SQL

Structured Query Language

SQL Agenda

Introduction to SQL

Querying Data

Sorting Data

Filtering Data

Group By

Aggregate Functions

Data Definition Language (DDL)

Constraints

Data Manipulation Language (DML)

Joining Multiple Tables

Conditional Expressions

Filtering Data Part 2

Subquery

Using SET Operators

Data Definition Language (DDL)

Data Types Explained

Depends on DBMS

String, Numeric, Date and Time

Examples

CHAR, VARCHAR, BINARY, BLOB
BIT, BOOLEAN, INT, FLOAT, DOUBLE, DECIMAL
DATE, DATETIME, TIMESTAMP

PostgreSQL Data Types

Boolean (TRUE, FALSE or NULL)

Character - char, varchar, text

Numeric - Integer and Floating-point number

Temporal - Date, Time, Timestamp, Interval

UUID, Array, JSON, hstore, others

CREATE TABLE

CREATE TABLE

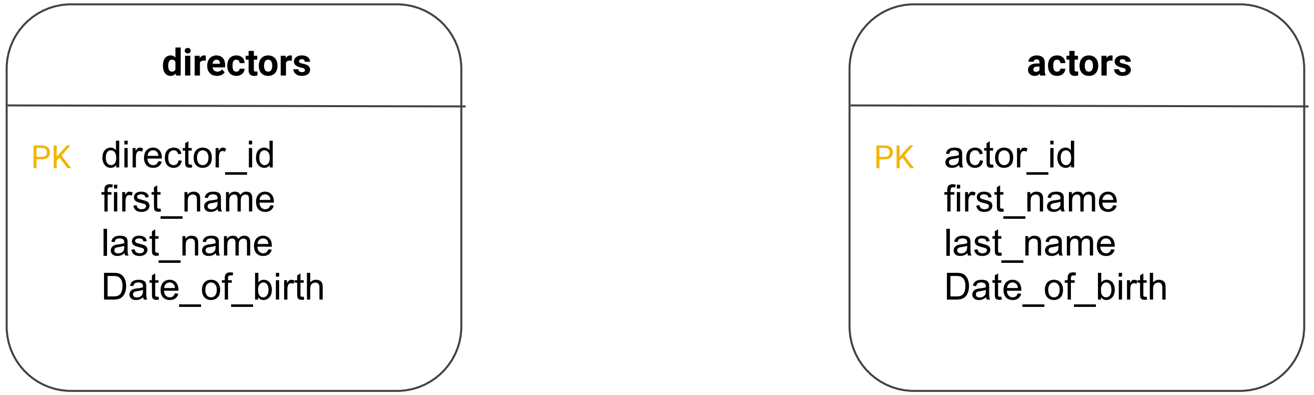
Example of Creating the director table

Do it...

CREATE TABLE

(see next slide for details)

Create Tables



DROP, ALTER,

```
DROP TABLE IF EXISTS actors;

ALTER TABLE courses ADD credit_hours INT NOT NULL;
```

TRUNCATE TABLE

Remove all data in a table efficiently and fast

```
TRUNCATE TABLE table_name;
```

```
TRUNCATE TABLE table_name1, table_name2, ...;
```

TRUNCATE TABLE vs DELETE

Logically the same effect to remove all data with some differences

DELETE - Logs, Allows Rollback
TRUNCATE - No Chance (Exception with a transaction that hasn’t committed)

DELETE - Foreign Key OK
TRUNCATE - Not OK (must use delete)

DELETE - Fires Delete Trigger
TRUNCATE - Does Not

DELETE - Can delete partial data
TRUNCATE - Removes all Data

Got it?

`Add TRUNCATE TABLE to directors and actors`

Using SQL Constraints

Constraints

PRIMARY KEY – show you how to define a primary key for a table.

FOREIGN KEY – walk you through the steps of enforcing the relationship between data in two tables using the foreign key constraint.

UNIQUE – ensure the uniqueness of values in a column or a set of columns.

