365√DataScience

SQL Basics

TABLE OF CONTENTS

LET’S BREAK THINGS DOWN

TABLE OF CONTENTS

LET’S BREAK THINGS DOWN

TABLE OF CONTENTS

LET'S BREAK THINGS DOWN

SQL Theory

365√DataScience

DATA DEFINITION LANGUAGE (DDL)

Data Definition Language

SQL's syntax

comprises several types of statements that allow you to perform various commands and operations

Data Definition Language (DDL)

- a syntax
- a set of statements that allow the user to define or modify data structures and objects, such as tables

the CREATE statement

used for creating entire databases and database objects as tables

Data Definition Language

the CREATE statement

used for creating entire databases and database objects as tables

SQL

```
CREATE object_type object_name;
```

Data Definition Language

the CREATE statement

used for creating entire databases and database objects as tables

SQL

```
CREATE object_type object_name;

CREATE TABLE object_name (column_name data_type);
```

Data Definition Language

SQL

```
CREATE TABLE object_name (column_name data_type);
```

Data Definition Language

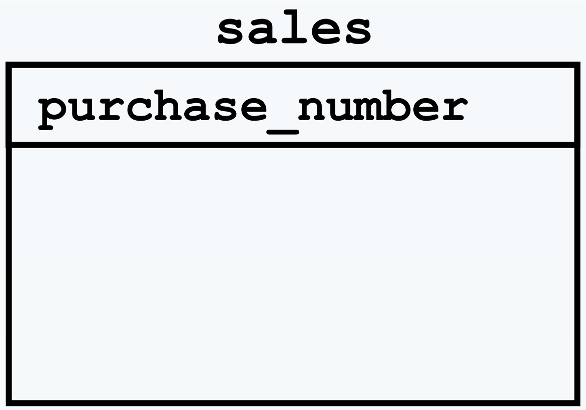
SQL

```
CREATE TABLE object_name (column_name data_type);

CREATE TABLE sales (purchase_number INT);
```

Data Definition Language

SQL

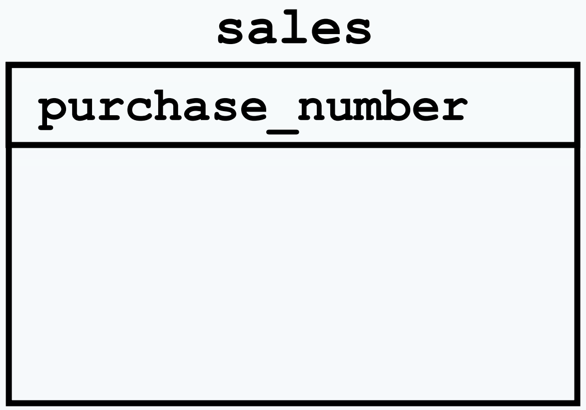
```
CREATE TABLE object_name (column_name data_type);

CREATE TABLE sales (purchase_number INT);
```



Data Definition Language

SQL

```
CREATE TABLE sales (purchase_number INT);
```



the table name can coincide with the name assigned to the database

Data Definition Language

the ALTER statement

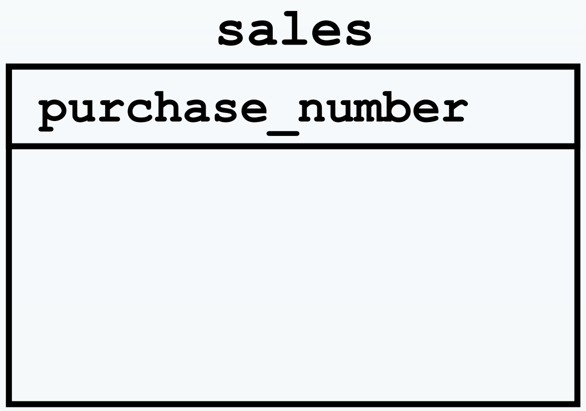
used when altering existing objects

- **ADD**
- **REMOVE**
- **RENAME**

Data Definition Language

SQL

```
ALTER TABLE sales

ADD COLUMN date_of_purchase DATE;
```



365√DataScience

Data Definition Language

SQL

```
ALTER TABLE sales

ADD COLUMN date_of_purchase DATE;
```

sales

purchase_number	date_of_purchase

Data Definition Language

<u>**the DROP statement**</u>

used for deleting a database object

Data Definition Language

SQL

```
DROP object_type object_name;
```

customers

customer_id	first_name

Data Definition Language

used for deleting a database object

SQL

```
DROP object_type object_name;
```

```
DROP TABLE customers;
```

customers

customer_id	first_name

Data Definition Language

used for deleting a database object

SQL

```
DROP object_type object_name;

DROP TABLE customers;
```

customers

customer_id	first_name

Data Definition Language

the RENAME statement

allows you to rename an object

Data Definition Language

SQL

```
RENAME object_type object_name TO new_object_name;
```

customers

customer_id	first_name

365√DataScience

Data Definition Language

used for deleting a database object

SQL

```
RENAME object_type object_name TO new_object_name;
```

```
RENAME TABLE customers TO customer_data;
```

customers

customer_id	first_name

Data Definition Language

used for deleting a database object

SQL

```
RENAME object_type object_name TO new_object_name;
```

```
RENAME TABLE customers TO customer_data;
```

customer_id	first_name

Data Definition Language

used for deleting a database object

SQL

```
RENAME object_type object_name TO new_object_name;
```

```
RENAME TABLE customers TO customer_data;
```

customer_data

customer_id	first_name

Data Definition Language

the TRUNCATE statement

instead of deleting an entire table through DROP, we can also remove its data and continue to have the table as an object in the database

Data Definition Language

SQL

```
TRUNCATE object_type object_name;
```

customers

customer_id	first_name

Data Definition Language

used for deleting a database object

SQL

```
TRUNCATE object_type object_name;
```

```
TRUNCATE TABLE customers;
```

customers

customer_id	first_name

Data Definition Language

used for deleting a database object

SQL

```sql
TRUNCATE object_type object_name;
```

```sql
TRUNCATE TABLE customers;
```

customers

customer_id	first_name

Data Definition Language

Data Definition Language (DDL)

- CREATE
- ALTER
- DROP
- RENAME
- TRUNCATE

SQL Keywords

365 DataScience

Keywords

<u>Keywords:</u>

- ADD

- CREATE

- ALTER

- etc.

<u>KEYWORDS IN SQL CANNOT BE VARIABLE NAMES!</u>

objects or databases cannot have names that coincide with SQL keywords

Keywords

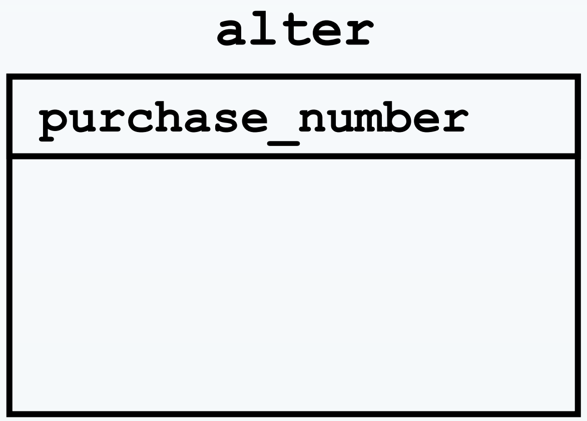
CREATE, ALTER:

Keywords

CREATE, ALTER:

SQL

```
CREATE TABLE alter (purchase_number INT);
```



Data Definition Language

ADD

Data Definition Language

ADD

SQL

```
ALTER TABLE sales

ADD COLUMN date_of_purchase DATE;
```

sales

purchase_number	date_of_purchase

Data Definition Language

ADD, ALTER

SQL

```
ALTER TABLE sales

ADD COLUMN date_of_purchase DATE;
```

sales

purchase_number	date_of_purchase

Keywords

<u>**Keywords = reserved words**</u>

they cannot be used when naming objects

DATA MANIPULATION LANGUAGE (DML)

