BASES DE DATOS RELACIONALES

Base de Datos 2019

BASE DE DATOS Y DBMS

*"Una **base de datos** es una colección de datos organizados relevantes a un dominio que son administrados y consultados mediante un sistema de administración de base de datos (DBMS)."*

*" Un **Database Management System (DBMS)** es un sistema que permite la gestión y consulta de base de datos."*

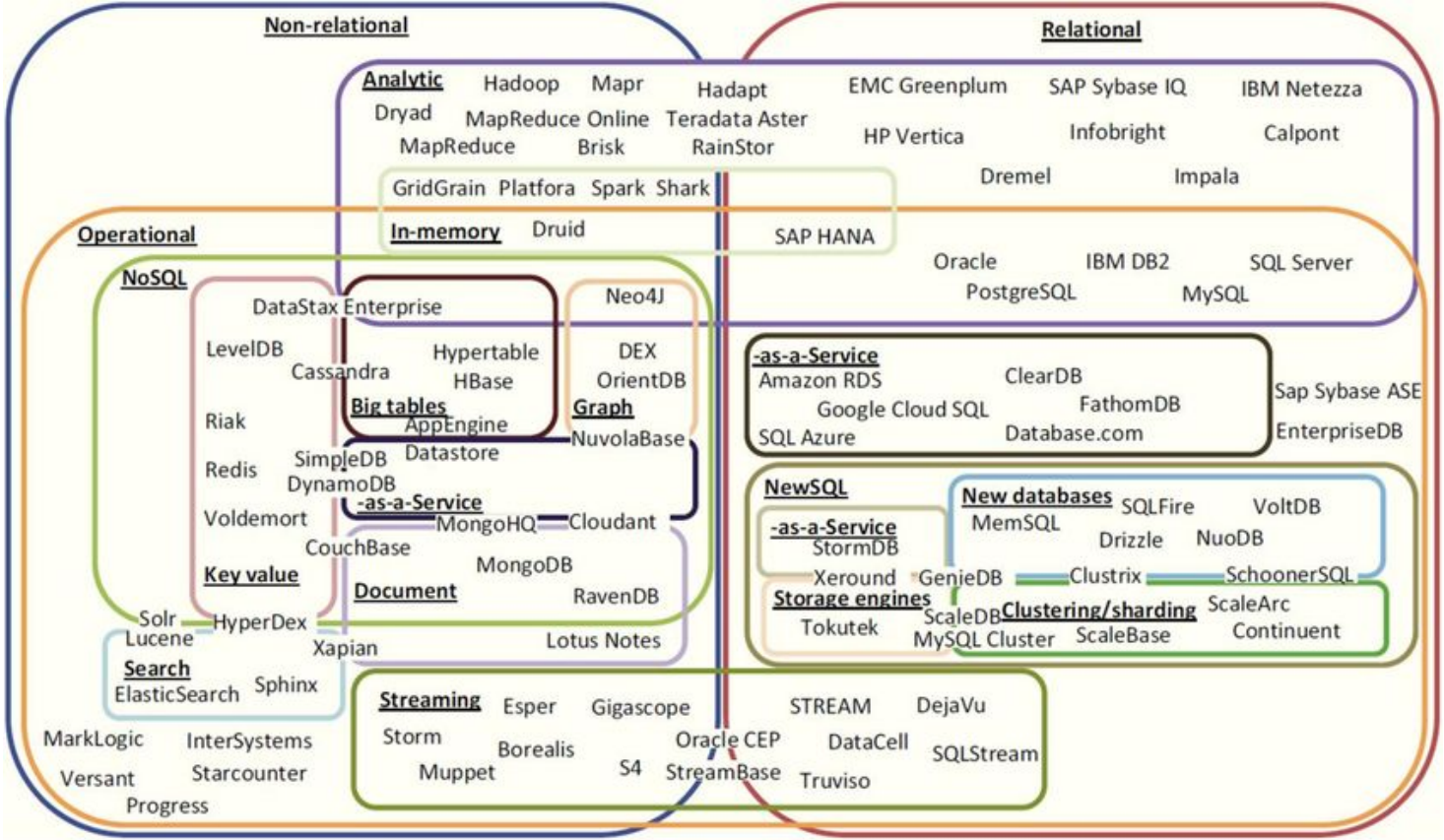
- Gestion:
  - Definición de esquema.
  - Creación de datos.
  - Creación de relaciones, actualización de datos, seguridad, etc.
- Consulta:
  - Responder preguntas sobre los datos.
  - Realizar análisis con los datos.