NOT NULL – ensure that the values inserted into or updated to a column are not NULL.

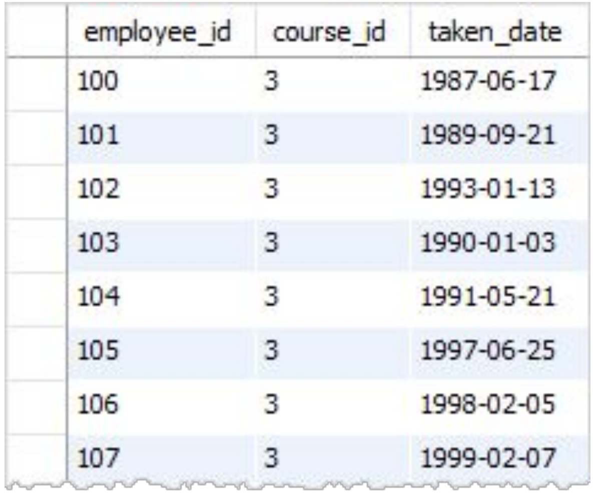
CHECK – validate data before it is stored in one or more columns based on a Boolean expression

Primary Key

Uniquely identify each row in the table

Composite Key - two or more columns

```
CREATE TABLE projects (
    project_id INT PRIMARY KEY,
    project_name VARCHAR(255),
    start_date DATE NOT NULL,
    end_date DATE NOT NULL
);
```



employee_id	course_id	taken_date
100	3	1987-06-17
101	3	1989-09-21
102	3	1993-01-13
103	3	1990-01-03
104	3	1991-05-21
105	3	1997-06-25
106	3	1998-02-05
107	3	1999-02-07

Foreign Key

Link between two tables

```
CREATE TABLE projects (
    project_id INT AUTO_INCREMENT PRIMARY KEY,
    ...
};

CREATE TABLE project_milestones (
    milestone_id INT AUTO_INCREMENT PRIMARY KEY,
    project_id INT,
    FOREIGN KEY (project_id)
        REFERENCES projects (project_id)
);
```

Unique

Unique values

```
CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL
);
```

NOT NULL

Non-NULL values only

```
CREATE TABLE training (
    employee_id INT,
    course_id INT,
    taken_date DATE NOT NULL,
    PRIMARY KEY (employee_id , course_id)
);
```

CHECK

Must satisfy a Boolean expression

```
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(255) NOT NULL,
    selling_price NUMERIC(10,2) CHECK (selling_price > 0)
);
```

Default Values

Default Values

- SERIAL
- DEFAULT

```
CREATE TABLE products (
    product_no SERIAL,
    price NUMERIC DEFAULT 9.99
);
```

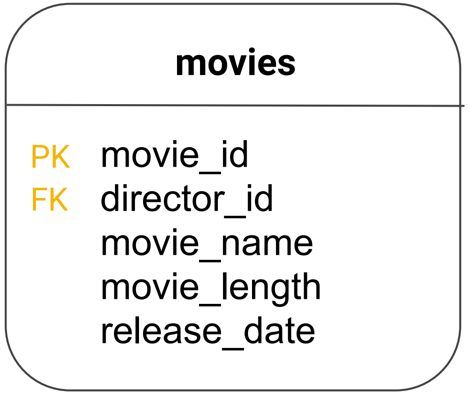
Challenge

`CREATE TABLE movies`

`(see next slide for details)`

CREATE TABLE movies

```
movie_id - SERIAL

movie_name - NOT NULL
```



Challenge

INSERT values into your the movie!

(see next slide for a little test date)

INSERT INTO test data

```
Movies
Avatar, 162, 2009-12-18, 1
Star Trek, 127, 2009-05-08, 2
```

Data Manipulation Language (DML)

INSERT

```
INSERT INTO dependents (
    first_name, last_name, relationship, employee_id
) VALUES (
    'Dustin', 'Johnson', 'Child', 178
);
```

UPDATE

```
UPDATE employees

SET last_name = 'Lopez'

WHERE employee_id = 192;
```

DELETE

```
DELETE FROM dependents

WHERE

    dependent_id = 16;
```

Challenge

INSERT values into your tables!