Data Manipulation Language

<u>Data Manipulation Language (DML)</u>

its statements allow us to manipulate the data in the tables of a database

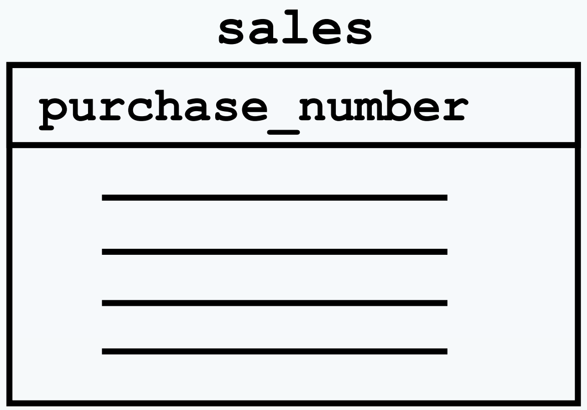
<u>the SELECT statement</u>

used to retrieve data from database objects, like tables

Data Manipulation Language

SQL

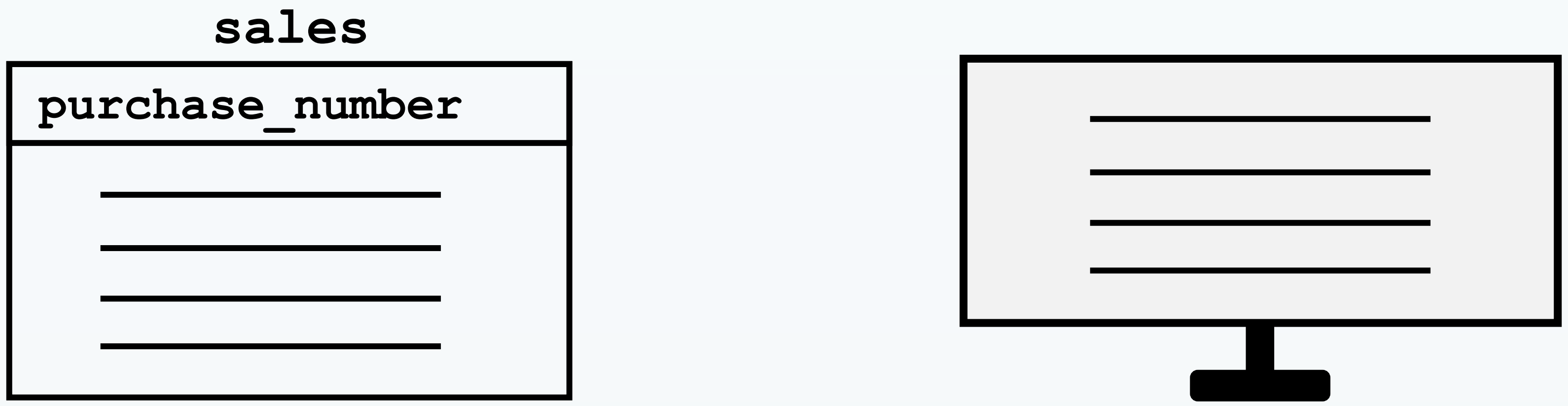
```
SELECT * FROM sales;
```



Data Manipulation Language

SQL

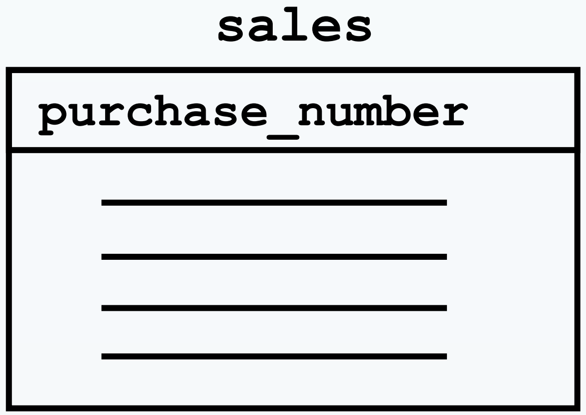
```
SELECT * FROM sales;
```



Data Manipulation Language

SQL

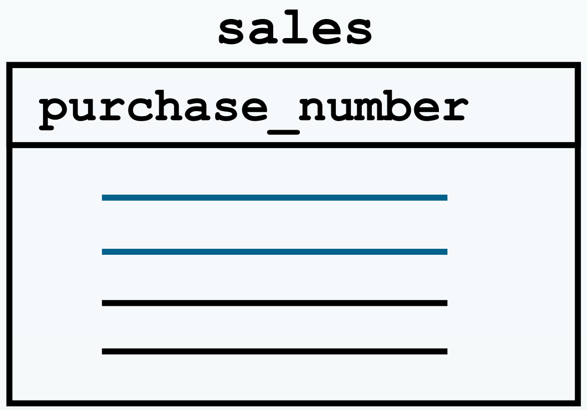
```
SELECT… FROM sales;
```



Data Manipulation Language

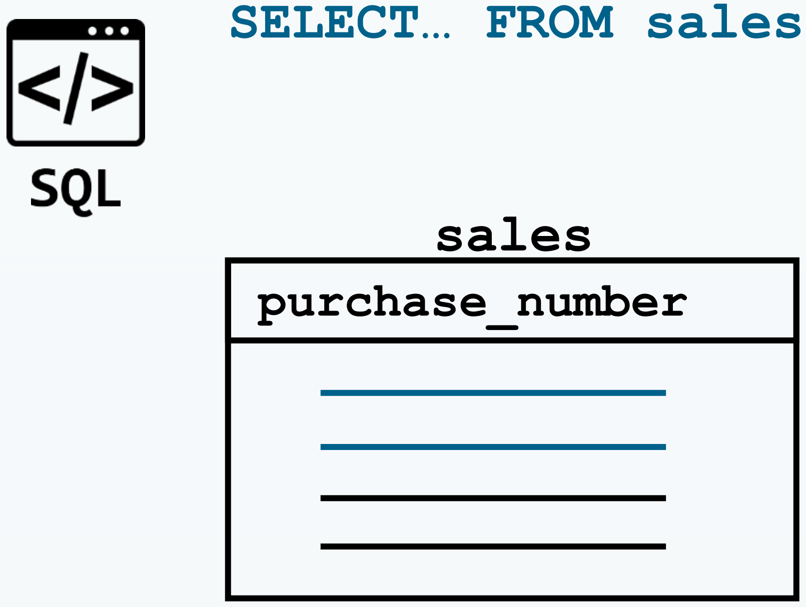
SQL

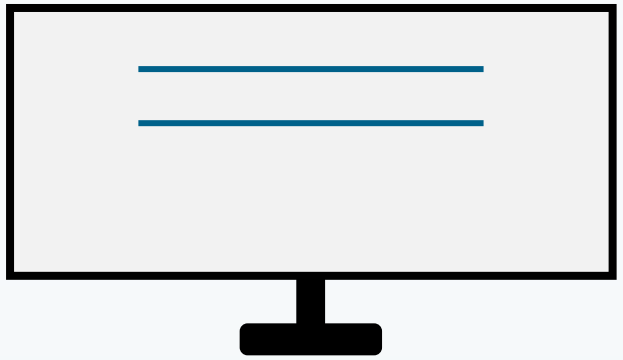
```
SELECT… FROM sales;
```



Data Manipulation Language

```
SELECT… FROM sales;
```





Data Manipulation Language

Why are we going to need just a piece of the table?

- **imagine a table with 2 million rows of data**
- **it can be helpful if you could extract only a portion of the table that satisfies given criteria**

- **you should know how to use SELECT perfectly well**

Data Manipulation Language

<u>the INSERT statement</u>

used to insert data into tables

<u>INSERT INTO… VALUES…;</u>

Data Manipulation Language

SQL

```
INSERT INTO sales (purchase_number, date_of_purchase) VALUES
(1, ‘2017-10-11’);
```

sales

purchase_number	date_of_purchase

365√DataScience

Data Manipulation Language

SQL

```
INSERT INTO sales (purchase_number, date_of_purchase) VALUES
(1, ‘2017-10-11’);
```

sales

purchase_number	date_of_purchase
1	2017-10-11

Data Manipulation Language

SQL

```
INSERT INTO sales VALUES
(1, ‘2017-10-11’);
```

sales

purchase_number	date_of_purchase
1	2017-10-11

Data Manipulation Language

SQL

```
INSERT INTO sales (purchase_number, date_of_purchase) VALUES
(1, ‘2017-10-11’);


INSERT INTO sales VALUES
(1, ‘2017-10-11’);
```

Data Manipulation Language

SQL

```
INSERT INTO sales (purchase_number, date_of_purchase) VALUES
(2, ‘2017-10-27’);
```

sales

purchase_number	date_of_purchase
1	2017-10-11
2	2017-10-27

Data Manipulation Language

<u>the UPDATE statement</u>

allows you to renew existing data of your tables

Data Manipulation Language

SQL

sales

purchase_number	date_of_purchase
1	2017-10-11
2	2017-10-27

Data Manipulation Language

SQL

```
UPDATE sales
SET date_of_purchase = '2017-12-12'
WHERE purchase_number = 1;
```

sales

purchase_number	date_of_purchase
1	2017-10-11
2	2017-10-27

Data Manipulation Language

SQL

```
UPDATE sales
SET date_of_purchase = '2017-12-12'
WHERE purchase_number = 1;
```

sales

purchase_number	date_of_purchase
1	2017-12-12
2	2017-10-27

Data Manipulation Language

<u>the DELETE statement</u>

- functions similarly to the TRUNCATE statement

<u>TRUNCATE vs. DELETE</u>

TRUNCATE allows us to remove all the records contained in a table

vs.

with DELETE, you can specify precisely what you would like to be removed

Data Manipulation Language

SQL

```
DELETE FROM sales;
```

sales

purchase_number	date_of_purchase
1	2017-10-11
2	2017-10-27

Data Manipulation Language

SQL

```
DELETE FROM sales;
```

```
TRUNCATE TABLE
sales;
```

sales

purchase_number	date_of_purchase
1	2017-10-11
2	2017-10-27

Data Manipulation Language

SQL

```
DELETE FROM sales;
```

```
TRUNCATE TABLE
sales;
```

sales

purchase_number	date_of_purchase
1	2017-10-11
2	2017-10-27

Data Manipulation Language

SQL

```
DELETE FROM sales
WHERE
    purchase_number = 1;
```

sales

purchase_number	date_of_purchase
1	2017-10-11
2	2017-10-27

Data Manipulation Language

SQL

```
DELETE FROM sales
WHERE
    purchase_number = 1;
```

sales

purchase_number	date_of_purchase
1	2017-10-11
2	2017-10-27

Data Manipulation Language

Data Manipulation Language (DML)

- SELECT… FROM…
- INSERT INTO… VALUES…
- UPDATE… SET… WHERE…
- DELETE FROM… WHERE…

DATA CONTROL LANGUAGE (DCL)

