SQL Intro

The Two Halves of RDBMSs

1. The storage mechanism: tables, columns & rows

2. The query mechanism: Structured Query Language (SQL)

Tables

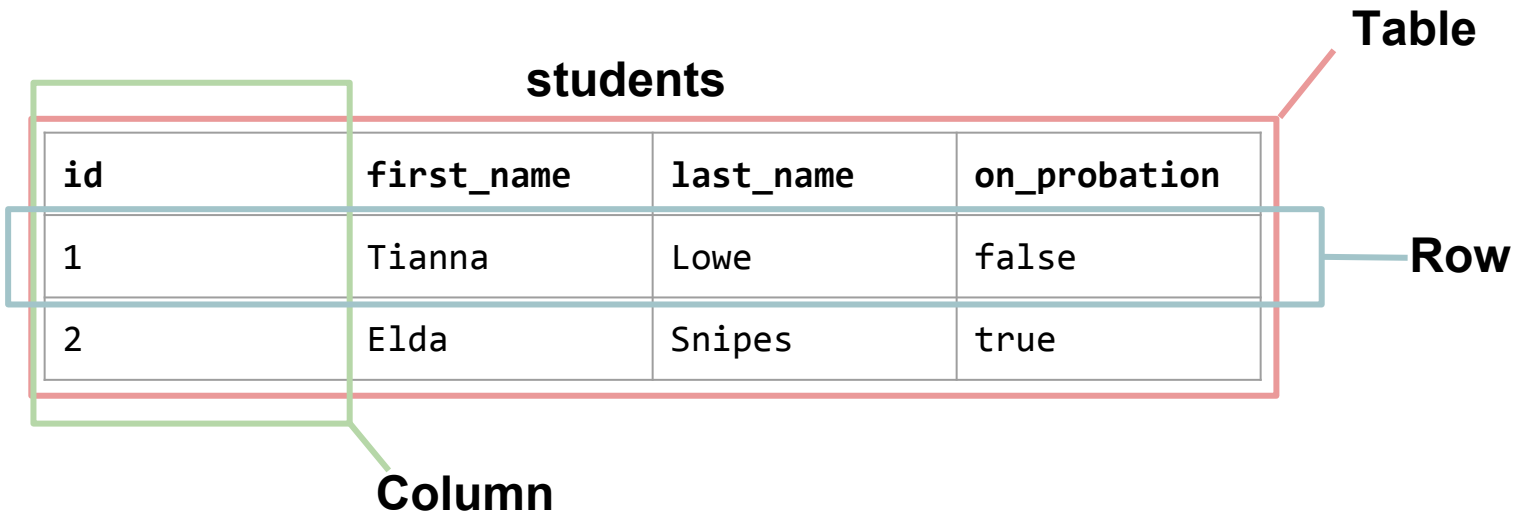
Table
students
id
first_name
last_name
on_probation
1
Tianna
Lowe
false
2
Elda
Snipes
true
Row
Column

Schema

Defined as: “A representation of a plan”. A database schema is the plan we’re using to store our data.

Primarily focused on describing three* things:

- table names
- column names
- column types

*Other parts of the schema include whether or a column must have a value (`NULL` vs `NOT NULL`), foreign keys, indexes, etc.

Tables

Tables are identified by a name:

```
CREATE TABLE students(...);

CREATE TABLE addresses(...);
```

While almost any name is valid, these are good naming conventions to follow:

- lowercase
- plural noun

It’s also a very good practice to include a unique integer column named `id` on every table you create.

Table Name Examples

What does it store?	Table Name
People	persons
Purchase Order	purchase_orders
Liking a Post	likes

Columns

Each column has a name. Lowercase, singular nouns or adjectives are a good naming convention:

```
CREATE TABLE students(
  id integer,
  first_name varchar(255),
  last_name varchar(255),
  on_probation boolean
);
```

Column Types

Each column type has a type which determines what kind of data it can hold.

```
CREATE TABLE students(
  id integer,
  first_name varchar(255),
  last_name varchar(255),
  on_probation boolean
);
```

Note: `varchar(255)` is a fancy way of saying “a string up to 255 characters long”

Column Types

Column types depend on the database system (e.g. Postgres, MySQL, MS SQL, Oracle, etc). But these column types are common:

Column type	**Description**
INTEGER	integer numbers from -2^31 to 2^31
DECIMAL	fractional number
VARCHAR([1-255)]	variable length strings from 1-255
TEXT	longer strings, up to 16KB
DATE	date, no time
TIMESTAMP	date with time

Schema Modifications

As you maintain software it’s expected that you will create new classes, write new methods and refactor existing code.

You’ll do the same to your database schema.

New features often require new tables and columns or perhaps migrating existing data to new schema.

Existing columns can also be modified to hold new types of data.

Schema Modifications

Changing a database schema often isn't "cheap". It can often require coordination with code changes and/or other tools (e.g. a third-party analytics tool or an excel formula run against a CSV export of a database table).

For this reason, *take your time* when designing a schema.

You'll often be stuck your schema choices for some time. So think about your domain's edge cases and the different directions your project might go.

If you're convinced you wish you had a different schema, the time to make the change is NOW. Making the change will never be "cheaper" than today.