(see next slide for a little test date)

INSERT INTO test data

Directors
James Cameron, 1954-08-16
J.J. Abrams, 1966-06-27

Actors
Sam Worthington, 1976-08-02
Zoe Saldana, 1978-06-19
Sigourney Weaver
John Cho
Chris Pine

Database Relationships

What are Database Relationships

One-to-one: A record in one table is related to one record in another table.

One-to-many: A record in one table is related to many records in another table.

Many-to-many: Multiple records in one table are related to multiple records in another table.

Joining Multiple Tables

Joining Multiple Tables

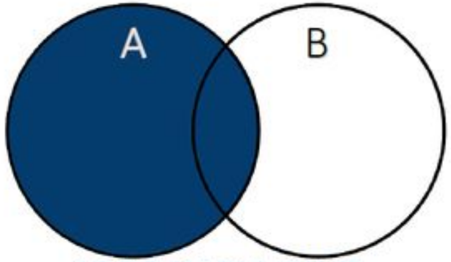
INNER JOINS - Challenge

RIGHT JOINS, LEFT JOINS, FULL JOINS - Challenge

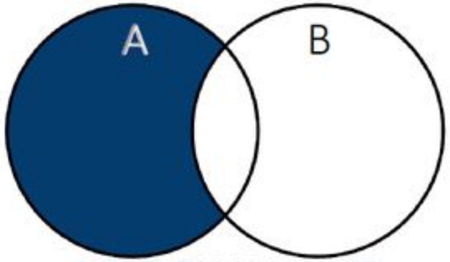
JOINING MORE THAN TWO TABLES - Challenge

UNION, UNION ALL - Challenge

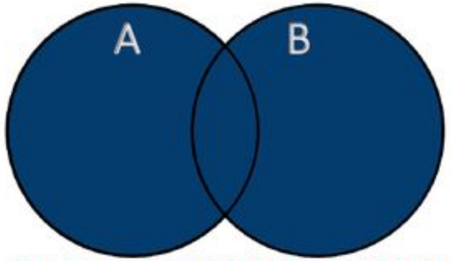
INTERSECT, EXCEPT, Challenge



LEFT INCLUSIVE



LEFT EXCLUSIVE



FULL OUTER INCLUSIVE

SQL JOINS

LEFT INCLUSIVE

```
SELECT [Select List]
FROM TableA A
LEFT OUTER JOIN TableB B
ON A.Key= B.Key
```

RIGHT INCLUSIVE

```
SELECT [Select List]
FROM TableA A
RIGHT OUTER JOIN TableB B
ON A.Key= B.Key
```

LEFT EXCLUSIVE

```
SELECT [Select List]
FROM TableA A
LEFT OUTER JOIN TableB B
ON A.Key= B.Key
WHERE B.Key IS NULL
```

RIGHT EXCLUSIVE

```
SELECT [Select List]
FROM TableA A
LEFT OUTER JOIN TableB B
ON A.Key= B.Key
WHERE A.Key IS NULL
```

