SQL - CRUD

Contents

- CRUD (Create, Read, Update, Delete)
  - Create: Using CREATE TABLE
  - Read: Using SELECT (we already know this!)
  - Update: Using ALTER TABLE
  - Delete: Using DROP TABLE

Creating Tables

We can create a new blank table or create a new one from existing tables using `SELECT`

- Blank tables can be created using the `CREATE TABLE` statement. The syntax is as follows:

```
CREATE TABLE {table_name} (
  {column_name_1} {data_type_1} {column_constraint_1},
  {column_name_2} {data_type_2} {column_constraint_2},
  …
  {column_name_last} {data_type_last} {column_constraint_last},
  {additional_contraints}
);
```

Creating Tables

- `{table_name}` is the name of the table
- `{column_name}` is the name of the column
- `{data_type}` is the data type of the column:
  - `INT:` −2,147,483,648 to 2,147,483,647
  - `text:` Holds a string with a maximum length of 65,535 bytes
  - `CHAR(size)`: A fixed length string
  - `VARCHAR(size)`: A fixed length string
  - `DATE:` YYYY-MM-DD
  - To see more data types, go to this link

Creating Tables - Constraints

- `{column_constraint}` one or more optional keywords giving special properties to the column:
  - `CHECK`: Ensure that a boolean condition is met
  - `NOT_NULL`: Ensure that no row has a NULL value
  - `UNIQUE`: Ensure that all values are different
  - `PRIMARY KEY`: Combination of `NOT_NULL` and `UNIQUE`. Helps the RDBSM finding the key quicker
  - For more constraints, visit this link

Remember that the table starts blank, and then we populate data. The added data will be limited by the data type constraints we set now.

Constraints - CHECK

- `CHECK:` Ensures that a boolean condition is met

```
CREATE TABLE movies (
   title VARCHAR(30),
   release_date DATE,
   minutes INT CHECK (minutes > 0)
  );
```

Constraints - CHECK

- `CONSTRAINT`: You can give a name to a condition, so error messages are descriptive

```
CREATE TABLE movies (
  title VARCHAR(30),
  release_date DATE,
  minutes INT CONSTRAINT positive CHECK (minutes > 0)
);
```

Constraints - CHECK

- We can also add constraints between columns

```
CREATE TABLE movies (
   title VARCHAR(30),
   release_date DATE,
   premier_access DATE,
   minutes INT CONSTRAINT positive CHECK (minutes > 0)
   CONSTRAINT valid_premier CHECK (release_date > premier_access)
);
```

Constraints - NOT NULL

- `NOT_NULL`: Ensures that no row has a NULL value

```
CREATE TABLE movies (
   title VARCHAR(30) NOT NULL,
   release_date DATE NOT NULL,
   premier_access DATE,
   minutes INT NOT NULL CHECK (minutes > 0)
   CONSTRAINT valid_premier CHECK (release_date > premier_access)
);
```

Constraints - UNIQUE

- `UNIQUE`: Ensure that all values are different

```
CREATE TABLE movies (
   title VARCHAR(30) UNIQUE NOT NULL,
   release_date DATE NOT NULL,
   premier_access DATE,
   minutes INT CONSTRAINT positive CHECK (minutes > 0)
   CONSTRAINT valid_premier CHECK (release_date > premier_access)
);
```

Constraints - PRIMARY KEY

- `PRIMARY KEY`: a column, or group of columns, can be used as a unique identifier for rows in the table. This requires that the values be both unique and not null

```
CREATE TABLE movies (
   title VARCHAR(30) PRIMARY KEY,
   release_date DATE NOT NULL,
   premier_access DATE,
   minutes INT CONSTRAINT positive CHECK (minutes > 0),
   CONSTRAINT valid_premier CHECK (release_date >
  premier_access)
);
```

Creating Tables from SELECT

- When creating a table, we could use an existing one by making a `SELECT` query on it. The syntax is:

```
CREATE TABLE {table_name} AS (
  {SELECT_QUERY}
);

CREATE TABLE actor_short AS (
  SELECT * FROM actor
  LIMIT 10
);
```

Updating Tables

- When updating a table, you can add or drop columns, add rows, or change data points.
- Manipulating columns is done using the `ALTER TABLE` statement. The syntax is as follows:

```
ALTER TABLE {table_name}
  ADD COLUMN {column name} {data_type} {constraint};

ALTER TABLE {table_name}
  DROP COLUMN {column name};
```

Adding Rows

- We can add rows using the `INSERT INTO` statement. The syntax is:

```
INSERT INTO {table_name} ({column_1}, {column_2}, …)
VALUES ({column_1}, {column_2}, …);

INSERT INTO {table_name} ({column_1}, {column_2}, …)
{SELECT QUERY};

INSERT INTO actor_short
SELECT * FROM actor
WHERE first_name LIKE '%EN';
```