MODELO DE DATOS

*"Un **modelo de datos** es una colección de herramientas conceptuales para describir los datos, las relaciones entre ellos, su semántica y las restricciones de consistencia."*

- Define la estructura lógica de la base de datos.
- Impacta en la forma en que los datos son almacenados y manipulados.
- Existen diferentes modelos de datos:
  - **Modelo Relacional (SQL)**
  - **Modelos No relacionales (NoSQL):**
    - Modelo de Objetos (Realm DB)
    - Modelo de Documentos (MongoDB)
    - Modelos de Grafos (Neo4J)

CLASIFICACIÓN DE BASE DE DATOS



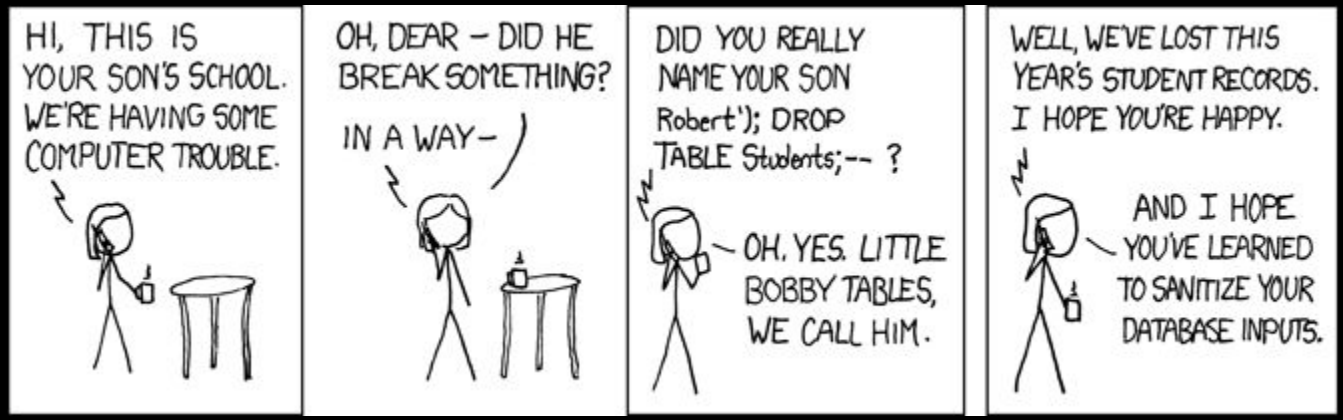
BASE DE DATOS RELACIONALES (I)

Las bases de datos relacionales (RDBMS) son una implementación del modelo relacional introducido en los 70s por Edgar Codd en *A Relational Model of Data for Large Shared Data Banks*

- La estructura básica es la **tabla** (relación).
- La tabla define **columnas** (atributos) y tiene **filas de datos** (tuplas).
- Las columnas tiene un cierto **tipo de datos** (dominio).
- Se pueden **relacionar** una o más tablas.
- Usan **SQL** como lenguaje para manipular y consultar datos.

$Columna_1$	$Columna_2$	$Columna_n$
$Valor_{11}$	$Valor_{12}$	$Valor_{1n}$
...	...	...
$Valor_{m1}$	$Valor_{m2}$	$Valor_{mn}$

SQL 101


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

SQL - UN POCO DE HISTORIA

- Desarrollado en los 70s por Donald D. Chamberlin and Raymond F. Boyce en IBM.
- Originalmente llamado Sequel, luego renombrado como **Structured Query Language (SQL)**
- En 1979, **Relational Software Inc (Oracle)** desarrolla la primer RDBMS basada en SQL.
- Es un estándar ANSI/ISO desde 1986 (SQL-86).
  - Nuevas versiones en 89, 92, 99, 2003, 2006, 2008, 2011 y 2016.
- Algunos vendors importantes:

SQL - EL LENGUAJE

SQL está compuesto por un DDL (Data Definition Language) y un DML (Data Manipulation Language).