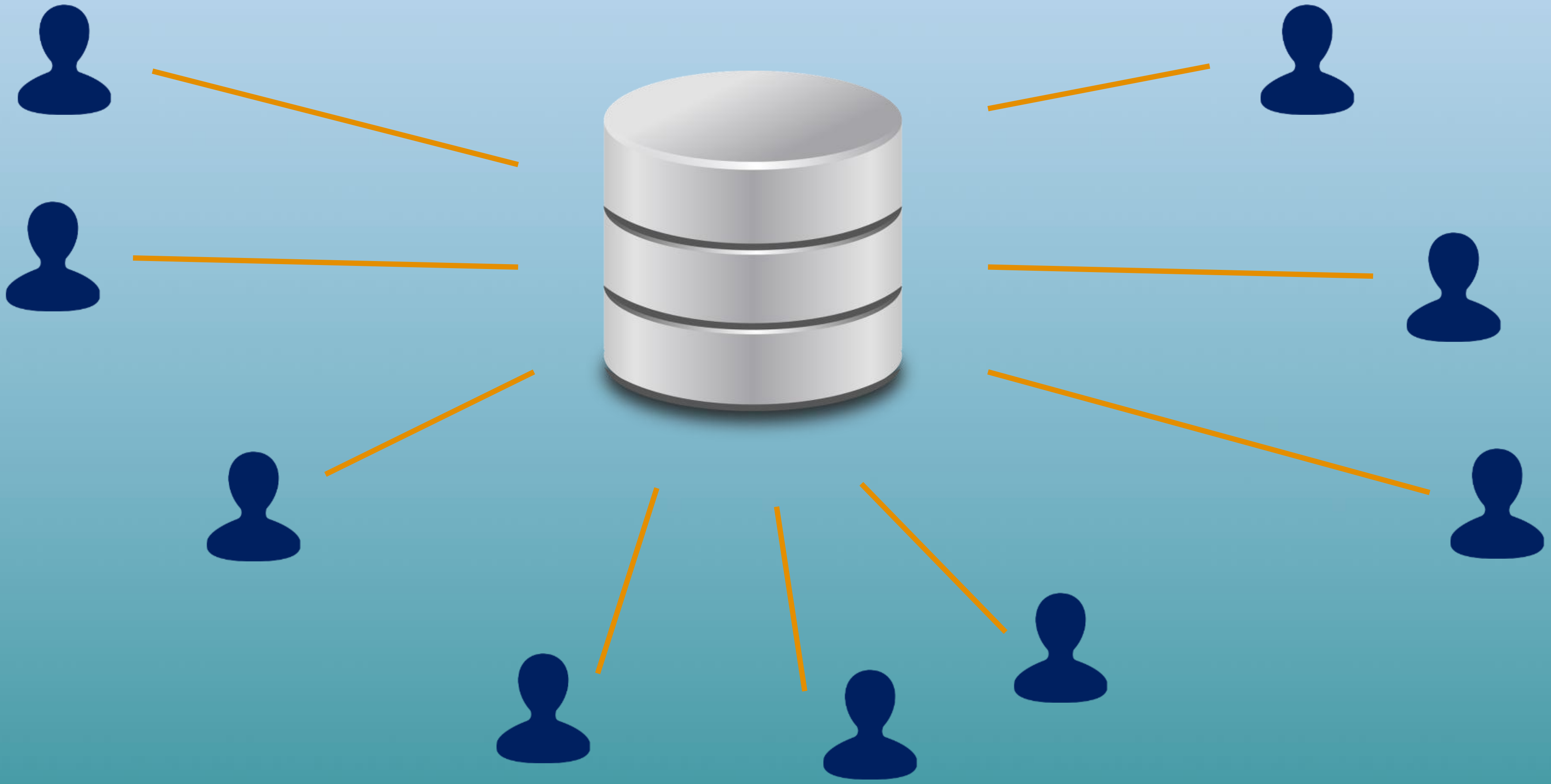
Data Control Language

<u>Data Control Language (DCL)</u>

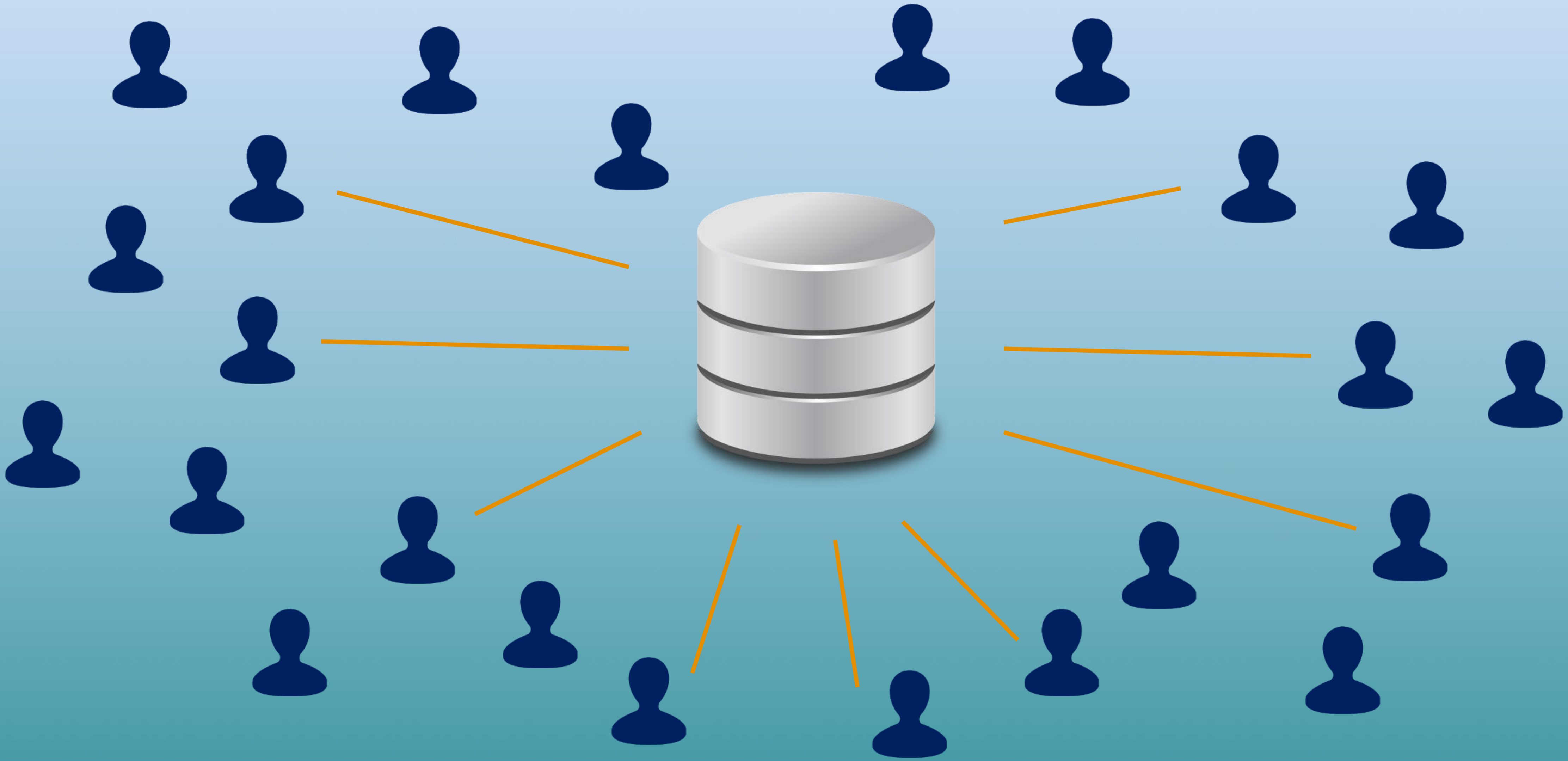
<u>the GRANT and REVOKE statements</u>

allow us to manage the rights users have in a database

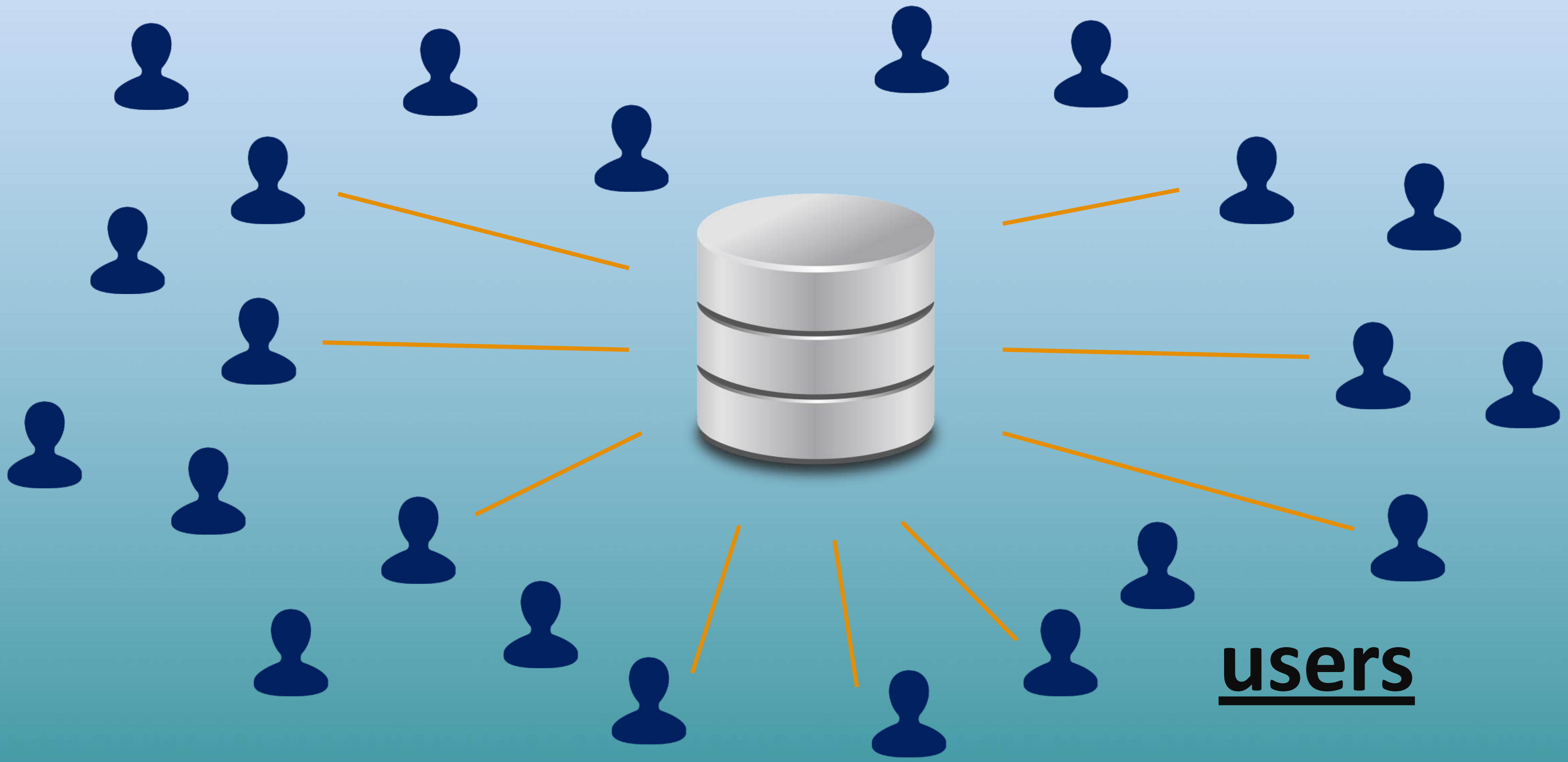
Data Control Language



Data Control Language



Data Control Language



Data Control Language

<u>The GRANT statement</u>

gives (or grants) certain permissions to users

Data Control Language

<u>**The GRANT statement**</u>

gives (or grants) certain permissions to users

SQL

Data Control Language

The GRANT statement

gives (or grants) certain permissions to users

SQL

```
GRANT type_of_permission ON database_name.table_name TO
‘username’@’localhost’
```

Data Control Language

The GRANT statement

gives (or grants) certain permissions to users

one can grant a *specific* type of permission, like *complete* or *partial access*

SQL

```
GRANT type_of_permission ON database_name.table_name TO
‘username’@’localhost’
```

Data Control Language

these rights will be assigned to a person who has a *username* registered at the *local server* (*'localhost': IP 127.0.0.1*)

big companies and corporations don't use this type of server, and their databases lay on *external*, more powerful servers

SQL

```
GRANT type_of_permission ON database_name.table_name TO
'username'@'localhost'
```

Data Control Language

Database administrators

people who have *complete* rights to a database

- they can grant access to users and can revoke it

the REVOKE clause

used to revoke permissions and privileges of database users

- the exact opposite of GRANT

Data Control Language

the REVOKE clause

used to revoke permissions and privileges of database users

SQL

365√DataScience

Data Control Language

the REVOKE clause

used to revoke permissions and privileges of database users

SQL

```
REVOKE type_of_permission ON database_name.table_name FROM
'username'@'localhost'
```

365√DataScience

TRANSACTION CONTROL LANGUAGE (TCL)

Transaction Control Language

Transaction Control Language (TCL)

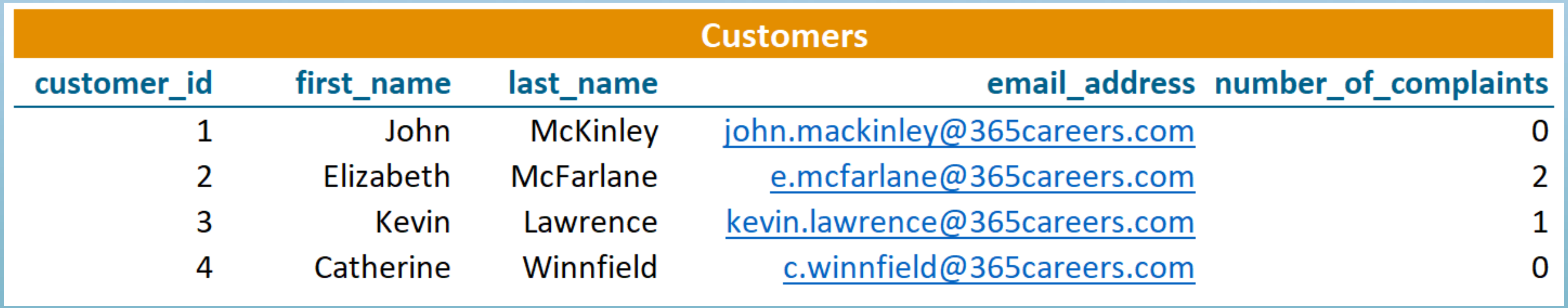
- not every change you make to a database is saved automatically

the COMMIT statement

- related to INSERT, DELETE, UPDATE
- will save the changes you’ve made
- will let other users have access to the modified version of the database

Transaction Control Language

<u>DB administrator</u>



Customers

customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Winnfield	c.winnfield@365careers.com	0

Transaction Control Language

DB administrator

- Change the last name of the 4th customer from ‘Winnfield’ to ‘Johnson’

Customers

customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Winnfield	c.winnfield@365careers.com	0

Transaction Control Language

DB administrator

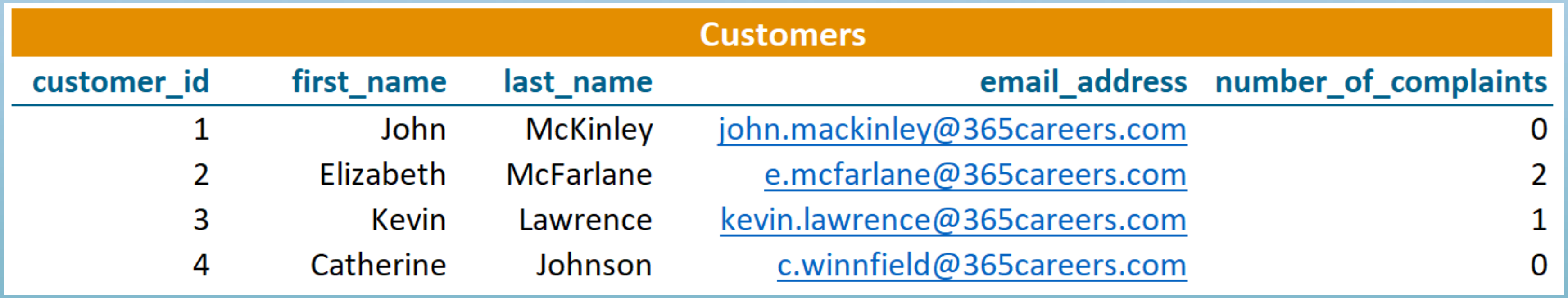
- Change the last name of the 4th customer from ‘Winnfield’ to ‘Johnson’

Customers				
customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine		c.winnfield@365careers.com	0

Transaction Control Language

DB administrator

- Change the last name of the 4th customer from ‘Winnfield’ to ‘Johnson’