Updating Rows

- We can update rows using the `UPDATE` statement. We need to tell the query what column to `SET` given a `WHERE` condition:

```
UPDATE {table_name}
SET {column_1} = {column_value_1}
    {column_2} = {column_value_2}
    …
WHERE {condition}

UPDATE actor_short
SET last_name = ‘AFFLECK’
WHERE first_name = ‘BEN’
```

Updating Rows

- You can use the same statement to delete values in a row

```
UPDATE {table_name}
SET {column_1} = NULL
    {column_2} = NULL
    …
WHERE {condition}
```

Deleting Rows

- You can delete rows, the content of a table, or a whole table
- The syntax for deleting a row is:

```
DELETE FROM {table_name}
WHERE {conditional};

DELETE FROM actor_short
WHERE actor_id = 83;
```

Deleting

- The syntax for deleting the content of the table is:

```
DELETE FROM {table name}

DELETE FROM actor_short
```

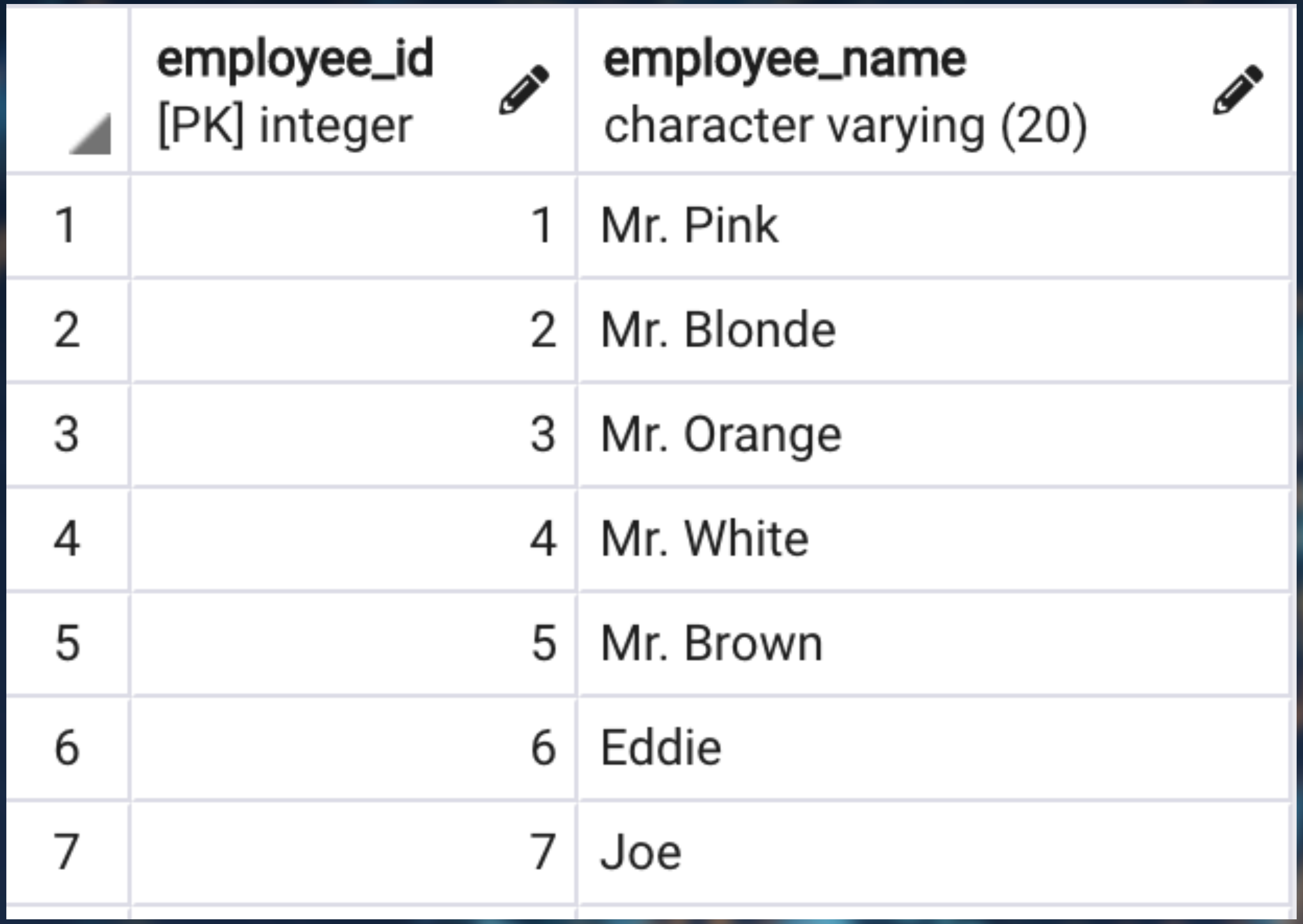
- The syntax for deleting the whole table is:

```
DROP TABLE {table name}

DROP TABLE actor_short
```