El **DDL** permite especificar y administrar la base de datos:

- Definir el esquema de una tabla.
- Definir tipos de datos de cada columna.
- Definir restricciones de Integridad.
- Definir índices, triggers, procedimientos almacenados.
- Gestionar la seguridad (crear usuarios, dar accesos a tablas, etc).
- Definir configuraciones (tipo de almacenamiento, optimizaciones, etc).

El **DML** permite consultar y manipular el contenido de la base de datos:

- Insertar filas en una tabla.
- Actualizar filas en una tabla.
- Borrar filas de una tabla.
- Consultar filas en una o más tablas.

DDL - CREATE TABLE

```
CREATE TABLE table_name (
     col1 type1,
     col2 type2,
     ...,
     coln Dn,
     integrity-constraint1,
     ...,
     integrity-constraintk);
```

Donde:

- **table_name** es el nombre la tabla
- **col$_i$** es el nombre de una columna.
- **type$_i$** es el tipo de dato de una columna.
- **integrity-constraint$_i$** es una restricción de integridad.
- Esta es la sintaxis estándar y cada DBMS la extiende.
  - Sintaxis MySQL
  - Sintaxis PostgresSQL

DDL - CREATE TABLE - TIPOS DE DATOS

Algunos tipos de datos estándar:

- **char(n):** String de tamaño fijo n.
- **varchar(n):** String de tamaño variable, con largo maximo n.
- **int:** nros. entero (machine-dependent).
- **numeric(p,d)**: Nro de punto fijo, con precisión de p digitos y d decimales.
- **double precision:** Nro. de punto flotante de doble precisión.
- **json:** Objetos JSON
- **date:** fechas sin componente de tiempo.
- **datetime:** fechas con componente de tiempo.

Recordar:

- La implementación puede variar según el DBMS.
  - http://bit.ly/2woWbLU
  -
- Cada DBMS agrega sus propios tipos de datos.
  - MySQL Data Types
  - PostgresSQL Data Types
- **Siempre leer la documentación!!!!**

DDL - CREATE TABLE - RESTRICCIONES DE INTEGRIDAD (I)

Sirven para asegurar la consistencia de los datos en la base de datos.

- **PRIMARY KEY(col_1, ...,col_n)**
  - Define a las columnas col_i como claves primarias (PK) de la tabla.
  - Las PKs tienen que ser **únicas** y **no nulas.**
  - Las PKs no deberían cambiar nunca
- **NOT NULL**
  - Indica que una columna no puede tener valores nulos.
- **UNIQUE**
  - Indica que una columna no puede tener valores
  - repetidos.

- **PRIMARY KEY ≈ UNIQUE NOT NULL**
- Se recomienda que el PK sea “pequeño”.
- Generación automática de IDs:
  - MySQL: **AUTO_INCREMENT** attribute
  - PostgreSQL(<10): **SERIAL** data type
  - PostgreSQL(10): **IDENTITY** columns

DDL - CREATE TABLE - RESTRICCIONES DE INTEGRIDAD (II)

- **FOREING KEY (col_1, ...,col_n) REFERENCES T [ON DELETE option] [ON UPDATE option]:**
  - Indica que los valores de las columnas col_1,...,col_n deben corresponderse con los valores de las claves primarias de la tabla T.
  - Si la restricción es violada, **ON DELETE | ON UPDATE** establecen como actuar.
  - **option: CASCADE | SET NULL | SET DEFAULT**
- **CHECK (condition):**
  - Indica que el predicado condition debe ser verdadero para toda fila en la tabla.
  - Según el estándar, **condition** puede ser una subquery.