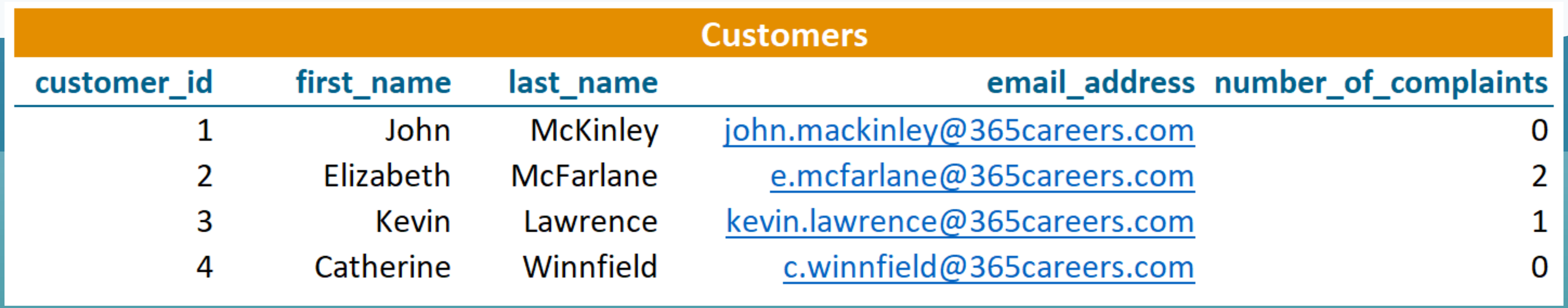


Customers				
customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Johnson	c.winnfield@365careers.com	0

Transaction Control Language

DB administrator

SQL



Customers				
customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Winnfield	c.winnfield@365careers.com	0

Transaction Control Language

<u>DB administrator</u>

SQL

```
UPDATE customers
SET last_name = 'Johnson'
WHERE customer_id = 4;
```

Customers

customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Winnfield	c.winnfield@365careers.com	0

Transaction Control Language

DB administrator

SQL

```
UPDATE customers
SET last_name = 'Johnson'
WHERE customer_id = 4;
```

Customers

customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine		c.winnfield@365careers.com	0

Transaction Control Language

DB administrator

SQL

```
UPDATE customers
SET last_name = 'Johnson'
WHERE customer_id = 4;
```

Customers

customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Johnson	c.winnfield@365careers.com	0

Transaction Control Language

DB administrator

Customers				
customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Johnson	c.winnfield@365careers.com	0

Problem:

users

Customers				
customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Winnfield	c.winnfield@365careers.com	0

Transaction Control Language

DB administrator

SQL

```
UPDATE customers
SET last_name = 'Johnson'
WHERE customer_id = 4;
```

Customers

customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Johnson	c.winnfield@365careers.com	0

Transaction Control Language

DB administrator

SQL

```
UPDATE customers
SET last_name = 'Johnson'
WHERE customer_id = 4
COMMIT;
```

Customers

customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Johnson	c.winnfield@365careers.com	0

Transaction Control Language

DB administrator

Customers				
customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Johnson	c.winnfield@365careers.com	0

users

Customers				
customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Johnson	c.winnfield@365careers.com	0

Transaction Control Language

the COMMIT statement

committed states can accrue

the ROLLBACK clause

the clause that will let you make a step back

- allows you to undo any changes you have made but don’t want to be saved permanently

Transaction Control Language

DB administrator

SQL

```
UPDATE customers
SET last_name = 'Johnson'
WHERE customer_id = 4
COMMIT;
```

Customers

customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Johnson	c.winnfield@365careers.com	0

Transaction Control Language

DB administrator

SQL

```
UPDATE customers
SET last_name = 'Johnson'
WHERE customer_id = 4
COMMIT;

ROLLBACK;
```

Customers

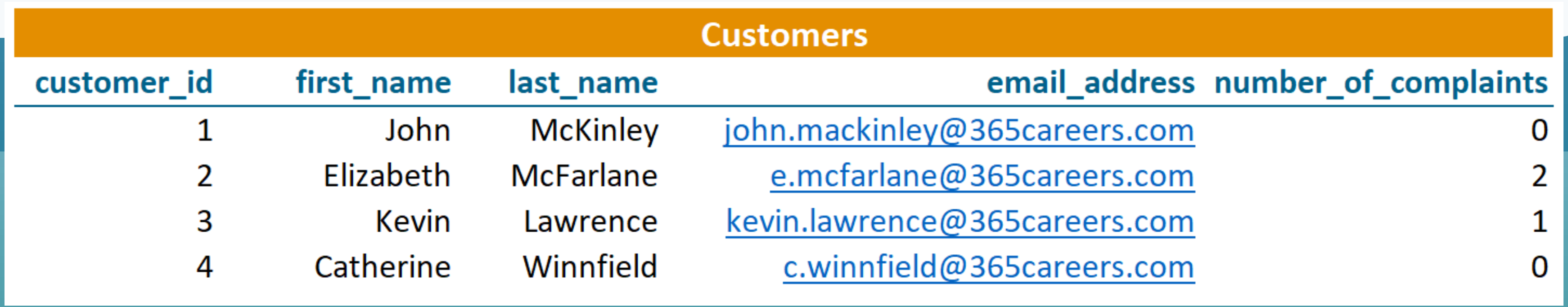
customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Johnson	c.winnfield@365careers.com	0

Transaction Control Language

<u>DB administrator</u>

SQL

```
UPDATE customers
SET last_name = 'Johnson'
WHERE customer_id = 4
COMMIT;

ROLLBACK;
```



Customers

customer_id	first_name	last_name	email_address	number_of_complaints
1	John	McKinley	john.mackinley@365careers.com	0
2	Elizabeth	McFarlane	e.mcfarlane@365careers.com	2
3	Kevin	Lawrence	kevin.lawrence@365careers.com	1
4	Catherine	Winnfield	c.winnfield@365careers.com	0

Transaction Control Language

the COMMIT statement

- saves the transaction in the database
- changes cannot be undone

the ROLLBACK clause

- allows you to take a step back
- the last change(s) made will not count
- reverts to the last non-committed state

SQL Syntax

DDL – Data Definition Language

creation of data

DML – Data Manipulation Language

manipulation of data

DCL – Data Control Language

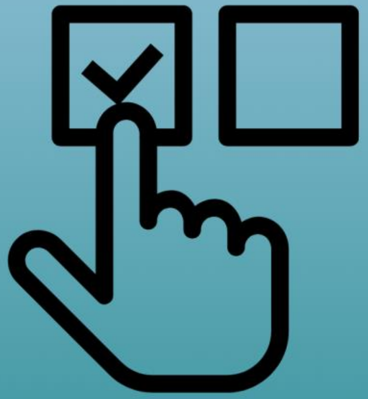
assignment and removal of permissions to use this data

TCL – Transaction Control Language

saving and restoring changes to a database

SQL

SELECT STATEMENT



365√DataScience

SELECT…FROM…

SELECT… FROM…

the <u>SELECT</u> statement

allows you to extract a fraction of the entire data set

- used to retrieve data from database objects, like tables

- used to *“query data from a database”*

SELECT… FROM…

SQL

```
SELECT column_1, column_2,… column_n
FROM table_name;
```

SELECT… FROM…

SQL

```
SELECT column_1, column_2,… column_n
FROM table_name;
```

- when extracting information, SELECT goes with FROM

SELECT… FROM…

SQL

```
SELECT column_1, column_2,… column_n
FROM table_name;
```

SELECT... FROM...

SQL

```
SELECT column_1, column_2,… column_n
FROM table_name;

SELECT first_name, last_name
FROM employees;
```

SELECT... FROM...

SQL

```sql
SELECT * FROM employees;
```

* - a wildcard character, means “all” and “everything”

WHERE

WHERE

SQL

```
SELECT * FROM employees;
```

WHERE

SQL

```
SELECT column_1, column_2,… column_n
FROM table_name;
```

WHERE

the <u>WHERE</u> clause

it will allow us to set a <u>condition</u> upon which we will specify what part of the data we want to retrieve from the database

WHERE

the WHERE clause

it will allow us to set a condition upon which we will specify what part of the data we want to retrieve from the database

SQL

```
SELECT column_1, column_2,… column_n
FROM table_name;
```

365√DataScience

WHERE

the WHERE clause

it will allow us to set a condition upon which we will specify what part of the data we want to retrieve from the database

SQL

```
SELECT column_1, column_2,… column_n
FROM table_name
WHERE condition;
```

AND

AND

= equal operator

in SQL, there are many other *linking keywords and symbols*, called operators, that you can use with the WHERE clause

- AND		- EXISTS	- NOT EXISTS
- OR		- IS NULL	- IS NOT NULL
- IN	- NOT IN	- comparison operators	
- LIKE	- NOT LIKE	- etc.	
- BETWEEN… AND…			

AND

<u>AND</u>

allows you to logically combine two statements in the condition code block

AND

AND

allows you to logically combine two statements in the condition code block

SQL

```
SELECT column_1, column_2,… column_n
FROM table_name
WHERE condition_1 AND condition_2;
```

- allows us to *narrow* the output we would like to extract from our data

OR

OR

AND

AND binds SQL to meet both conditions enlisted in the WHERE clause *simultaneously*

SQL

```
SELECT column_1, column_2,… column_n
FROM table_name
WHERE condition_1 AND condition_2;
```

OR

<u>AND</u>

conditions set on *different* columns

<u>OR</u>

conditions set on *the same* column

OPERATOR PRECEDENCE

Operator Precedence

logical operator precedence

an SQL rule stating that in the execution of the query, the operator AND is applied first, while the operator OR is applied second

AND > OR

regardless of the order in which you use these operators, SQL will always start by reading the conditions around the AND operator

WILDCARD CHARACTERS

Wildcard Characters

<u>wildcard characters</u>

%   _   *

you would need a <u>wildcard character</u> whenever you wished to put *"anything"* on its place

Wildcard Characters

%

- a substitute for a ***sequence*** of characters

LIKE ('Mar%')

Mar<u>k</u>, Mar<u>tin</u>, Mar<u>garet</u>

_

- helps you match a ***single*** character

LIKE ('Mar_')

Mar<u>k</u>, Mar<u>y</u>, Mar<u>l</u>

Wildcard Characters

***** **will deliver a list of *all* columns in a table**

```
SELECT * FROM employees;
```

- it can be used to count *all* rows of a table

BETWEEN…AND…

BETWEEN… AND…

<u>BETWEEN… AND…</u>

helps us designate the interval to which a given value belongs

BETWEEN... AND...

SQL

```
SELECT
    *
FROM
    employees
WHERE
    hire_date BETWEEN '1990-01-01' AND '2000-01-01';
```

365√DataScience

BETWEEN... AND...

SQL

```
SELECT
    *
FROM
    employees
WHERE
    hire_date BETWEEN '1990-01-01' AND '2000-01-01';
```

'1990-01-01' **AND** '2000-01-01' ***will be included* in the retrieved list of records**

BETWEEN… AND…

NOT BETWEEN… AND…

will refer to an interval composed of two parts:

- **an interval below the first value indicated**
- **a second interval above the second value**

BETWEEN... AND...