Practical

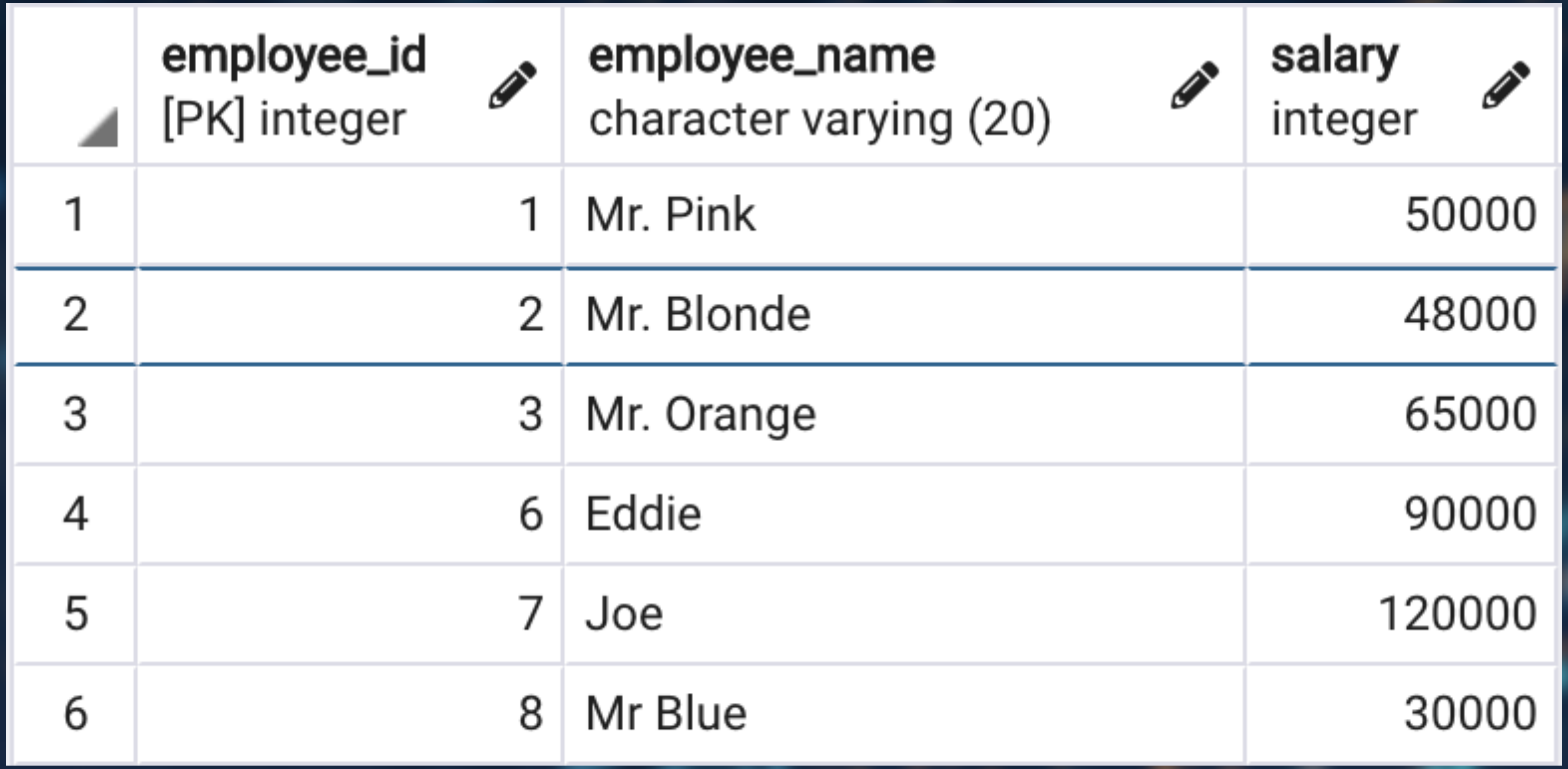
- Go to pgAdmin4 and create a new Database
- Create a table named employee_details with the following values
- Take into account the constraints



	employee_id [PK] integer	employee_name character varying (20)
1	1	Mr. Pink
2	2	Mr. Blonde
3	3	Mr. Orange
4	4	Mr. White
5	5	Mr. Brown
6	6	Eddie
7	7	Joe

Practical

- Create a table named employee_salary with the following values
- Take into account the constraints



	employee_id [PK] integer	employee_name character varying (20)	salary integer
1	1	Mr. Pink	50000
2	2	Mr. Blonde	48000
3	3	Mr. Orange	65000
4	6	Eddie	90000
5	7	Joe	120000
6	8	Mr Blue	30000

Don't delete the database! We will use it in the next lesson