- **FOREING KEY** es una version mas especifica de **integridad referencial.**
- En una cadena de dependencias FOREING KEY, una modificación en una punta se puede propagar.
- **CHECK** es ignorado por MySQL.
- **CHECK** es soportado por PostgreSQL
  - No soporta subqueries arbitrarias.
  -

DDL/DML - ACTUALIZACIÓN DE TABLAS

```
DROP TABLE table_name;

ALTER TABLE table_name
ADD COLUMN col_1 type_1;

ALTER TABLE table_name
DROP COLUMN col_1;

INSERT INTO table_name (col_1,...,col_n)
VALUES (val_1,...,val_n);

DELETE FROM table_name WHERE condition;

UPDATE table_name SET col_1 = val_1, ...,
WHERE condition;
```

- **ALTER TABLE** hace mucho más que agregar/quitar columnas.
  - MySQL Syntax
  - PostgresSQL Syntax
- Cuando se agrega una columna, todas las filas existentes son asignadas **NULL** en la nueva columna.
- En un **INSERT** se pueden insertar múltiples filas.
- **CUIDADO al usar DELETE/UPDATE!!**
  - Siempre especificar el WHERE

IT'S DEMO TIME

CONSULTAS EN SQL

```
SELECT select_expr
FROM table_expr
[WHERE where_condition]
[ORDER BY order_expr]
```

- **select_expr** es un listado de una o más columnas.
- **table_expr** es un listado de una o más tablas.
- **where_condition** es un predicado.
- **order_expr** es una lista de expresiones del tipo **{col | alias | pos} [ASC|DESC]**
- El resultado de una consulta es una **tabla.**
- Ejemplo:

```
SELECT name FROM instructor;
```

CONSULTAS EN SQL - SELECT

- Por defecto SQL permite duplicados en los resultados de una query.
- Para eliminar duplicados, usar **DISTINCT**

```
SELECT DISTINCT name FROM instructor;
```

- Si queremos seleccionar todas las columnas, usamos el *.

```
SELECT * FROM instructor;
```

- Se puede usar un literal como columna.

```
SELECT 'UNC', name FROM instructor;
```

- Las columnas se pueden renombrar.

```
SELECT name AS fullname
FROM instructor;
```

- Se pueden crear columnas con expresiones aritméticas (+,-,*,/).

```
SELECT name AS fullname,
       salary/40 AS usd_salary
FROM instructor;
```

- SQL es case-insensitive.

CONSULTAS EN SQL - FROM

- **FROM** permite especificar las tablas involucradas en la query.
- **FROM T_1, T_2, ... , T_n** realiza el producto cartesiano T_1 x T_2 x ... x T_n.

```
SELECT * FROM instructor, teaches
```

- Cuidado cuando la cardinalidad de T_i no es trivial.
- Se pueden renombrar las tablas.

```
SELECT t.ID, i.ID
FROM instructor AS i, teaches AS t
```

instructor

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

teaches

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

FROM instructor, teaches

Inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Pinance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Pinance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Pinance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

CONSULTAS EN SQL - WHERE

- WHERE permite especificar condiciones que el resultado debe satisfacer.

```
SELECT *
FROM instructor
WHERE dep_name = 'Finance';
```

- Se pueden combinar predicados usando **AND, OR, NOT**.

```
SELECT *
FROM instructor
WHERE dep_name = 'Finance'
      AND salary <= 90000;
```

- SQL provee el operador **LIKE** para matching sobre strings.
  - **%** matchea cualquier substring.
  - **_** matchea cualquier caracter.

```
SELECT *
FROM instructor
WHERE dep_name LIKE '%inan%';
```

- Tambien provee el operador **BETWEEN**.

```
SELECT *
FROM instructor
WHERE dep_name = 'Finance'
      AND salary BETWEEN 9000 AND 10000;
```

CONSULTAS EN SQL - ORDER BY

- **ORDER BY** permite ordenar los resultados.

```
SELECT *
FROM instructor
WHERE dep_name = 'Finance'
ORDER BY salary DESC;
```

- Se puede ordenar por más de una columna.

```
SELECT *
FROM instructor
WHERE dep_name = 'Finance'
ORDER BY salary DESC, name ASC;
```

- Por defecto el orden es ascendente.

```
SELECT *
FROM instructor
WHERE dep_name = 'Finance'
ORDER BY salary, name;
```

- Se puede usar el nro de columna para ordenar.

```
SELECT salary, name
FROM instructor
WHERE dep_name = 'Finance'
ORDER BY 1 DESC, 2;
```

CONSULTAS EN SQL - AGREGACIONES

- SQL provee un conjunto de funciones de agregación.
- Una función de agregación toma una colección de valores y retorna un solo valor.
- Algunas funciones de agregación:
  - **max(C_i)**
  - **min(C_i)**
  - **avg(C_i)**
  - **count(C_i) | count(distinct C_i)**
  - **sum(C_i)**
  - **Otras en MySQL**