Single Responsibility

Successful software is built from small modules that **do one thing** well.

As software developers you try and avoid big classes, long methods and long lines. We've learned that **smaller is always better** when it comes to software.

Single Responsibility

The same adage applies to storing data in a RDBMS. Large tables that store more than one record or model should be broken up into multiple tables.

id	first_name	last_name	on_probation
1	Tianna	Lowe	false
2	Elda	Snipes	true

students

line_1	line_2	city	state	zipcode
381 Maple St.		Chicago	IL	60657
8917 E. Rogers Blvd.	#2E	Joliet	IL	60403

addresses

Don’t Repeat Yourself

Just like you wouldn’t stand for repeating the same snippet of code a half dozen times, you should avoid repeating the same data in your database.

Design your schema to avoid data duplication.

students			
id	**first_name**	**last_name**	**on_probation**
1	Tianna	Lowe	false
2	Elda	Snipes	true

With this duplication, what are the downsides we can expect if a student changes their name?

			addresses				
id	**first_name**	**last_name**	**line_1**	**line_2**	**city**	**state**	**zipcode**
4	Tianna	Lowe	381 Maple St.		Chicago	IL	60657
5	Elda	Snipes	8917 E. Rogers Blvd.	#2E	Joliet	IL	60403

duplication

students			
id	**first_name**	**last_name**	**on_probation**
1	Tianna	Lowe	false
2	Elda	Snipes	true

What's an alternative schema to eliminate this duplication?

			addresses				
id	**first_name**	**last_name**	**line_1**	**line_2**	**city**	**state**	**zipcode**
4	Tianna	Lowe	381 Maple St.		Chicago	IL	60657
5	Elda	Snipes	8917 E. Rogers Blvd.	#2E	Joliet	IL	60403

duplication

students				
id	**first_name**	**last_name**	**on_probation**	**address_id**
1	Tianna	Lowe	false	4
2	Elda	Snipes	true	5

addresses					
id	**line_1**	**line_2**	**city**	**state**	**zipcode**
4	381 Maple St.		Chicago	IL	60657
5	8917 E. Rogers Blvd.	#2E	Joliet	IL	60403

Instead refer to another table using the table's primary key (often the `id` column).

Data Integrity

Good schema design improves your data's integrity.

Careful use of database constraints such column uniqueness or NOT NULL will protect you from situations like this:

users		
id	**email**	**password**
2	james@example.com	PEAoS9mzrudaJEDzZRbutQ==
3	james@example.com	2i4G59b9eQs+KPXpHS/PBw==

Which password should we check when James logs in?

Queries

In SQL all types “commands” are lumped together and called queries. Including commands that don’t necessarily query for anything but instead do something.

`CREATE TABLE` is an example of a query.

Most often you’ll be querying a table with the following commands:

`SELECT`	Retrieve rows that match a query
`INSERT`	Insert new rows
`UPDATE`	Change rows that match a query
`DELETE`	Delete rows that match a query

SELECT

By far the most common query, a `SELECT` query allows you to retrieve matching rows from the database.

```
SELECT * FROM students;
```

```
 id | first_name | last_name
----+------------+-----------
  1 | Tianna     | Lowe
  2 | Elda       | Sipes
(2 rows)
```

SELECT

Instead of selecting all the columns from a table (*), you can specify which columns you want returned:

```
SELECT id, first_name FROM students;
```

```
 id | first_name
----+------------
  1 | Tianna
  2 | Elda
(2 rows)
```

SELECT

Often you'll want to retrieve only records that match a condition:

```
SELECT id, first_name FROM students WHERE id = 2;
```

```
 id | first_name
----+------------
  2 | Elda
(1 row)
```

WHERE

There are numerous conditionals supported by the WHERE clause. Here are a few:

`=`	Column name is equal to a value
`!= or <>`	Column is not equal to a value
`>, >=, <, <=`	Column is greater or less than a value
`BETWEEN`	Column is between two values
`IN`	Column is present in a list of values
`LIKE`	Column matches a string with wildcards

Example Conditions

```
SELECT * FROM students WHERE first_name = 'Andre';

SELECT * FROM enrollments WHERE grade <> 'A';

SELECT * FROM classes WHERE credits BETWEEN 3 AND 4;

SELECT * FROM enrollments WHERE grade IN ('A','B');

SELECT * FROM classes WHERE name LIKE 'ART%' AND credits > 3;
```

INSERT

New records are inserted in a table by specifying the table name, the columns and the values you wish to insert:

```
INSERT INTO students (first_name, last_name, birthdate) VALUES
('Michelle', 'Dupont', '1982-02-11');
```

UPDATE

Updates allow you to change the values of a column. The new column value is applied to all rows that match the `WHERE` conditions.

```
UPDATE addresses

SET line_1='200 Church St.' zipcode='60010'

WHERE id = 2;
```

DELETE

Deleting removes matching rows from a table.

```
DELETE FROM addresses WHERE id = 2;
```

UPDATE & DELETE without conditions

Caution: If you fail to include a WHERE condition, you'll affect all rows in a table.

```
# Everyone moves to 200 Church

UPDATE addresses SET line_1='200 Church St.' zipcode='60010';


# Delete all addresses

DELETE FROM addresses;
```