FULL OUTER INCLUSIVE

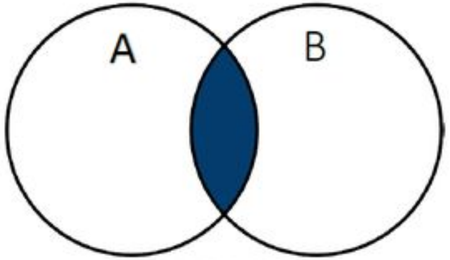
```
SELECT [Select List]
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

FULL OUTER EXCLUSIVE

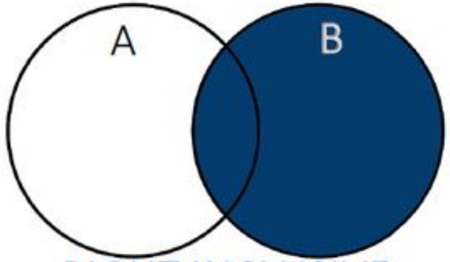
```
SELECT [Select List]
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL OR B.Key IS NULL
```

INNER JOIN

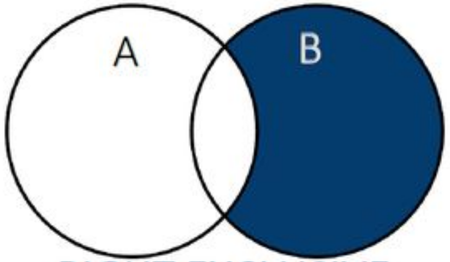
```
SELECT [Select List]
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



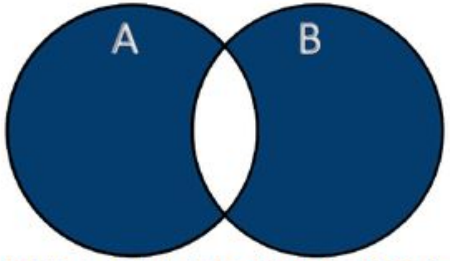
INNER JOIN



RIGHT INCLUSIVE



RIGHT EXCLUSIVE



FULL OUTER EXCLUSIVE

Joining Multiple Tables

SQL Aliases – make your query shorter and more understandable.

INNER JOIN – introduce you to the join concept and show you how to use the INNER JOIN clause to combine data from multiple tables.

LEFT OUTER JOIN – provide you with another kind of joins that allows you to combine data from multiple tables.

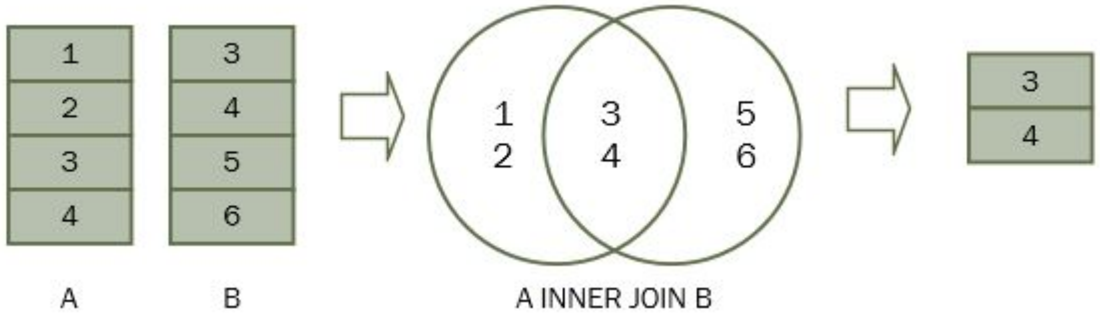
FULL OUTER JOIN – join multiple tables by including rows from both tables whether or not the rows have matching rows from another table.

CROSS JOIN – produce a Cartesian product of rows of the joined tables using the cross join operation.

SELF JOIN – join a table to itself using either the inner join or left join clause.

