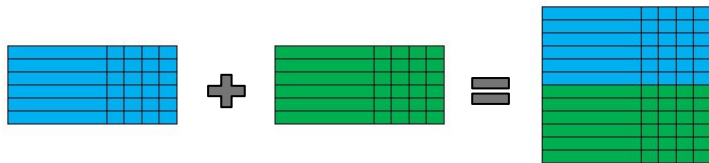


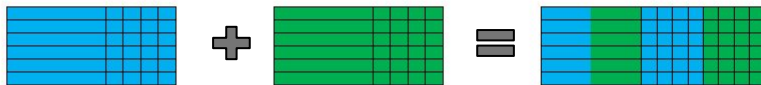
Joining Tables

Data Blending

- Vertical Blending
 - Set operations



- Horizontal Blending
 - Join



Relationship between Tables

- Need to lookup a value from another table: Instructor for a course



JOIN execution

- For each row in one table, go over each row in another table to find the rows with matching values.

Course	
Course Name	Instructor Name
Introduction to the Force	Obi-wan Kenobi
Introduction to the Force	Obi-wan Kenobi

=
↔

Instructor	
Instructor Name	Instructor Affiliation
Obi-wan Kenobi	Jedi
Darth Sidious	Sith

Match
No Match

Lightsaber for Dummies	Obi-wan Kenobi
Lightsaber for Dummies	Obi-wan Kenobi

=
↔

Obi-wan Kenobi	Jedi
Darth Sidious	Sith

Match
No Match

Throat checking 101	Darth Sidious
Throat checking 101	Darth Sidious

↔
=

Obi-wan Kenobi	Jedi
Darth Sidious	Sith

No Match
Match

JOIN execution

- Result:

Course	
Course Name	Instructor Name
Introduction to the Force	Obi-wan Kenobi
Lightsaber for Dummies	Obi-wan Kenobi
Throat checking 101	Darth Sidious

=

Instructor	
Instructor Name	Instructor Affiliation
Obi-wan Kenobi	Jedi
Obi-wan Kenobi	Jedi
Darth Sidious	Sith

=

Match
Match
Match

- The end result is equivalent of applying a WHERE clause of
`Instructor.Instructor_Name=Course.Instructor_Name`

Join

- Linking two tables by matching common column(s)
 - List two (or more) table names in the FROM clause, separated by comma
 - Add join condition to WHERE clause

```
SELECT Instructor_Affiliation
FROM Instructor, Course    -- Note the comma!!!
WHERE Instructor.Instructor_Name=Course.Instructor_Name
```

Qualifying Columns and Tables

- How to qualify a column (identify which table it belongs to)

```
SELECT Instructor.Instructor_Affiliation  
FROM Instructor, Course  
WHERE Instructor.Instructor_Name=Course.Instructor_Name
```

- How to qualify a table using alias: Add an alias after table name.

```
SELECT i.Instructor_Affiliation  
FROM Instructor i, Course c  
WHERE i.Instructor_Name = c.Instructor_Name
```

Table Alias

- You can add an alias for tables in SELECT statement

```
SELECT Instructor_Name  
FROM Instructor AS Jedi_Instructors
```

- Keyword `AS` is optional

Join with JOIN

Mixed Usage of WHERE Clause

- Mixed usage of WHERE clause: Readability issue

```
SELECT i.Instructor_Affiliation
FROM Instructor i, Course c
WHERE i.Instructor_Name = c.Instructor_Name
AND c.Course_Name LIKE '%Force%'
```

- Imagine joining 12 tables in one query, with 13 filtering conditions!

JOIN ON

- Separate join from filtering conditions
- Easier to see join linkage when joining multiple tables

```
SELECT i.Instructor_Affiliation  
FROM Instructor i, Course c  
WHERE i.Instructor_Name =  
c.Instructor_Name
```



```
SELECT i.Instructor_Affiliation  
FROM Instructor i  
JOIN Course c  
ON i.Instructor_Name =  
c.Instructor_Name
```

JOIN ON

- JOIN syntax:
 - Keyword JOIN following FROM clause
 - Followed by table name
 - Followed by ON keyword, followed by join condition

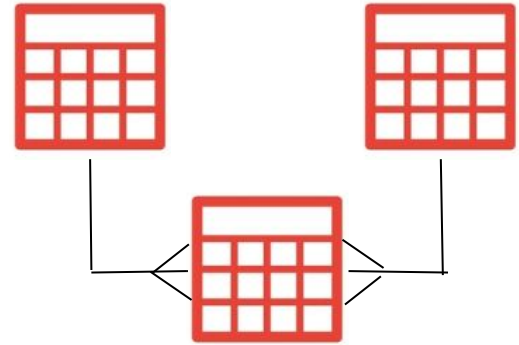
```
SELECT <qualified column list>  
FROM <table 1> <alias 1>  
JOIN <table 2> <alias 2>  
    ON <matching column 1> = <matching column 1>
```

