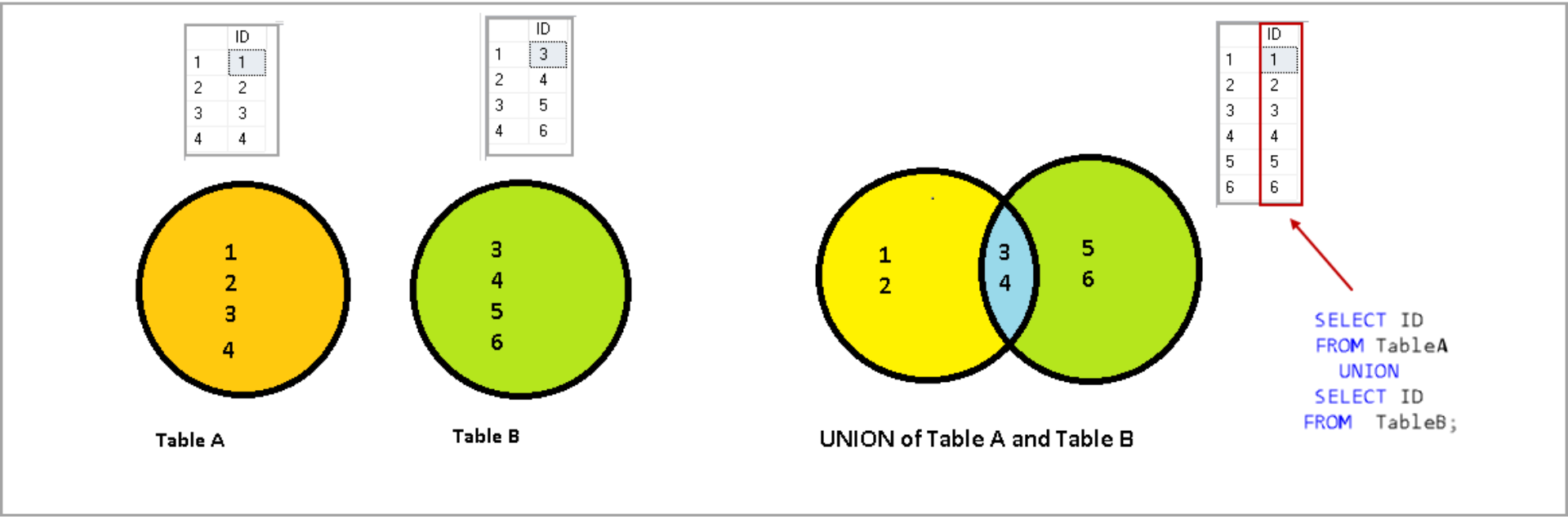
NOTES:

(a) Every SELECT output must have the same number of columns

(b) The columns should have similar data types

(c) The columns from each SELECT operation must be sorted equally.

The UNION operator - EXAMPLE



SQL Views

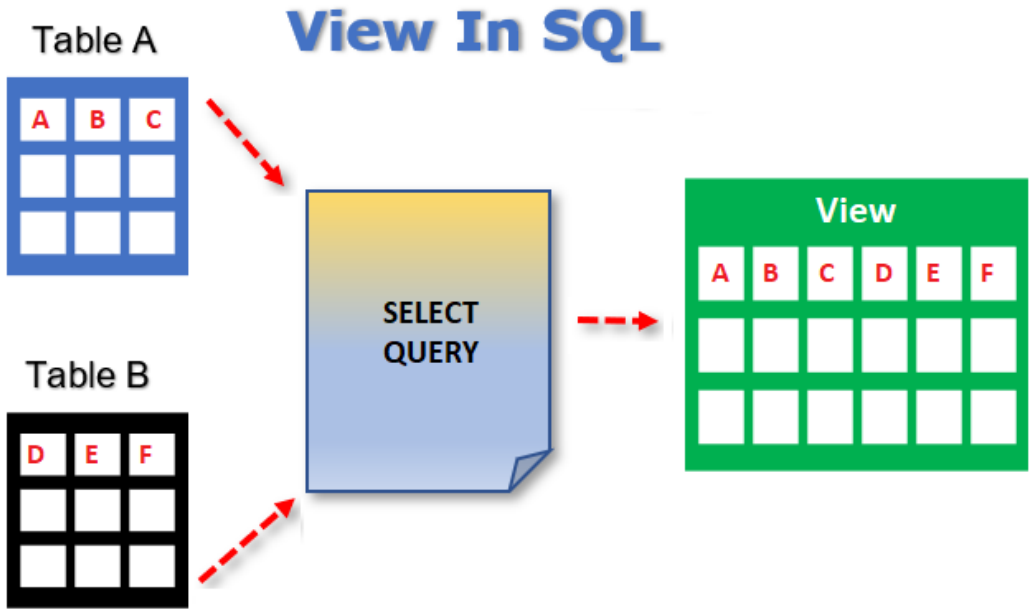
Types of Views

- Virtual views:
  - Used in databases
  - Computed only on-demand – slow*er* at runtime
  - Always up to date

- Materialized views
  - Used in data warehouses
  - Precomputed offline – fast*er* at runtime
  - May have stale data

SQL Views

- In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table.



