Hands on With SQL

Obectives

- Define and Understand DDL and DML
- Create a Simple Schema and Seed data in .sql file
- Understand how SQL handles CRUD
- Dump, then Delete you DB then restore it from the dump file
- SQL query practice with JOINS
- Understand and use 3 Aggregate functions

DDL **Data Definition Language**

(things like CREATE TABLE, DROP TABLE, ALTER, RENAME, TRUNCATE(empty for re-use) and defining data types in SQL fields)

DML **Data Manipulation Language**

(used in queries, especially CRUD operations)

How does SQL Handle CRUD?

C

R

U

D

How does SQL Handle CRUD?

- **C** INSERT INTO
- **R** SELECT
- **U** UPDATE, SET
- **D** DELETE FROM

EXAMPLES

INSERT (CREATE)

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Komal', 22, 'MP', 4500.00 );
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

SELECT (READ)

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

```
SELECT ID, NAME, SALARY FROM CUSTOMERS
```

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  1 | Ramesh   |  2000.00 |
|  2 | Khilan   |  1500.00 |
|  3 | kaushik  |  2000.00 |
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

UPDATE (UPDATE)

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+

UPDATE CUSTOMERS
SET ADDRESS = 'Pune'
WHERE ID = 6;`
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | Pune      |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

DELETE (DELETE)

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

```
DELETE FROM CUSTOMERS
WHERE ID = 6;
```

Now, CUSTOMERS table would have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

I have developed a Zen-like approach to the operating systems that people use: 'When you're ready, the right operating system will appear in your life.

Guy Kawasaki

Create your schema using .sql files

- createdb weatherapp
- mkdir SQL_FUN
- //cd into SQL_FUN
- touch db/schema.sql
- touch db/seed.sql //seed data

...in your schema

```
DROP TABLE IF EXISTS weather;
DROP TABLE IF EXISTS cities;

CREATE TABLE cities (
  id serial primary key,
  city varchar(80),
  location point
);

CREATE TABLE weather (
  id serial primary key,
  city_id int references cities(id) on delete cascade,
  temp_hi int CHECK (temp_hi > temp_lo),
  temp_lo int,
  prcp real,
  date date
);
```

. . .in your seed file

```
INSERT INTO
  cities
VALUES
  ( default, 'Boulder', point(2,5) ),
  ( default, 'Denver', point(7,2) ),
  ( default, 'Brooklyn', point(9,1) );

INSERT INTO
  weather
VALUES
  ( default, (SELECT id FROM cities WHERE city = 'Boulder'), 75, 42, 210000, now() ),
  ( default, (SELECT id FROM cities WHERE city = 'Denver'), 65, 55, 300030, now() ),
  ( default, (SELECT id FROM cities WHERE city = 'Brooklyn'), 55, 39, 120000, now() ),
  ( default, (SELECT id FROM cities WHERE city = 'Boulder'), 71, 55, 103000, ( now() - interval '1' day ) ),
  ( default, (SELECT id FROM cities WHERE city = 'Denver'), 74, 51, 300040, ( now() - interval '1' day ) ),
  ( default, (SELECT id FROM cities WHERE city = 'Brooklyn'), 72, 66, 203000, ( now() - interval '1' day ) ),
  ( default, (SELECT id FROM cities WHERE city = 'Boulder'), 81, 65, 104000, ( now() - interval '2' day ) ),
  ( default, (SELECT id FROM cities WHERE city = 'Denver'), 64, 55, 300300, ( now() - interval '2' day ) ),
  ( default, (SELECT id FROM cities WHERE city = 'Brooklyn'), 42, 36, 202000, ( now() - interval '2' day ) );
```

...from the command line

- `psql weatherapp > db/schema.sql`
- `psql weatherapp > db/seed.sql`
- `psql weatherapp`
- `\dt // JOY!!! you have tables`
- `\d cities // check your schema and foreign keys`

JOINS

Joins are where SQL gets magical.
Joins are how you utilize the relational part of the database. d
Without joins, your database is essentially useless. :(

LET'S JOIN SOME RELATIONAL DATA together

There are many joins, but INNER JOIN is your friend

Anatomy of a SQL JOIN

```
SELECT <select_list>
FROM Table_A
INNER JOIN Table_B
ON A.Key = B.Key
```

SEMANCTIC aside to make your joins easier...

- call the PK in any table 'id'
- when you reference that in another table, name it '**tableName**'_id

EXAMPLE

users		posts		comments	
id	PK	id	PK	id	PK
name		user_id	FK	user_id	FK
age		body		post_id	FK

This solves the 'ambiguous' error you may have run into

BACK TO JOINS

1. LETS JOIN cities and Weather (we do)
   (Ya'll do)
2. Write a Query to display the date and the city name where the high was over 70
3. Write a Query to display the date and the city name where the temp range was less than 8 degrees

Dumping and Restoring

DUMP

```
pg_dump dbname > outfile.sql
```

RESTORE

```
psql dbname < infile
```

Aggregate functions

(simply, they aggregate data...things like SUM, MIN, MAX, COUNT, AVG)

SQL can also compute values on the fly

ie)
column 1 :population
column 2 :GDP
column 3 (computed): population/GDP AS perCapita GDP

SQL CHEATSHEET

SQL cheat sheet

REBELLABS by ZEROTURNAROUND

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

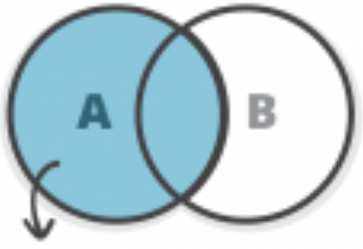
```
-- update specific data with the WHERE clause
   UPDATE table1 SET col1 = 1 WHERE col2 = 2
-- insert values manually
   INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
      VALUES (1, 'Rebel', 'Labs');
-- or by using the results of a query
   INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
      SELECT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

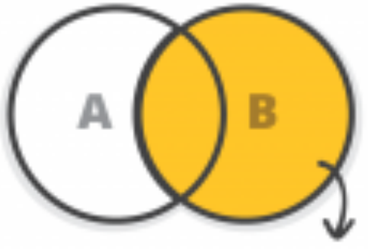
```
CREATE VIEW view1 AS
SELECT col1, col2
FROM table1
WHERE ...
```

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

More practice

- JOINS on SQL Zoo!
- Query Stack Overflow

  https://data.stackexchange.com/stackoverflow/query/new

Coming soon...

- new information to make all of this easier and nearly irrelevant!!!