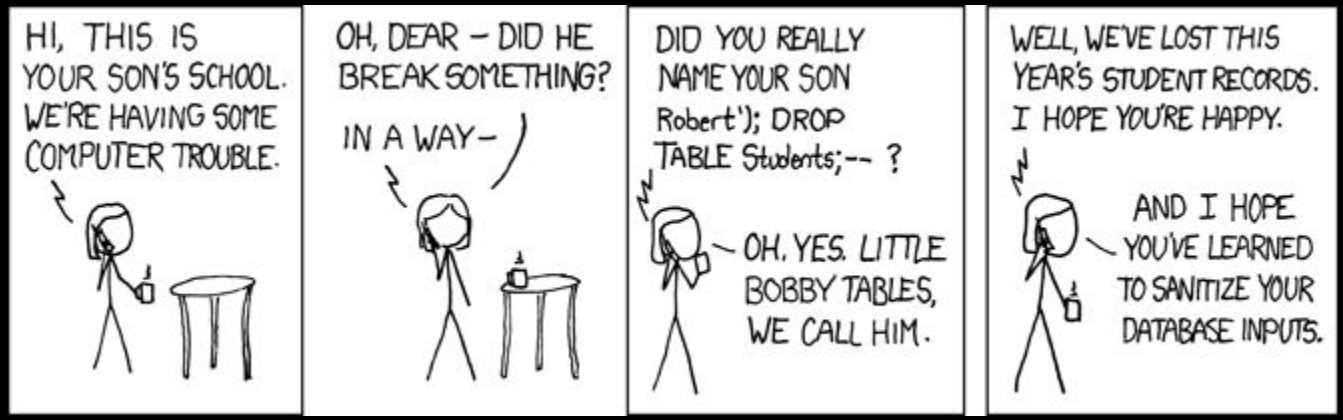
SQL 101


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.

Data Definition Language (DDL)

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

Data Manipulation Language (DML)

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 - NULL VALUES

- En una operación aritmética:

  NULL (+ | - | * | /) X = NULL

- En operaciones booleanas:

  NULL **AND** TRUE = NULL
  NULL **AND** FALSE = FALSE
  NULL **AND** NULL = NULL
  NULL **OR** TRUE = TRUE
  NULL **OR** FALSE = NULL
  NULL **OR** NULL = NULL
  NOT NULL = NULL

- Si el predicado de un WHERE evalúa a FALSE o NULL para una tupla, la misma no forma parte del resultado.
- Para testear si un valor es NULL:

```
WHERE salary IS null;

WHERE salary IS NOT null;
```

- Todas las funciones de agregación, excepto **COUNT**, ignoran los valores nulos.

TEMAS PARA ESTUDIAR (Próxima Clase)

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

Referencias:

- Las filminas del libro.
- El libro.