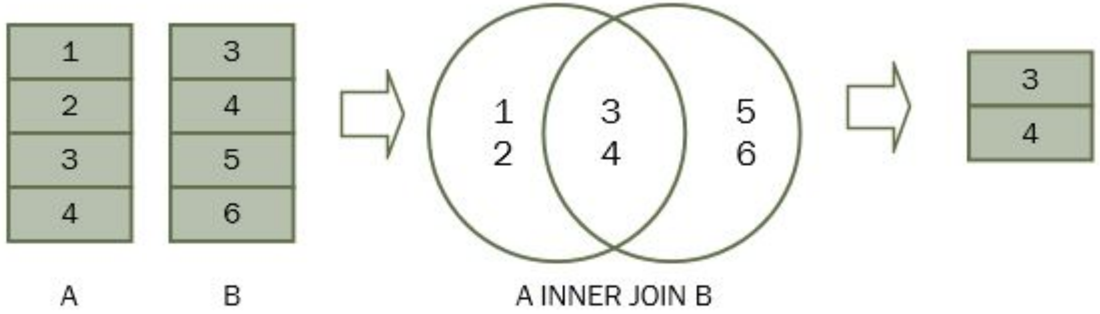
INNER JOIN

INNER JOIN



INNER JOIN

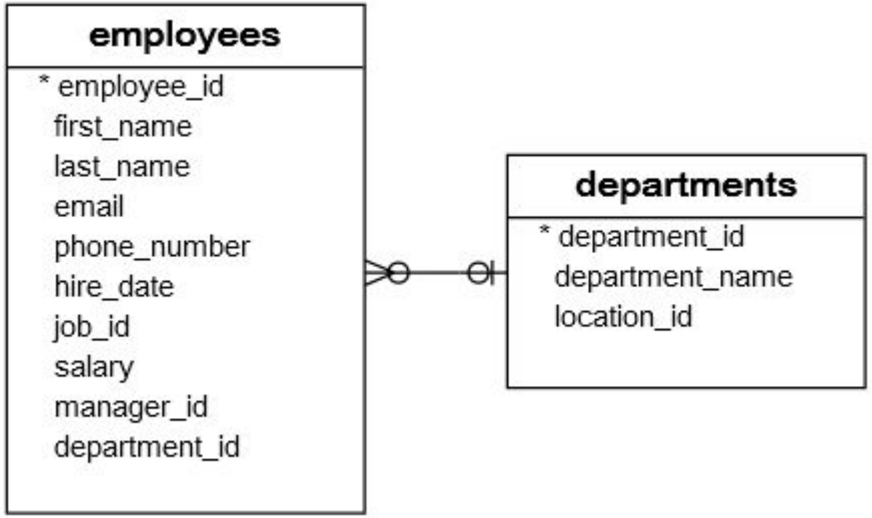
```
SELECT

  A.n

FROM A

INNER JOIN B ON
B.n = A.n;
```