SQL

```
SELECT
    *
FROM
    employees
WHERE
    hire_date NOT BETWEEN '1990-01-01' AND '2000-01-01';
```

365√DataScience

BETWEEN… AND…

SQL

```
SELECT
    *
FROM
    employees
WHERE
    hire_date NOT BETWEEN '1990-01-01' AND '2000-01-01';
```

- the hire_date is ***before*** '1990-01-01'

or

- the hire_date is ***after*** '2000-01-01'

BETWEEN… AND…

SQL

```
SELECT
    *
FROM
    employees
WHERE
    hire_date NOT BETWEEN '1990-01-01' AND '2000-01-01';
```

'1990-01-01' **AND** '2000-01-01' ***are not included*** **in the intervals**

BETWEEN… AND…

<u>BETWEEN… AND…</u>

- not used only for date values

- could also be applied to strings and numbers

IS NOT NULL / IS NULL

IS NOT NULL / IS NULL

<u>IS NOT NULL</u>

used to extract values that are not null

IS NOT NULL / IS NULL

IS NOT NULL

used to extract values that are not null

SQL

```
SELECT column_1, column_2,… column_n
FROM table_name
WHERE column_name IS NOT NULL;
```

IS NOT NULL / IS NULL

IS NULL

used to extract values that are null

SQL

```
SELECT column_1, column_2,… column_n
FROM table_name
WHERE column_name IS NULL;
```

OTHER COMPARISON OPERATORS

Other Comparison Operators

SQL	
=	equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

Other Comparison Operators

SQL	<u>“Not Equal” operators</u>
<>, !=	not equal,≠ different from

SELECT DISTINCT

SELECT DISTINCT

the SELECT statement

can retrieve rows from a designated column, given some criteria

SELECT DISTINCT

SELECT DISTINCT

selects all *distinct, different* data values

SELECT DISTINCT

SELECT DISTINCT

selects all *distinct, different* data values

SQL

```
SELECT DISTINCT column_1, column_2,… column_n
FROM table_name;
```

INTRODUCTION TO AGGREGATE FUNCTIONS

Introduction to Aggregate Functions

aggregate functions

they are applied on *multiple rows* of *a single column* of a table and *return* an output of *a single value*

Introduction to Aggregate Functions

COUNT()

counts the number of non-null records in a field

SUM()

sums all the non-null values in a column

MIN()

returns the minimum value from the entire list

MAX()

returns the maximum value from the entire list

AVG()

calculates the average of all non-null values belonging to a certain column of a table

Introduction to Aggregate Functions

COUNT()

counts the number of non-null records in a field

- it is frequently used in combination with the reserved word <u>“DISTINCT”</u>

Introduction to Aggregate Functions

COUNT()

SQL

```
SELECT COUNT(column_name)

FROM table_name;
```

the parentheses after COUNT() must start right after the keyword, not after a whitespace

Introduction to Aggregate Functions

COUNT(DISTINCT)

SQL

```
SELECT COUNT(DISTINCT column_name)

FROM table_name;
```

Introduction to Aggregate Functions

aggregate functions

they are applied on *multiple rows* of *a single column* of a table and *return* an output of *a single value*

- they ignore NULL values unless told not to

GROUP BY

GROUP BY

<u>GROUP BY</u>

When working in SQL, results can be grouped according to a specific field or fields

- <u>GROUP BY</u> must be placed immediately after the <u>WHERE</u> conditions, if any, and just before the <u>ORDER BY</u> clause

- <u>GROUP BY</u> is one of the most powerful and useful tools in SQL

GROUP BY

GROUP BY

SQL

```
SELECT column_name(s)
FROM table_name
WHERE conditions
GROUP BY column_name(s)
ORDER BY column_name(s);
```

365√DataScience

GROUP BY

GROUP BY

in most cases, when you need an aggregate function, you must add a GROUP BY clause in your query, too

Always include the field you have grouped your results by in the SELECT statement!

HAVING

HAVING

<u>HAVING</u>

refines the output from records that do not satisfy a certain condition

- frequently implemented with <u>GROUP BY</u>

HAVING

SQL

```
SELECT column_name(s)
FROM table_name
WHERE conditions
GROUP BY column_name(s)
HAVING conditions
ORDER BY column_name(s);
```

- **HAVING** is like **WHERE** but applied to the **GROUP BY** block

HAVING

WHERE vs. HAVING

after HAVING, you can have a condition with an aggregate function, while WHERE cannot use aggregate functions within its conditions

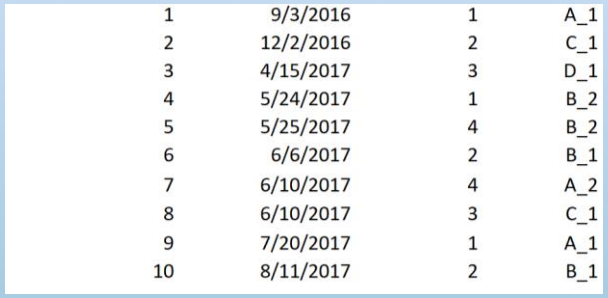
WHERE vs HAVING

WHERE vs HAVING

<u>WHERE</u>

allows us to set conditions that refer to subsets of *individual* rows

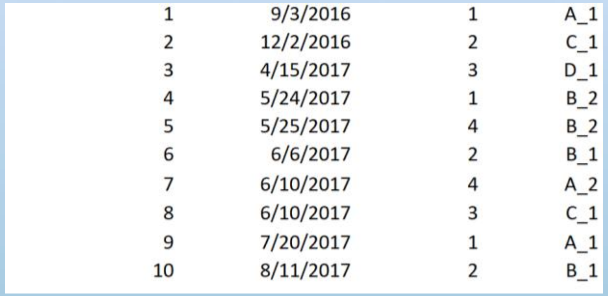
WHERE vs HAVING



1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
5	5/25/2017	4	B_2
6	6/6/2017	2	B_1
7	6/10/2017	4	A_2
8	6/10/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

WHERE vs HAVING

<u>**WHERE**</u>



1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
5	5/25/2017	4	B_2
6	6/6/2017	2	B_1
7	6/10/2017	4	A_2
8	6/10/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

WHERE vs HAVING

WHERE

1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
5	5/25/2017	4	B_2
6	6/6/2017	2	B_1
7	6/10/2017	4	A_2
8	6/10/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

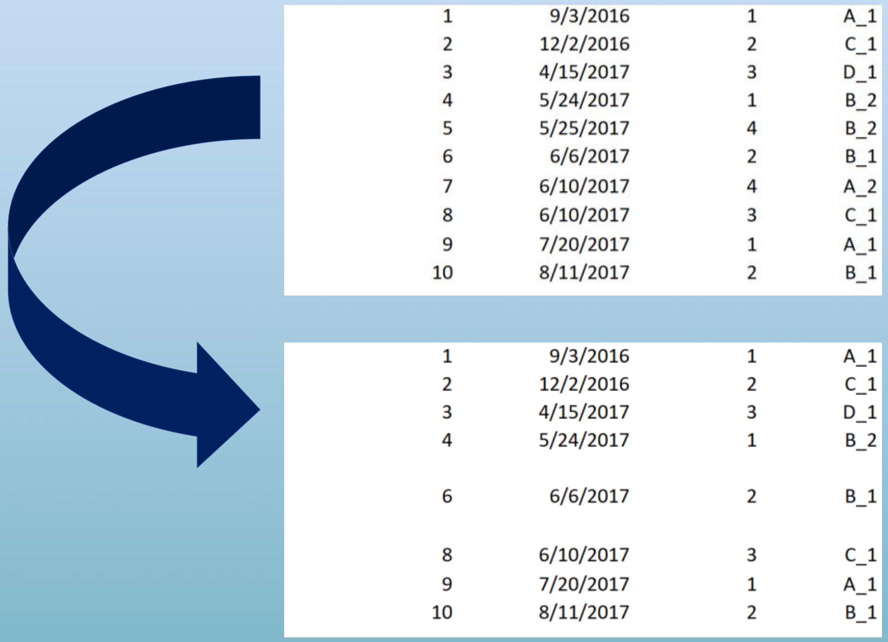
1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
6	6/6/2017	2	B_1
8	6/10/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

re-organizing the output into groups

(GROUP BY)

WHERE vs HAVING

WHERE



1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
5	5/25/2017	4	B_2
6	6/6/2017	2	B_1
7	6/10/2017	4	A_2
8	6/10/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
6	6/6/2017	2	B_1
8	6/10/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

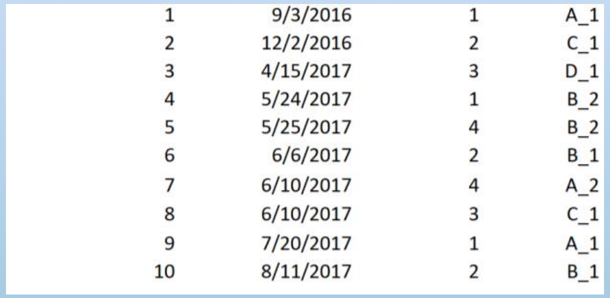
re-organizing the output into groups

(GROUP BY)

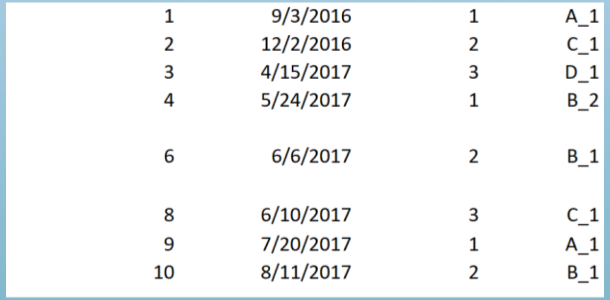
the output can be further improved, or *filtered*

WHERE vs HAVING

WHERE



1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
5	5/25/2017	4	B_2
6	6/6/2017	2	B_1
7	6/10/2017	4	A_2
8	6/10/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1



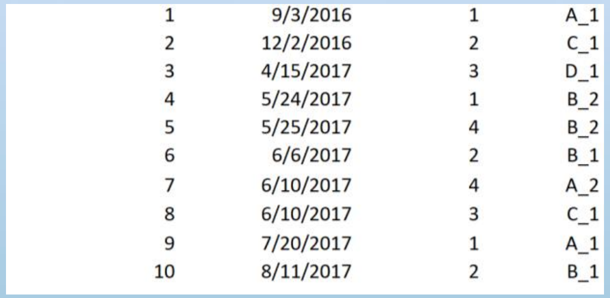
1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
6	6/6/2017	2	B_1
8	6/10/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

re-organizing the output into groups

(GROUP BY)

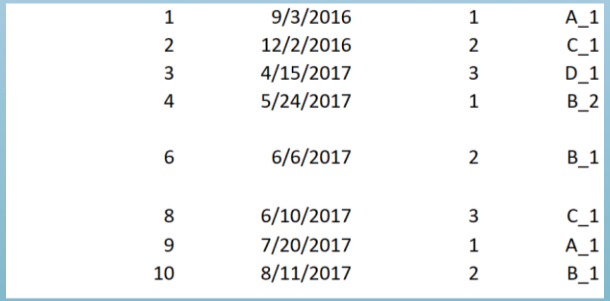
HAVING

WHERE vs HAVING



1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
5	5/25/2017	4	B_2
6	6/6/2017	2	B_1
7	6/10/2017	4	A_2
8	6/10/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

WHERE

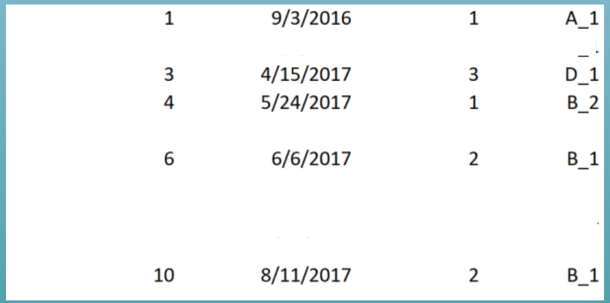


1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
6	6/6/2017	2	B_1
8	6/10/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

re-organizing the output into groups

(GROUP BY)

HAVING



1	9/3/2016	1	A_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
6	6/6/2017	2	B_1
10	8/11/2017	2	B_1

WHERE vs HAVING

WHERE

1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
5	5/25/2017	4	B_2
6	6/6/2017	2	B_1
7	6/10/2017	4	A_2
8	6/10/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

1	9/3/2016	1	A_1
2	12/2/2016	2	C_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
6	6/6/2017	2	B_1
8	6/10/2017	3	C_1
9	7/20/2017	1	A_1
10	8/11/2017	2	B_1

re-organizing the output into groups

(GROUP BY)

HAVING

1	9/3/2016	1	A_1
3	4/15/2017	3	D_1
4	5/24/2017	1	B_2
6	6/6/2017	2	B_1
10	8/11/2017	2	B_1

ORDER BY…

WHERE vs HAVING

<u>HAVING</u>

- you *cannot* have both an aggregated and a non-aggregated condition in the HAVING clause

WHERE vs HAVING

Aggregate functions – <u>GROUP BY</u> and <u>HAVING</u>

General conditions - <u>WHERE</u>

WHERE vs HAVING

SQL

```
SELECT column_name(s)
FROM table_name
WHERE conditions
GROUP BY column_name(s)
HAVING conditions
ORDER BY column_name(s);
```

LIMIT

LIMIT

SQL

```
SELECT column_name(s)
FROM table_name
WHERE conditions
GROUP BY column_name(s)
HAVING conditions
ORDER BY column_name(s)
LIMIT number ;
```