- Ejemplo:

```
SELECT avg(salary) AS avg_salary,
       max(salary) AS max_salary,
       min(salary) AS min_salary
FROM instructor
WHERE dept_name = 'Music';
```

CONSULTAS EN SQL - GROUP BY

```
SELECT select_expr
FROM table_expr
[WHERE where_condition]
[GROUP BY {col|alias|pos},]
[ORDER BY order_expr]
```

- **GROUP BY** permite calcular las agrupaciones sobre un grupo de filas.
- Las columnas especificadas en el **GROUP BY** definen los grupos. A igual valor, mismo grupo.
- Si se omite el **GROUP BY**, todas las filas se tratan como parte de un solo grupo.
- Las columnas en el **SELECT** que no sean agregadas deben aparecer en el **GROUP BY**.

CONSULTAS EN SQL - GROUP BY

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

```
SELECT dept_name, avg(salary) AS salary
FROM instructor
GROUP BY dept_name;
```

dept_name	*salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

CONSULTAS EN SQL - HAVING

```
SELECT select_expr
FROM table_expr
[WHERE where_condition]
[GROUP BY {col|alias|pos},]
[HAVING where_condition]
[ORDER BY order_expr]
```

- **HAVING** permite filtrar grupos sobre los valores agregados.
- Importante:
  - El predicado de HAVING se chequea **luego** de que los grupos son computados.
  - El predicado del WHERE se chequea **antes** de que los grupos sean computados.

CONSULTAS EN SQL - HAVING

ID	name	dept_name	salary
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

```
SELECT dept_name, avg(salary) AS salary
FROM instructor
GROUP BY dept_name
HAVING avg(salary) > 42000;
```

dept_name	salary
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

dept_name	avg(avg_salary)
Physics	91000
Elec. Eng.	80000
Finance	85000
Comp. Sci.	77333
Biology	72000
History	61000

CONSULTAS EN SQL - JOINS

- Las operaciones de JOIN permiten combinar dos tablas y retornan otra tabla.
- Son productos cartesianos que requieren que las filas en ambas tablas satisfagan ciertas condiciones.
- Existen 4 tipos basicos de JOIN:
  - INNER JOIN
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL JOIN
- Existen otros tipos de joins:
  - MySQL Joins

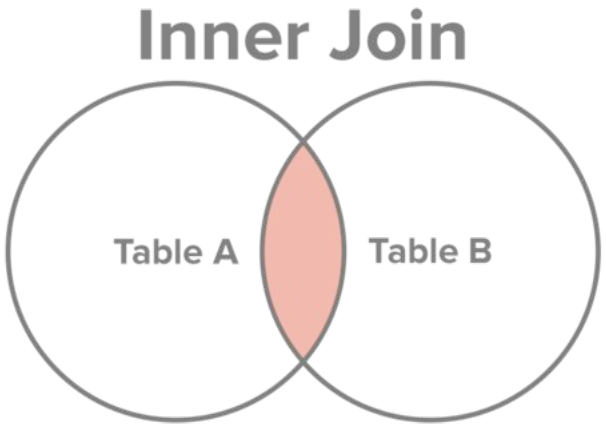
CONSULTAS EN SQL - INNER JOIN

```
SELECT ...
FROM A
INNER JOIN B ON join_condition
...
```

- **join_condition** es un predicado sobre las columnas de A y B
  - A.id = B.id
  - A.x >= B.y
- **ON join_condition** se puede reemplazar con **USING(columns)**

Interpretacion:

- Selecciona todas las filas de las tablas A y B donde la condición del join se satisface.