Multi Table Join

Star Join

- Join from one table to multiple tables

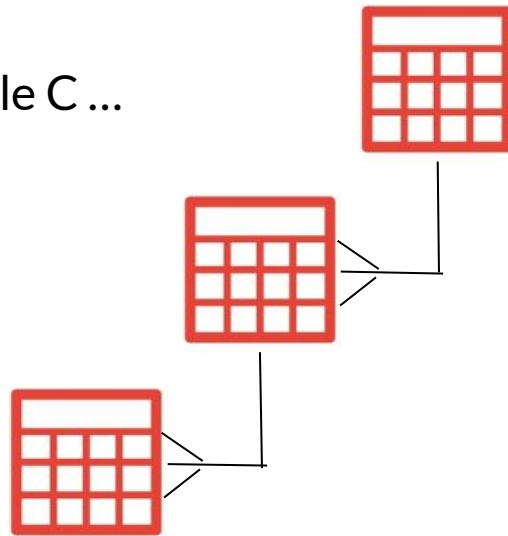
```
SELECT c.Course_Name, s.Student_Name  
FROM Registration r  
JOIN Student s  
    ON r.Student_ID = s.Student_ID  
JOIN Course c  
    ON r.Course_ID = c.Course_ID
```



Chain Join

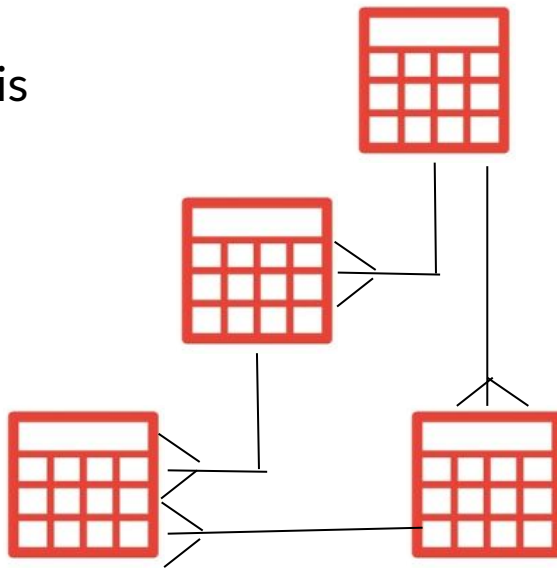
- Join from table A to table B, then from table B to table C ...
 - Avoid looping joins

```
SELECT c.Course_Name,  
i.Instructor_Affiliation  
FROM Registration r  
JOIN Course c  
  ON r.Course_ID = c.Course_ID  
JOIN Instructor i  
  on c.Instructor_Name = i.Instructor_Name
```



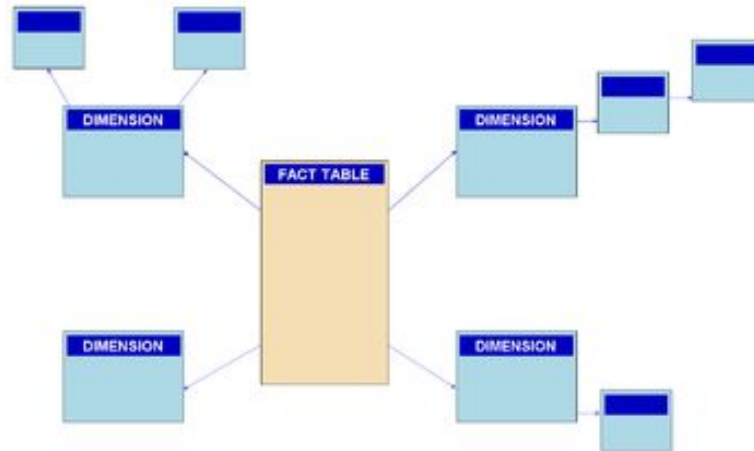
Chain Join

- Avoid looping joins
 - May cause data loss if there is any data inconsistency



Snowflake Schema

- A combination of star join and chain join
 - Most common usage of SQL query: snowflake schema