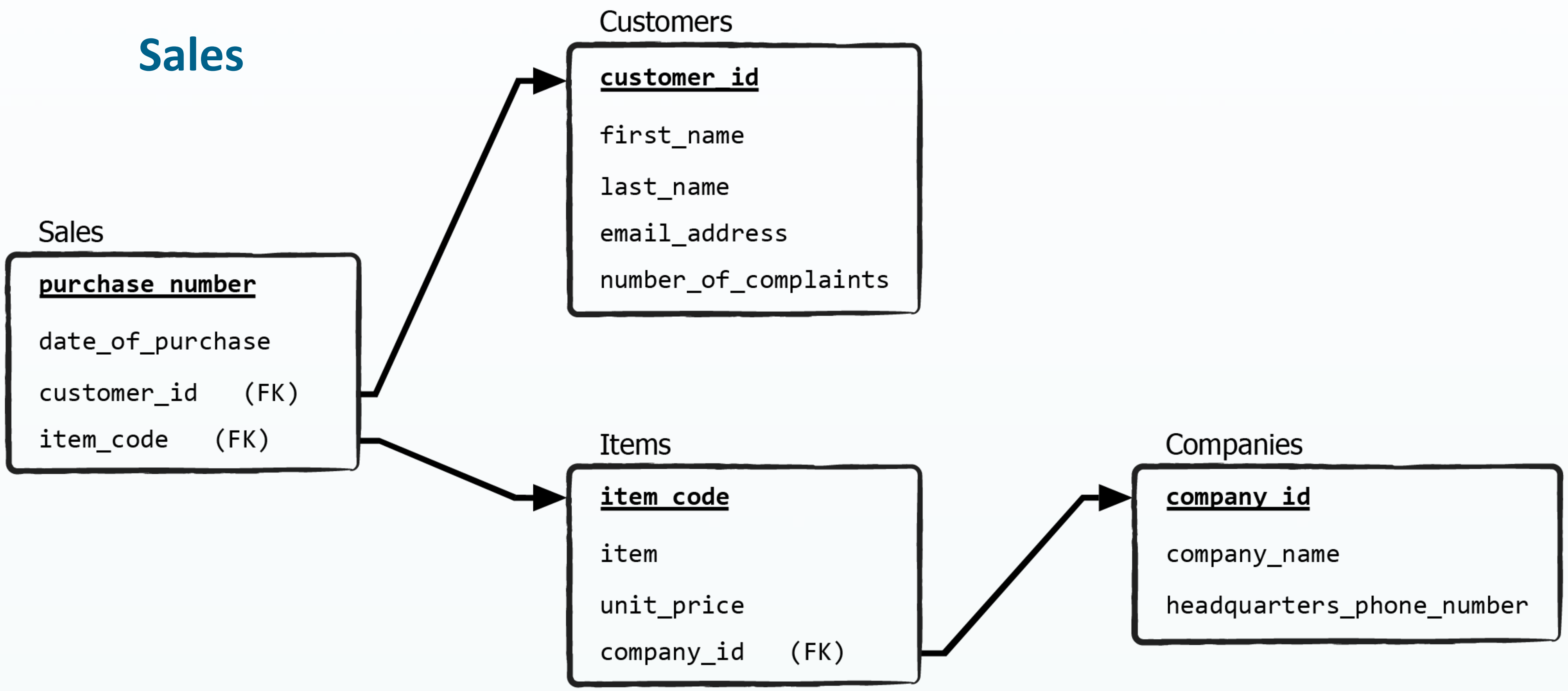
SQL

INSERT STATEMENT

365DataScience

The INSERT Statement

Sales



The INSERT Statement

The INSERT Statement

SQL

```
INSERT INTO table_name (column_1, column_2, …, column_n)
VALUES (value_1, value_2, …, value_n);
```

INSERTING DATA INTO A NEW TABLE

Inserting Data INTO a New Table

INSERT INTO SELECT

SQL

```
INSERT INTO table_2 (column_1, column_2, …, column_n)
SELECT column_1, column_2, …, column_n
FROM table_1
WHERE condition;
```

SQL

UPDATE STATEMENT

365√DataScience

TCL’S COMMIT AND ROLLBACK

TCL’s COMMIT and ROLLBACK

the COMMIT statement

- saves the transaction in the database
- changes cannot be undone

used to save the state of the data in the database at the moment of its execution

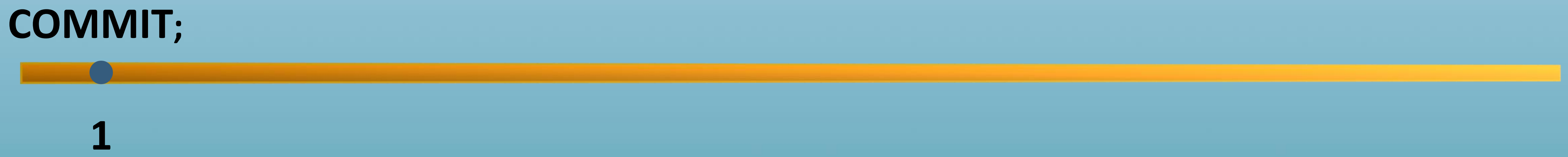
the ROLLBACK clause

- allows you to take a step back
- the last change(s) made will not count
- reverts to the last non-committed state

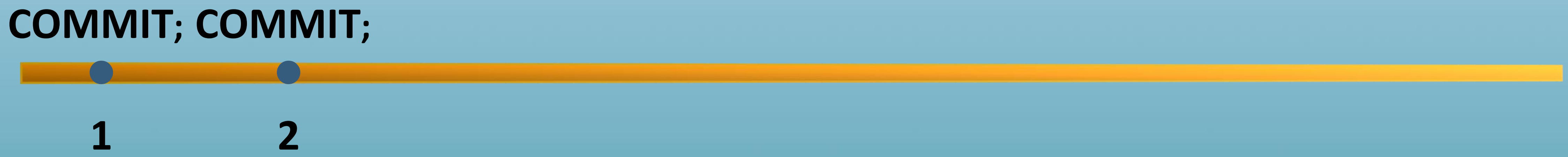
it will refer to the state corresponding to the last time you executed COMMIT

TCL’s COMMIT and ROLLBACK

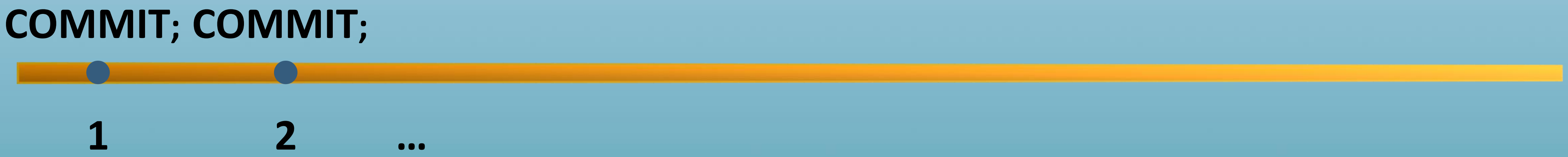
TCL's COMMIT and ROLLBACK



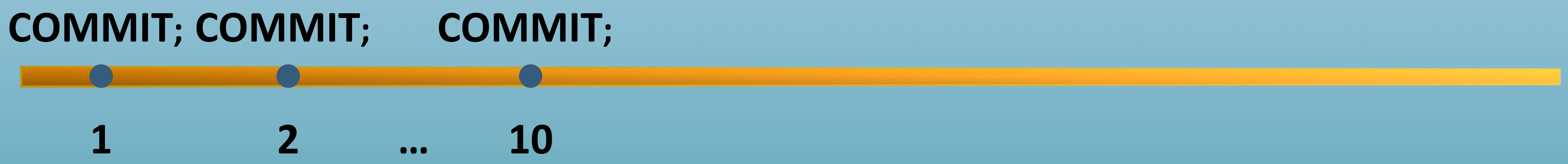
TCL's COMMIT and ROLLBACK



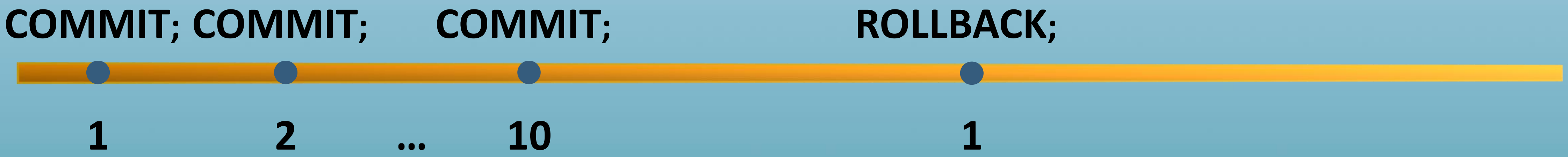
TCL's COMMIT and ROLLBACK



TCL's COMMIT and ROLLBACK

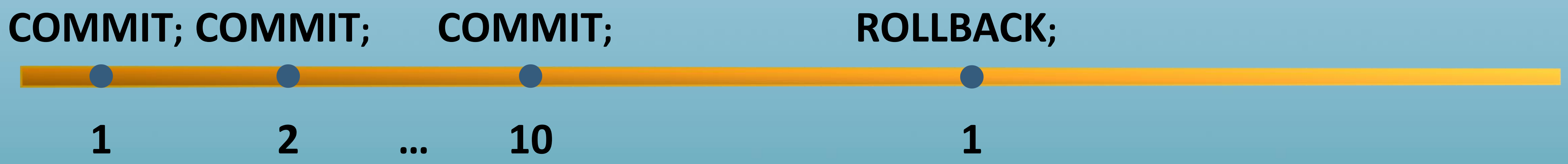


TCL's COMMIT and ROLLBACK



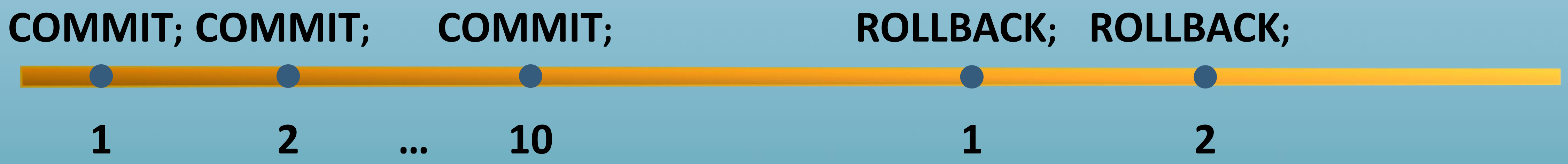
TCL's COMMIT and ROLLBACK

- ROLLBACK will have an effect *on the last execution* you have performed



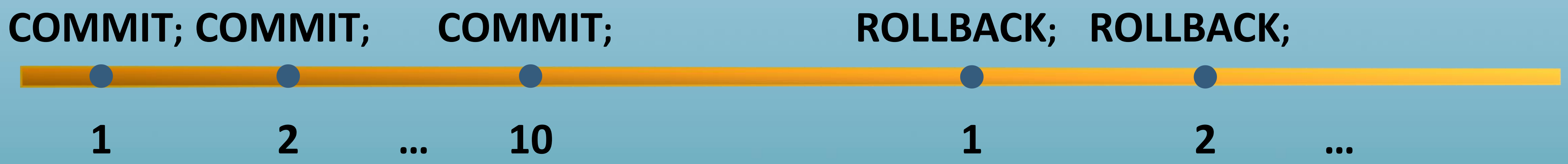
TCL's COMMIT and ROLLBACK

- ROLLBACK will have an effect *on the last execution* you have performed



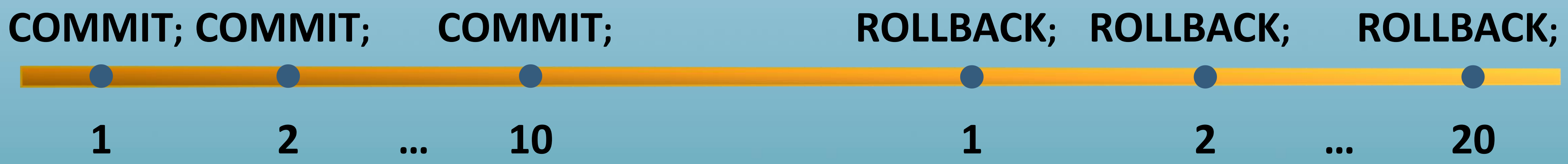
TCL's COMMIT and ROLLBACK

- ROLLBACK will have an effect *on the last execution* you have performed



TCL's COMMIT and ROLLBACK

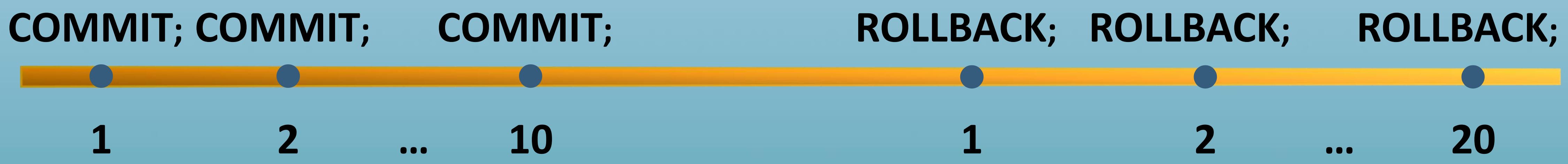
- ROLLBACK will have an effect *on the last execution* you have performed