CONSULTAS EN SQL - INNER JOIN

course

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

```
SELECT course.*, prereq.*
FROM course
INNER JOIN prereq ON course.course_id = prereq.course_id
```

course_id	*title*	*dept_name*	*credits*	*prere_id*	*course_id*
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

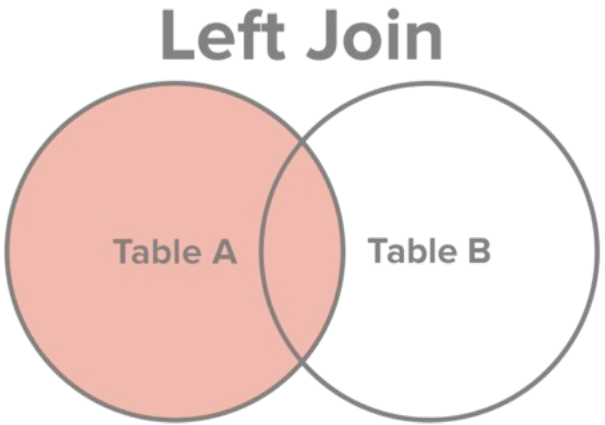
CONSULTAS EN SQL - LEFT JOIN

```
SELECT ...
FROM A
LEFT JOIN B ON join_condition
...
```

- El término **OUTER** es opcional y no tiene efecto.

Interpretacion:

- Selecciona todas las filas de las tablas A y aquellas de la tabla B donde la condición del join se satisface.



CONSULTAS EN SQL - LEFT JOIN

course

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

```
SELECT course.*, prereq.prereq_id
FROM course
LEFT JOIN prereq ON course.course_id = prereq.course_id
```

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

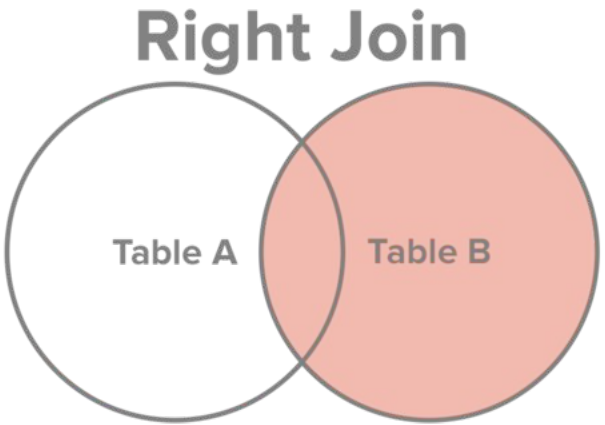
CONSULTAS EN SQL - RIGHT JOIN

```
SELECT ...
FROM A
RIGHT JOIN B ON join_condition
...
```

Interpretacion:

- Selecciona todas las filas de las tablas B y aquellas de la tabla A donde la condición del join se satisface.

- MySQL suele optimizar estos joins transformándolos en LEFT JOINS
  - Outer Join Optimization
  - Outer Join Simplification



CONSULTAS EN SQL - RIGHT JOIN

course

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

```
SELECT p.course_id, c.title, c.dept_name,
       c.credits,  p.prereq_id
FROM course AS c
RIGHT JOIN prereq AS p ON c.course_id = p.course_id
```

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

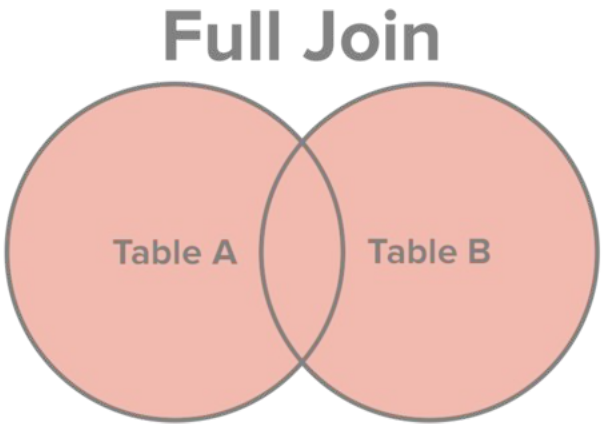
CONSULTAS EN SQL - FULL JOIN

```
SELECT ...
FROM A
FULL JOIN B ON join_condition
...
```

Interpretación:

- Selecciona todas las filas de las tablas A y B independientemente de si la condición del join se satisface.

- No es EL producto cartesiano.



CONSULTAS EN SQL - FULL JOIN

course

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

```
SELECT *
FROM course
FULL JOIN prereq USING(course_id)
```

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

TEMAS PARA ESTUDIAR

- Subqueries Anidadas
- Operaciones de conjunto.
- Cómo lidiar con los NULLS.

Referencias:

- Las filminas del libro.
- El libro.

BD-2019