INNER JOIN

```
SELECT

  A.n

FROM A

INNER JOIN B ON B.n = A.n
INNER JOIN C ON C.n = A.n;
```

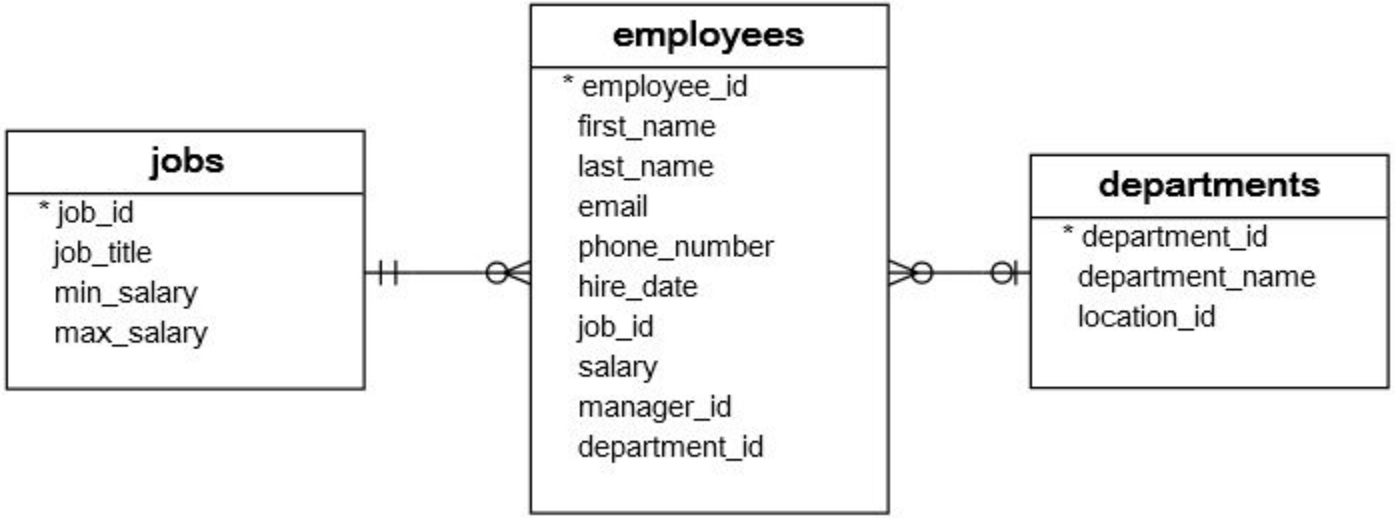
INNER JOIN


employees
* employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
manager_id
department_id
departments
* department_id
department_name
location_id

See it...

SQL INNER JOIN

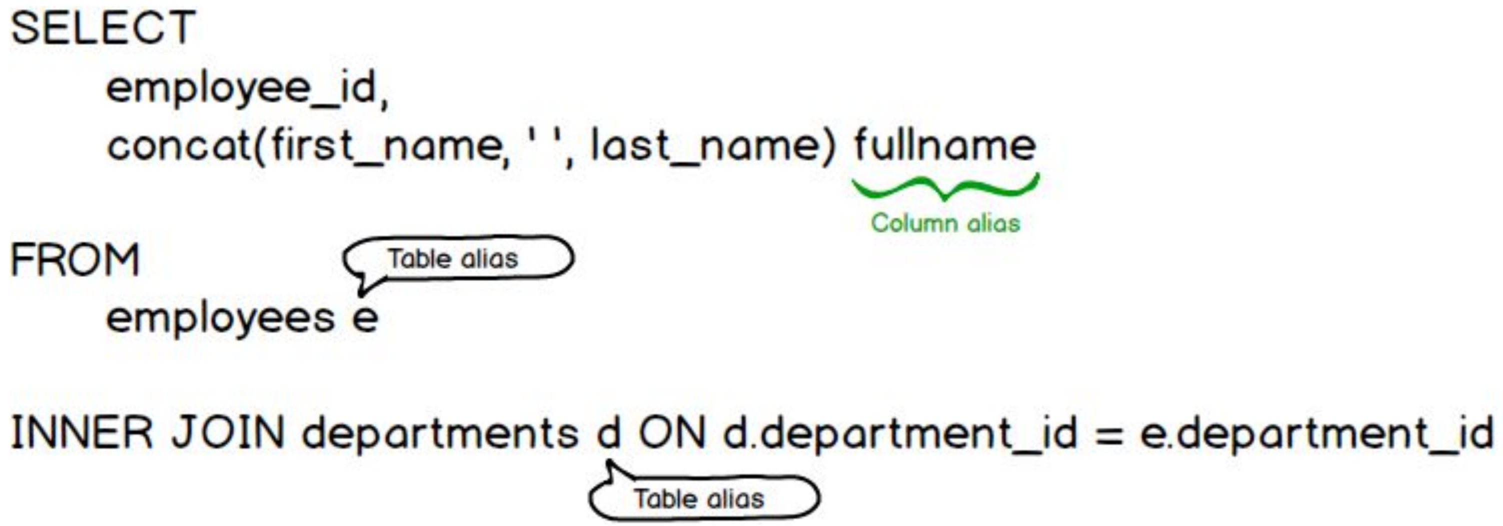
INNER JOIN


jobs
* job_id
job_title
min_salary
max_salary
employees
* employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
manager_id
department_id
departments
* department_id
department_name
location_id