TCL's COMMIT and ROLLBACK

- ROLLBACK will have an effect *on the last execution* you have performed

- you cannot restore data to a state corresponding to an earlier COMMIT



THE SQL UPDATE STATEMENT

The UPDATE Statement

<u>the UPDATE Statement</u>

used to update the values of existing records in a table

The UPDATE Statement

the UPDATE Statement

used to update the values of existing records in a table

SQL

```
UPDATE table_name

SET column_1 = value_1, column_2 = value_2 …

WHERE conditions;
```

- we do not have to update each value of the record of interest

- we can still say we have updated the specific record

The UPDATE Statement

the UPDATE Statement

used to update the values of existing records in a table

SQL

```
UPDATE table_name
SET column_1 = value_1, column_2 = value_2 …
WHERE conditions;
```

- if you don’t provide a WHERE *condition*, all rows of the table will be updated

SQL

DELETE STATEMENT

365√DataScience

The DELETE Statement

<u>**the DELETE statement**</u>

removes records from a database

SQL

```
DELETE FROM table_name
WHERE conditions;
```

FOREIGN KEY Constraint

<u>ON DELETE CASCADE</u>

if a specific value *from the parent table's primary key* has been deleted, all the records *from the child table* referring to this value will be removed as well

DROP vs TURNCATE vs DELETE

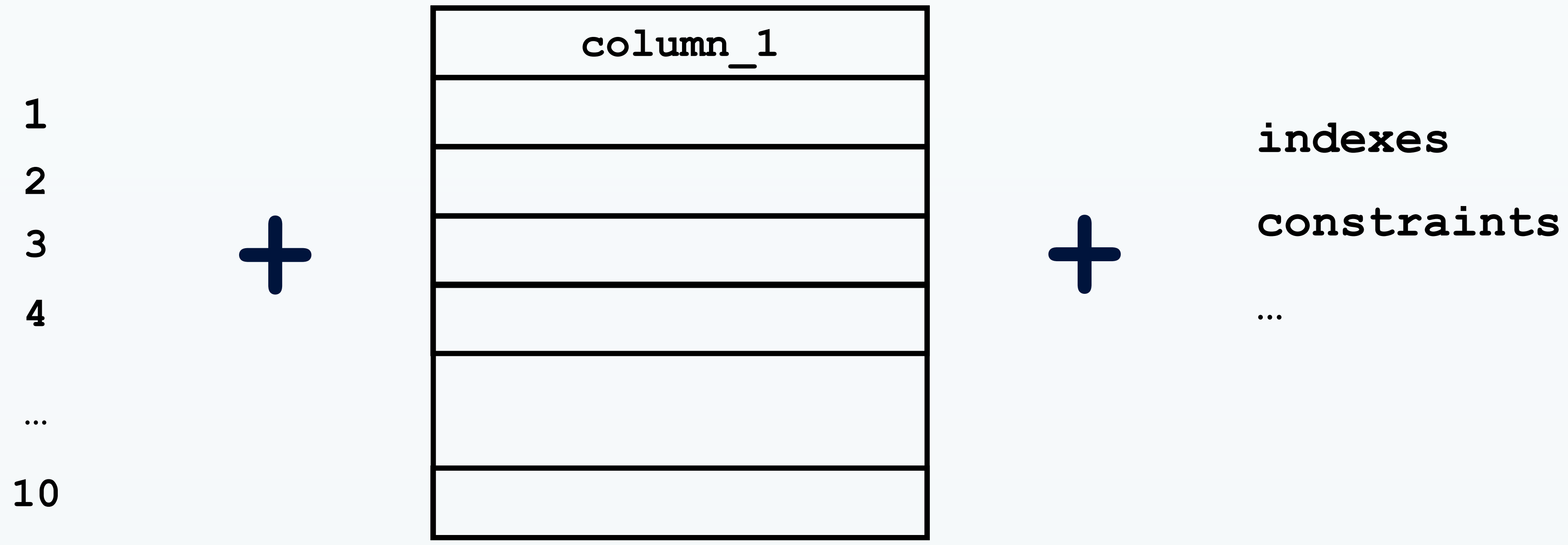
DROP vs TRUNCATE vs DELETE

DROP

column_1
1
2
3
4
…
10

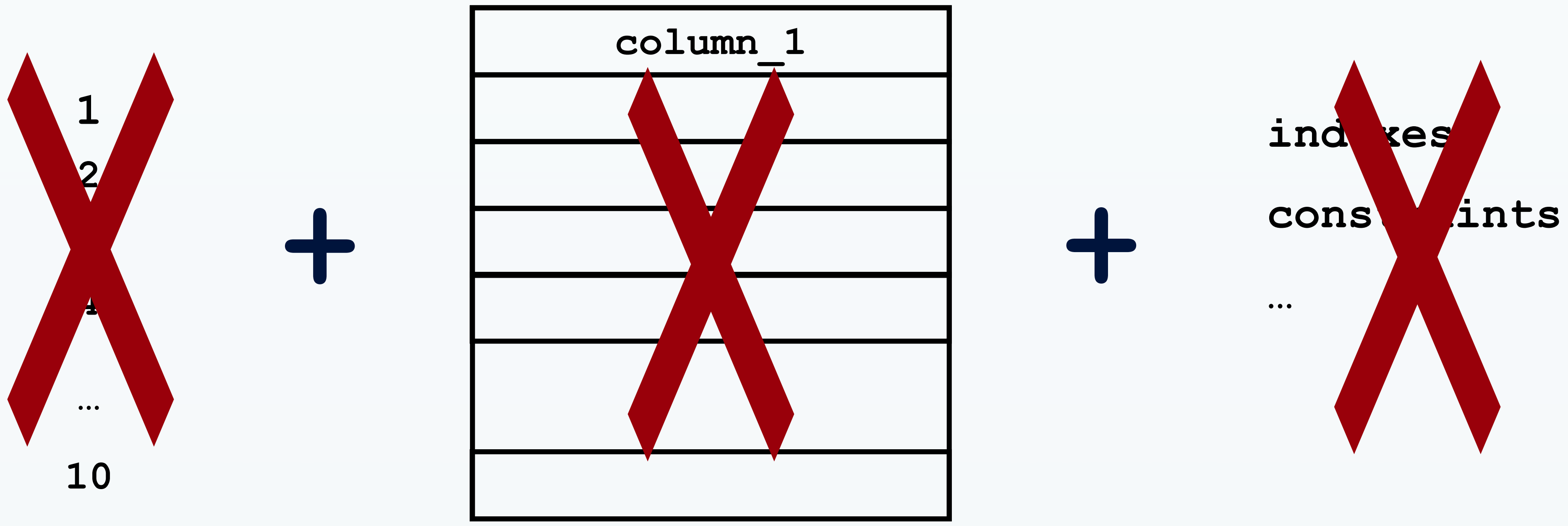
DROP vs TRUNCATE vs DELETE

DROP



DROP vs TRUNCATE vs DELETE

DROP



DROP vs TRUNCATE vs DELETE

DROP

- you won't be able to roll back to its initial state, or to the last COMMIT statement

use DROP TABLE only when you are sure you aren't going to use the table in question anymore