JOIN vs Normalization

- There are two types of data usage:
 - Operational (transactional):
 - Fast. Single entry. Requires integrity.
 - Requires normalization to avoid complexity and redundancy.
 - Analytical:
 - Historical statistics. Involving a variety of information
 - Needs to join data from multiple places to perform complex transformation/calculation

Columns in Join

Natural Join

- For this course, all joins are performed on one column with same names from different tables, i.e., different entities will join each other on the same attribute.
 - This is called Natural Join
- SQL can join tables on multiple columns, or columns with different names
 - This is more common in real world

Join on Multiple Columns

- Join can match on multiple columns

```
SELECT <qualified column list>
FROM <table 1> <alias 1>
JOIN <table 2> <alias 2>
    ON <matching column 1> = <matching column 1>
    AND <matching column 2> = <matching column 2>
WHERE <boolean expression>
```

Join on Different Column Names

- SQL can match tables on columns with different names
 - Same syntax

```
SELECT <qualified column list>  
FROM <table 1> <alias 1>  
JOIN <table 2> <alias 2>  
    ON <column name 1> = <column name 2>
```

- Still a best practice to using table alias on all columns

Non-equi JOIN

- Associate two tables with non-equal operators (>, <, BETWEEN...AND)
 - Frequently seen in slow-changing dimension: value changes over time

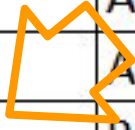
```
SELECT *
FROM Transaction t
JOIN Coupon c
  ON t.Coupon_ID = c.Coupon_ID
  AND t.Transaction_Date
      BETWEEN c.Effective_Date
      AND c.Termination_Date
```

Self Join

Self Join

- When a table is referencing itself, e.g., employee's supervisor is also an employee, it needs to join itself, i.e., to perform a self join.
 - Find the supervisor of supervisors

Employee	Supervisor
Bill	Alice
Tom	Alice
Alice	Rachel



Self Join

- Matching the Supervisor column to the Employee column

```
SELECT s.Supervisor  
FROM Employee e  
JOIN Employee s  
On e.supervisor = s.employee
```

Employee	Supervisor
Bill	Alice
Tom	Alice
Alice	Rachel



Employee	Supervisor
Bill	Alice
Tom	Alice
Alice	Rachel

Outer Join

Regular Join Review

- Result:

Course	
Course Name	Instructor Name
Introduction to the Force	Obi-wan Kenobi
Lightsaber for Dummies	Obi-wan Kenobi
Throat checking 101	Darth Sidious

=

Instructor	
Instructor Name	Instructor Affiliation
Obi-wan Kenobi	Jedi
Obi-wan Kenobi	Jedi
Darth Sidious	Sith

=

Match

Match

Match

- Instructors with no courses in Course table will be ignored. Result is the common set between the two tables, which is called “Inner Join”.
- What if we want to see all instructors, even for those with no course?

Inner Join and Outer Join

- To preserve all data from a certain table(s) during join, use “Outer Join”.



Outer Join Syntax

- Left outer join

```
SELECT c.Course_Name, i.Instructor_Name  
FROM Instructor i  
LEFT OUTER JOIN Course c  
    on c.Instructor_Name = i.Instructor_Name
```

- FROM <table 1> LEFT OUTER JOIN <table 2>
- **Can be simplified as** FROM <table 1> LEFT JOIN <table 2>
- Left/Right is based on FROM/JOIN clause order. The order in ON clause does not matter.

Outer Join Result

- Left outer join

```
SELECT c.Course_Name, i.Instructor_Name  
FROM Instructor i  
LEFT OUTER JOIN Course c  
    on c.Instructor_Name = i.Instructor_Name
```

- Data from left table will be preserved (all instructors will show up)
- If there is no data from right table, will show NULL

Outer Join Types

- Three types of outer join: Replace the word LEFT with RIGHT or FULL
 - LEFT OUTER JOIN: Preserve the data for table on the left
 - RIGHT OUTER JOIN: Preserve the data for table on the right
 - FULL OUTER JOIN: Preserve the data for both tables
- LEFT JOIN and RIGHT JOIN are reversible:

`T1 LEFT JOIN T2 = T2 RIGHT JOIN T1`

Join on Key

Regular Join Review

- Result:

Course	
Course Name	Instructor Name
Introduction to the Force	Obi-wan Kenobi
Lightsaber for Dummies	Obi-wan Kenobi
Throat checking 101	Darth Sidious

=

Instructor	
Instructor Name	Instructor Affiliation
Obi-wan Kenobi	Jedi
Obi-wan Kenobi	Jedi
Darth Sidious	Sith

=

Match

Match

Match

Join Without Key

- What if we forget the ON clause?