Challenge

Create INNER JOIN for Movie Database

SQL Alias



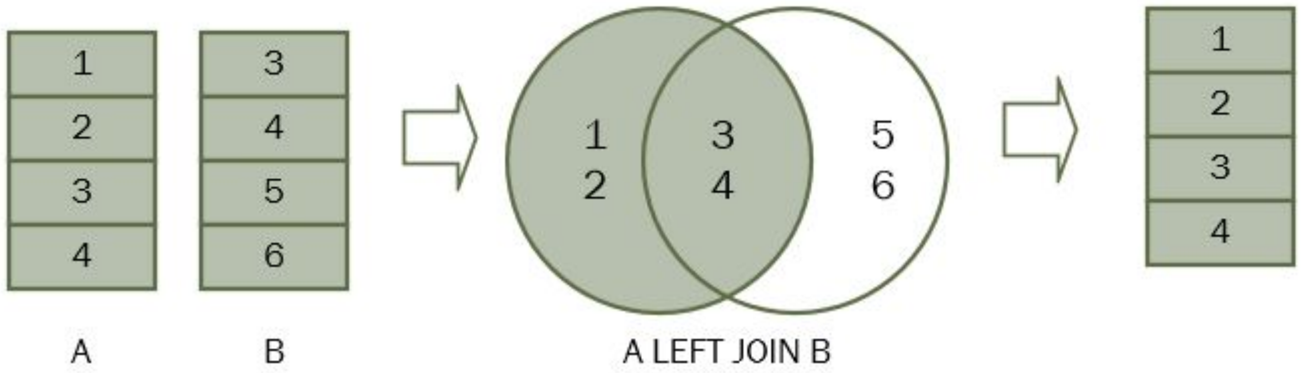
```
SELECT
        employee_id,
        concat(first_name, ' ', last_name) fullname

FROM
        employees e

INNER JOIN departments d ON d.department_id = e.department_id
```

Column alias
Table alias
Table alias

LEFT, RIGHT, FULL JOIN

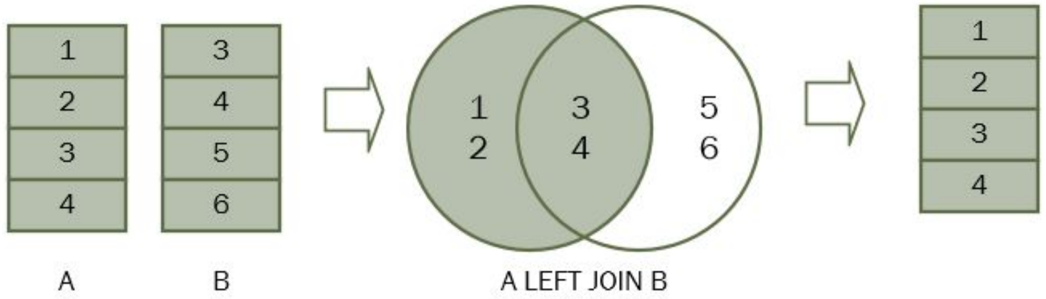
LEFT JOIN

LEFT JOIN and LEFT OUTER JOIN

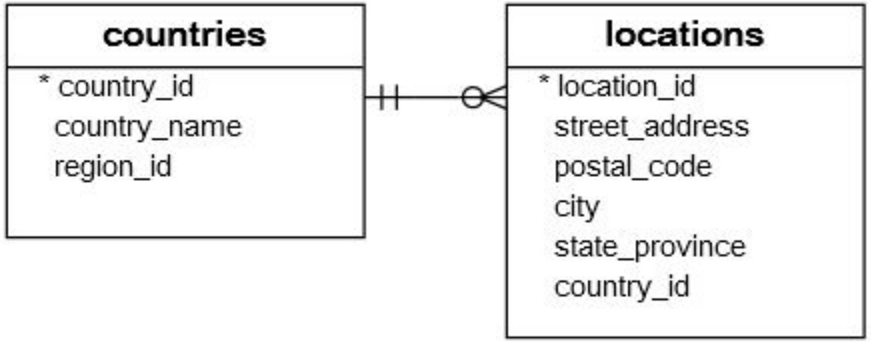


LEFT JOIN

```
SELECT
    A.n
FROM
    A
LEFT JOIN B ON B.n = A.n;
```



LEFT JOIN


countries
* country_id
country_name
region_id
locations
* location_id
street_address
postal_code
city
state_province
country_id

See it...