Defining Views - Syntax

Views are relations, except that they are not physically stored.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

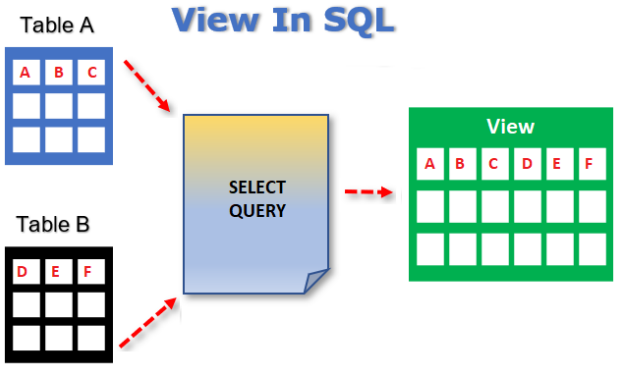
Defining Views - Syntax

Views are relations, except that they are not physically stored.

For presenting different information to different users

Employee(ssn, name, department, project, salary)

```
CREATE VIEW  Developers AS
  SELECT name, project
  FROM  Employee
  WHERE department = "Development"
```



Payroll has access to Employee, others only to Developers

A Different View

Person(name, city)
Purchase(buyer, seller, product, store)
Product(name, maker, category)

```
CREATE VIEW  Seattle-view  AS

        SELECT  buyer, seller, product, store
        FROM    Person, Purchase
        WHERE   Person.city = "Seattle"    AND
                Person.name = Purchase.buyer
```

We have a new virtual table:
Seattle-view(buyer, seller, product, store)

A Different View

We can later use the view:

```
SELECT   name, store
FROM     Seattle-view, Product
WHERE    Seattle-view.product = Product.name  AND
         Product.category = “shoes”
```

What Happens When We Query a View ?

```
SELECT   name, Seattle-view.store
FROM     Seattle-view, Product
WHERE    Seattle-view.product = Product.name  AND
         Product.category = "shoes"
```

```
SELECT  name, Purchase.store
FROM    Person, Purchase, Product
WHERE   Person.city = "Seattle"    AND
        Person.name = Purchase.buyer   AND
        Purchase.poduct = Product.name  AND
        Product.category = "shoes"
```

Updating Views

How can I insert a tuple into a table that doesn't exist?

Employee(ssn, name, department, project, salary)

```
CREATE VIEW  Developers AS
  SELECT name, project
  FROM  Employee
  WHERE department = "Development"
```

If we make the following insertion:

```
INSERT INTO  Developers
VALUES("Joe", "Optimizer")
```

It becomes:

```
INSERT INTO  Employee
VALUES(NULL, "Joe", NULL, "Optimizer", NULL)
```

Non-Updatable Views

```
CREATE VIEW  Seattle-view  AS

        SELECT  seller, product, store
        FROM    Person, Purchase
        WHERE   Person.city = "Seattle"    AND
                Person.name = Purchase.buyer
```

How can we add the following tuple to the view?

("Joe", "Shoe Model 12345", "Nine West")

We need to add "Joe" to Person first, but we don't have all its attributes

Week 6 (Week 12)

Materialized Views

More Complex OLAP operations with SQL

References:

(a) A Conceptual Poverty Mapping Data Model
Link: https://www.researchgate.net/figure/Key-thematic-layers-for-poverty-spatial-data-modeling_fig2_229724703

(b) Relational Database relationships
https://www.youtube.com/watch?v=C3icLzBtg8I

(c) https://courses.ischool.berkeley.edu/i202/f97/Lecture13/DatabaseDesign/sld002.htm

(d) https://nexwebsites.com/database/database-management-systems/

(e) Acknowledgement – Thanks to http://courses.cs.washington.edu/courses/cse544/ for providing part of this presentation.

(f) Acknowledgement – Thanks to © Silberchatz, Korth and Surdashan for providing part of this presentation.

(e) Malinowski, Elzbieta, Zimányi, Esteban (2008) *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications .* Springer Berlin Heidelberg. Copyright © 2008 Elzbieta Malinowski & Esteban Zimányi