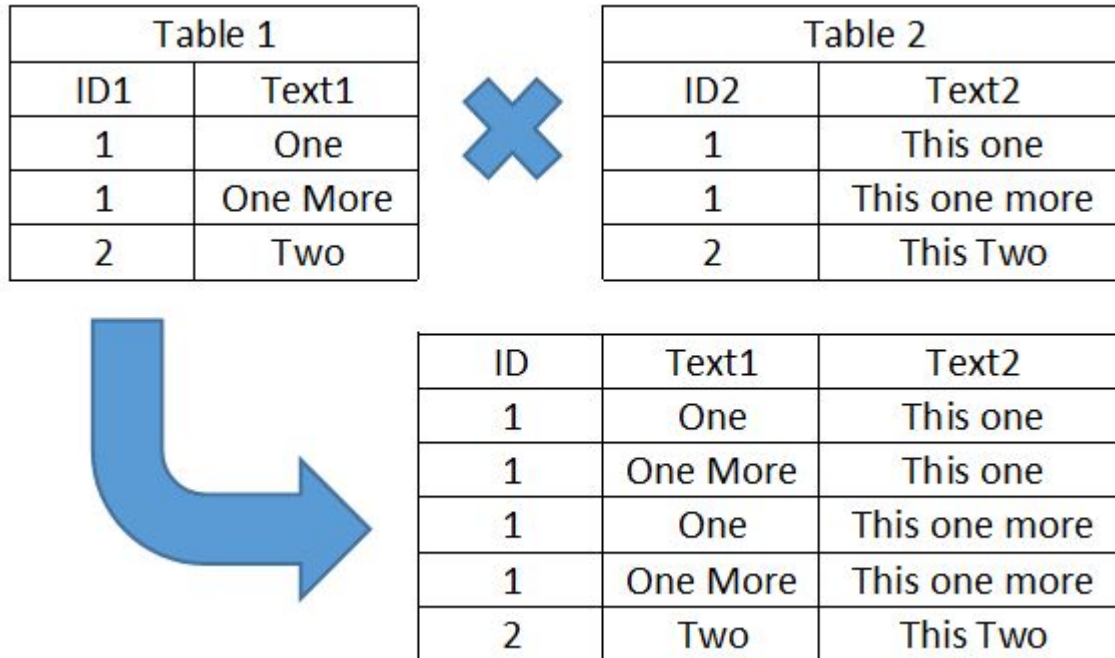
```
SELECT *  
FROM Course  
JOIN Instructor
```

- Each row in Course table will join with each row in Instructor table
- Cartesian join (aka cross join or cartesian product)
- Modern DBMS usually don't allow this but can do it with WHERE

Partial Cross Join

- If both sides are not unique, partial cross join will happen
 - Instructor table joins itself on Affiliation column
 - The join is still based on some conditions. So it is called “partial” cross join

Partial Cross Join



Join on Key

- The join key usually should be the primary key of one of the two tables
 - It is called “foreign key” on the other table

Foreign Key and Referential Integrity

OUTER JOIN

- Outer join has its own issue
 - Unknown result + NULL in value
 - Performance in handling
- Just because you can do it, doesn't mean it is a good solution
 - Avoid whenever possible
 - Enforce proper data values from DBMS

Why OUTER JOIN

- Outer join is necessary when the columns joined on have values not existing in the corresponding columns in the other tables
- If we want to eliminate outer join, we must make sure columns from both tables have same values

Referential Integrity

Referential integrity ... requires that if a value of one attribute of a relation references a value of another attribute, then the referenced value must exist.

Foreign Key

- The join key usually should be the primary key of one of the two tables
 - It is called “foreign key” on the other table
- Foreign key is a column constraint that defines a restriction on data
 - Column in one table must have all the values in the corresponding column in another table

Foreign Key

- CREATE TABLE column constraint

```
CREATE TABLE products (  
    product_no integer PRIMARY KEY,  
    name text  
);
```

```
CREATE TABLE orders (  
    order_id integer PRIMARY KEY,  
    product_no integer REFERENCES products (product_no)  
);
```

Foreign Key Constraint Violation

- You will see an error when try to insert a value in `order.product_no` that is not in `product.product_no`
 - You will see an error when you try to delete a value from `product.product_no` if the value is in `order.product_no`
 - Must delete from `order.product_no` first: cascade delete
 - PostgreSQL allows you to perform a `CASCADE DELETE` but usually don't do it: You may accidentally delete useful data
- ```
product_no integer REFERENCES products (product_no)
ON DELETE CASCADE
```