LEFT JOIN

Challenge

Crate LEFT JOIN with Movie Database

Cheat Sheets

Basic Queries

```
-- filter your columns
  SELECT col1, col2, col3, ... FROM table1
-- filter the rows
  WHERE col4 = 1 AND col5 = 2
-- aggregate the data
  GROUP by ...
-- limit aggregated data
  HAVING count(*) > 1
-- order of the results
  ORDER BY col2
```

Useful keywords for **SELECTS**:

DISTINCT - return unique results
BETWEEN a **AND** b - limit the range, the values can be numbers, text, or dates
LIKE - pattern search within the column text
IN (a, b, c) - check if the value is contained among given.

Data Modification

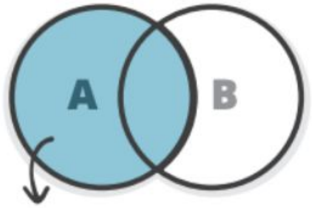
```
-- update specific data with the WHERE clause
  UPDATE table1 SET col1 = 1 WHERE col2 = 2
-- insert values manually
  INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
     VALUES (1, 'Rebel', 'Labs');
-- or by using the results of a query
  INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
     SELECT id, last_name, first_name FROM table2
```

Views

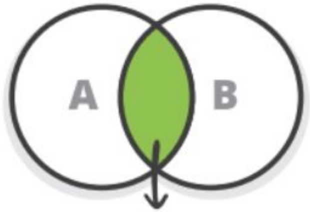
A **VIEW** is a virtual table, which is a result of a query. They can be used to create virtual tables of complex queries.

```
CREATE VIEW view1 AS
SELECT col1, col2
FROM table1
WHERE ...
```

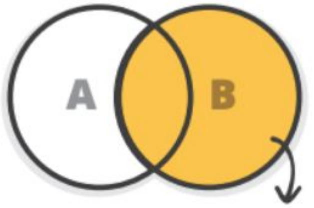
The Joy of JOINs



***LEFT OUTER JOIN** - all rows from table A, even if they do not exist in table B*



***INNER JOIN** - fetch the results that exist in both tables*



***RIGHT OUTER JOIN** - all rows from table B, even if they do not exist in table A*

Updates on JOINed Queries

You can use **JOIN**s in your **UPDATE**s

```
UPDATE t1 SET a = 1
FROM table1 t1 JOIN table2 t2 ON t1.id = t2.t1_id
WHERE t1.col1 = 0 AND t2.col2 IS NULL;
```

NB! Use database specific syntax, it might be faster!

Semi JOINs

You can use subqueries instead of **JOIN**s:

```
SELECT col1, col2 FROM table1 WHERE id IN
   (SELECT t1_id FROM table2 WHERE date >
      CURRENT_TIMESTAMP)
```

Indexes

If you query by a column, index it!

```
CREATE INDEX index1 ON table1 (col1)
```

Don't forget:

Avoid overlapping indexes
Avoid indexing on too many columns
Indexes can speed up **DELETE** and **UPDATE** operations

Useful Utility Functions

```
-- convert strings to dates:
  TO_DATE (Oracle, PostgreSQL), STR_TO_DATE (MySQL)
-- return the first non-NULL argument:
  COALESCE (col1, col2, "default value")
-- return current time:
  CURRENT_TIMESTAMP
-- compute set operations on two result sets
  SELECT col1, col2 FROM table1
  UNION / EXCEPT / INTERSECT
  SELECT col3, col4 FROM table2;
```

Union - returns data from both queries
Except - rows from the first query that are not present in the second query
Intersect - rows that are returned from both queries

Reporting

Use aggregation functions

COUNT - return the number of rows
SUM - cumulate the values
AVG - return the average for the group
MIN / MAX - smallest / largest value