DROP vs TRUNCATE vs DELETE

TRUNCATE

column_1
1
2
3
4
…
10

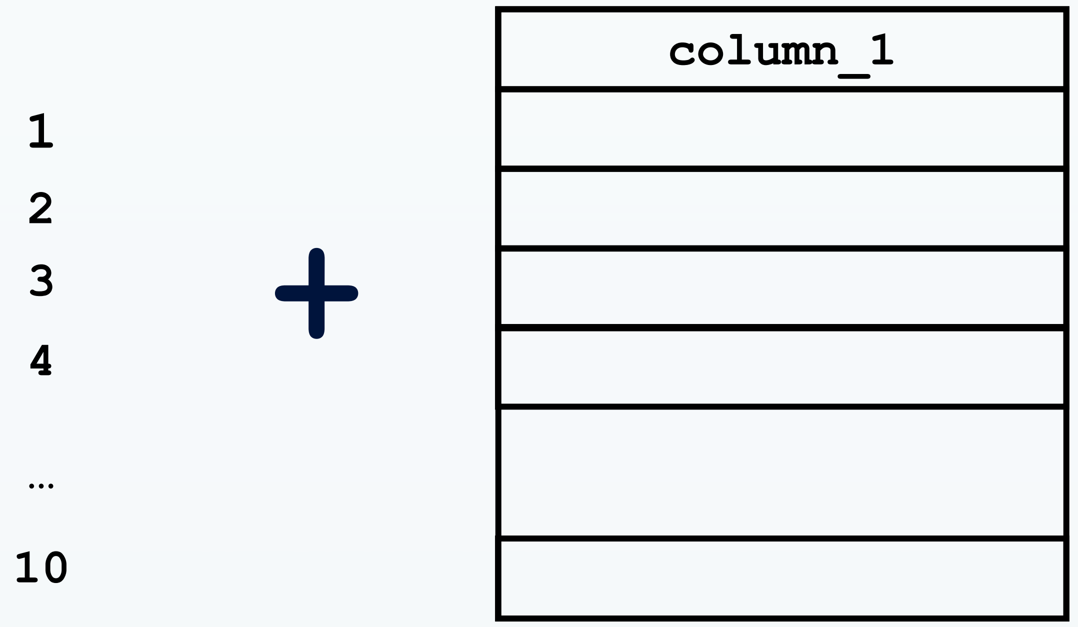
DROP vs TRUNCATE vs DELETE

TRUNCATE ~ DELETE without WHERE

column_1
1
2
3
4
…
10

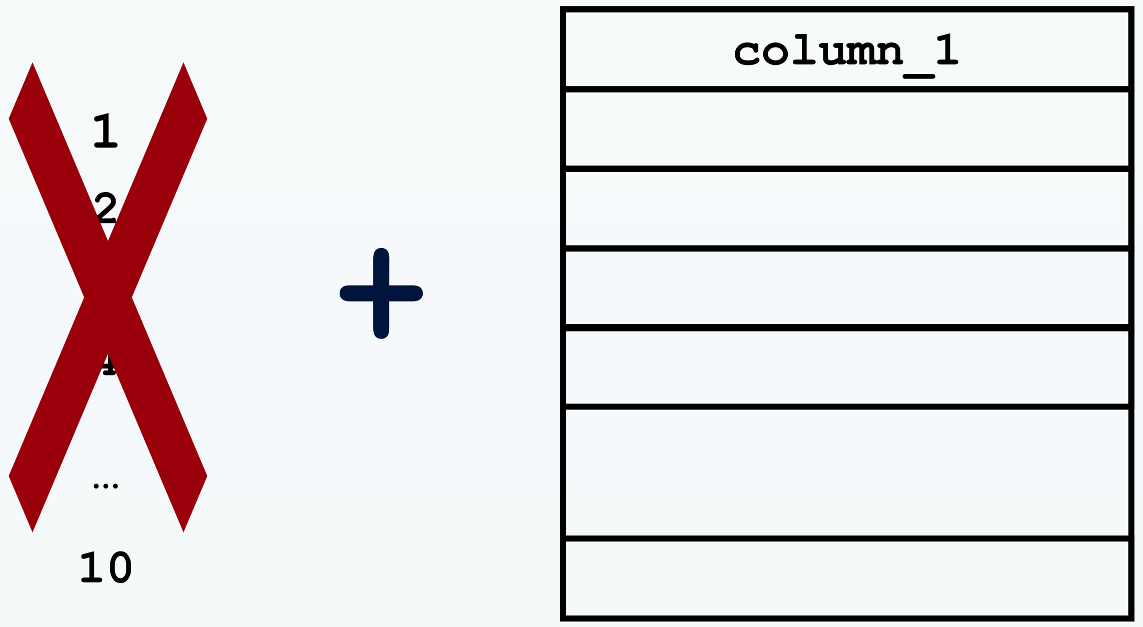
DROP vs TRUNCATE vs DELETE

TRUNCATE ~ DELETE without WHERE



DROP vs TRUNCATE vs DELETE

TRUNCATE ~ DELETE without WHERE



DROP vs TRUNCATE vs DELETE

<u>TRUNCATE</u>

when truncating, auto-increment values will be reset

DROP vs TRUNCATE vs DELETE

TRUNCATE

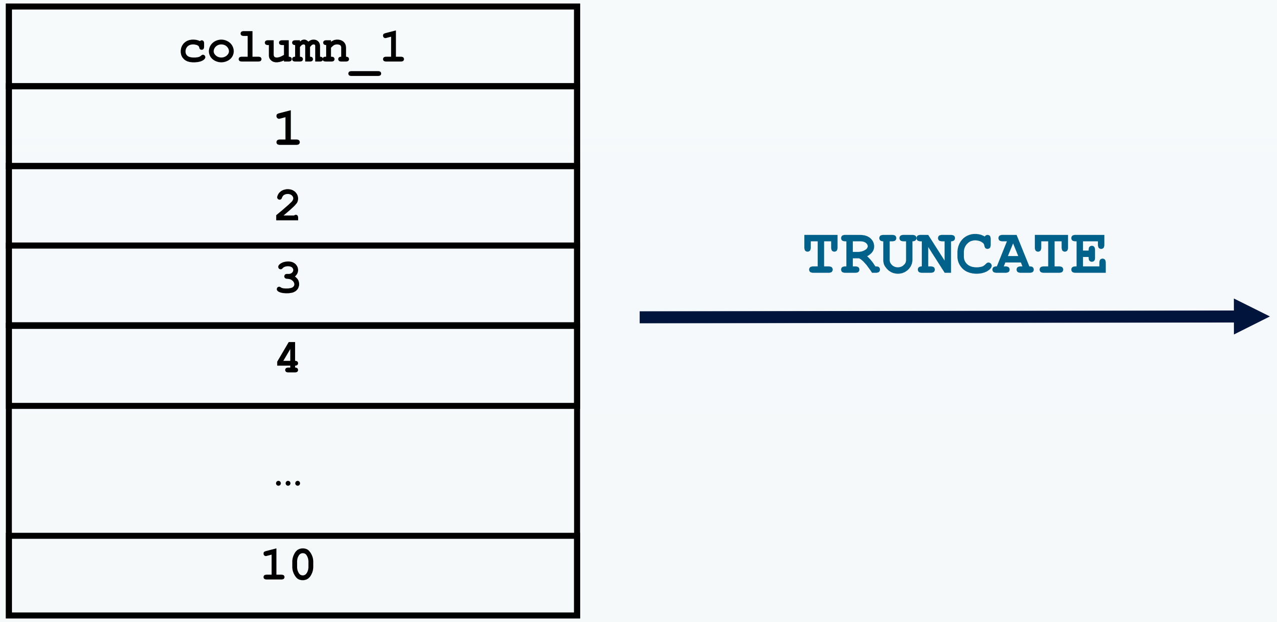
when truncating, auto-increment values will be reset

column_1
1
2
3
4
…
10

DROP vs TRUNCATE vs DELETE

TRUNCATE

when truncating, auto-increment values will be reset

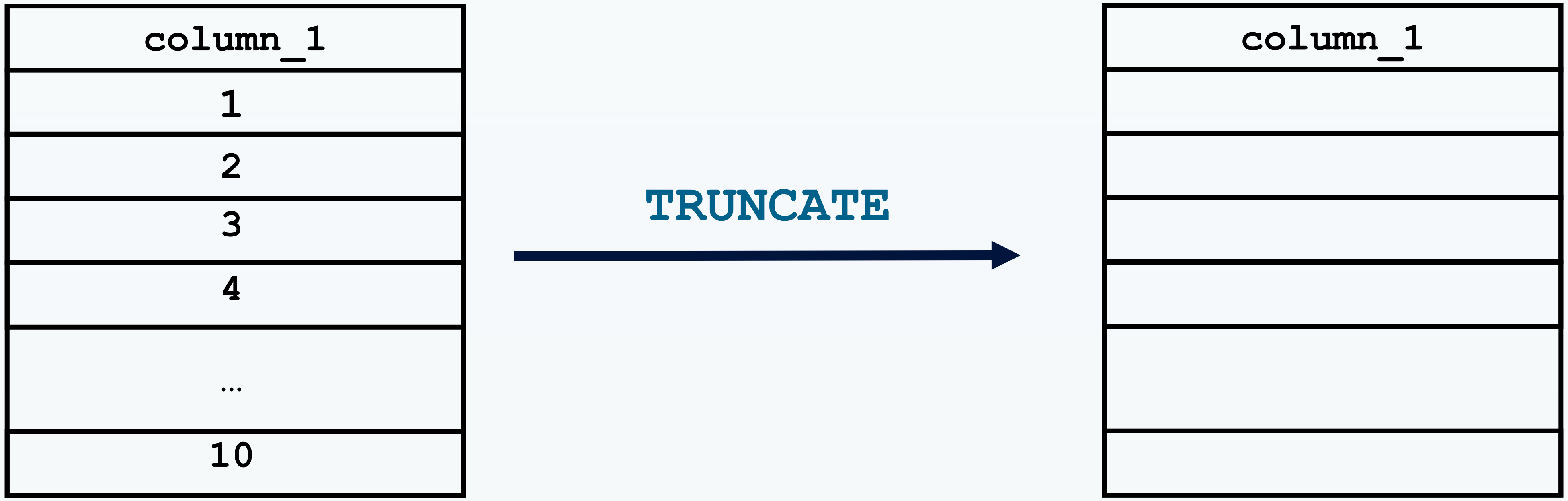


column_1
1
2
3
4
…
10

DROP vs TRUNCATE vs DELETE

TRUNCATE

when truncating, auto-increment values will be reset

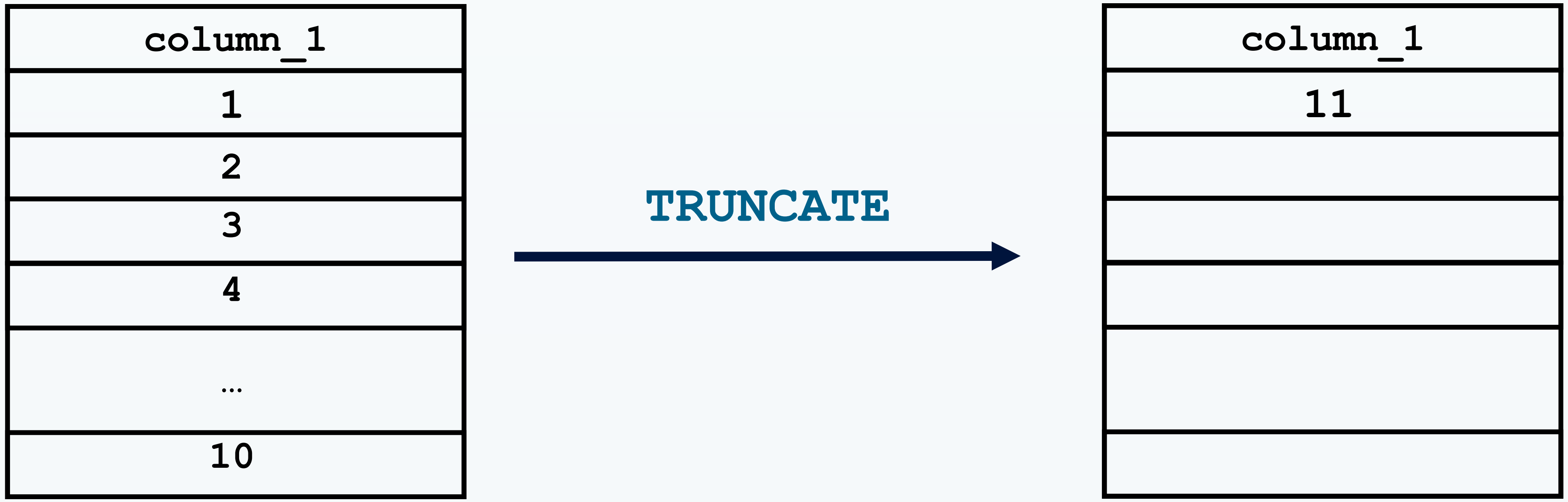


365√DataScience

DROP vs TRUNCATE vs DELETE

TRUNCATE

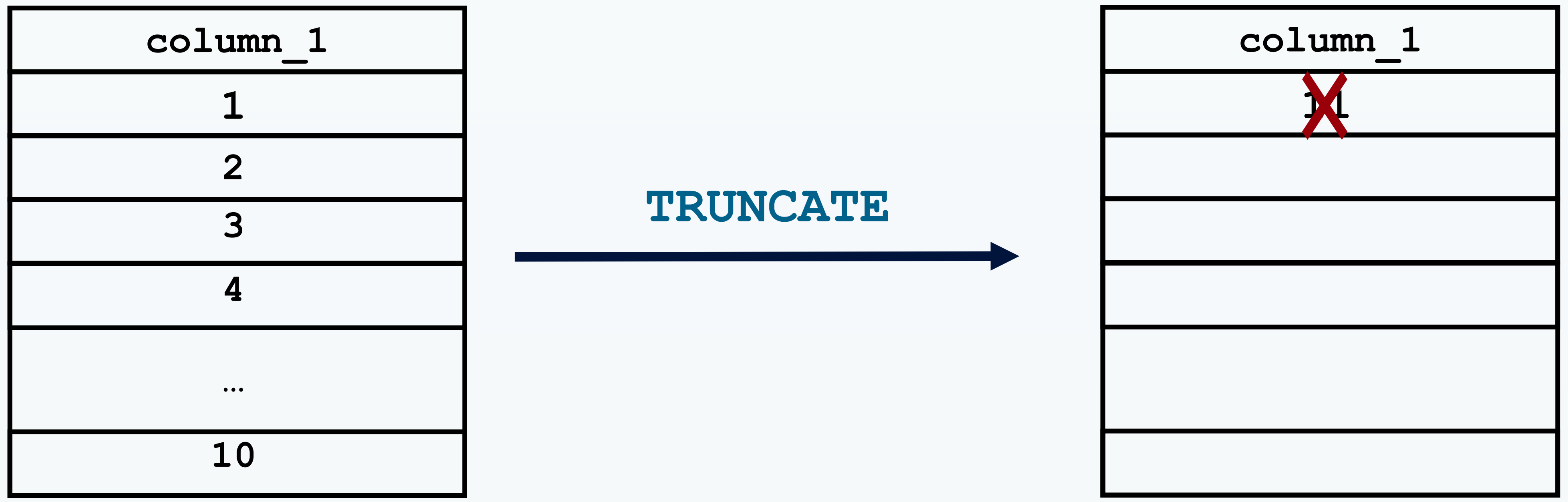
when truncating, auto-increment values will be reset



DROP vs TRUNCATE vs DELETE

<u>TRUNCATE</u>

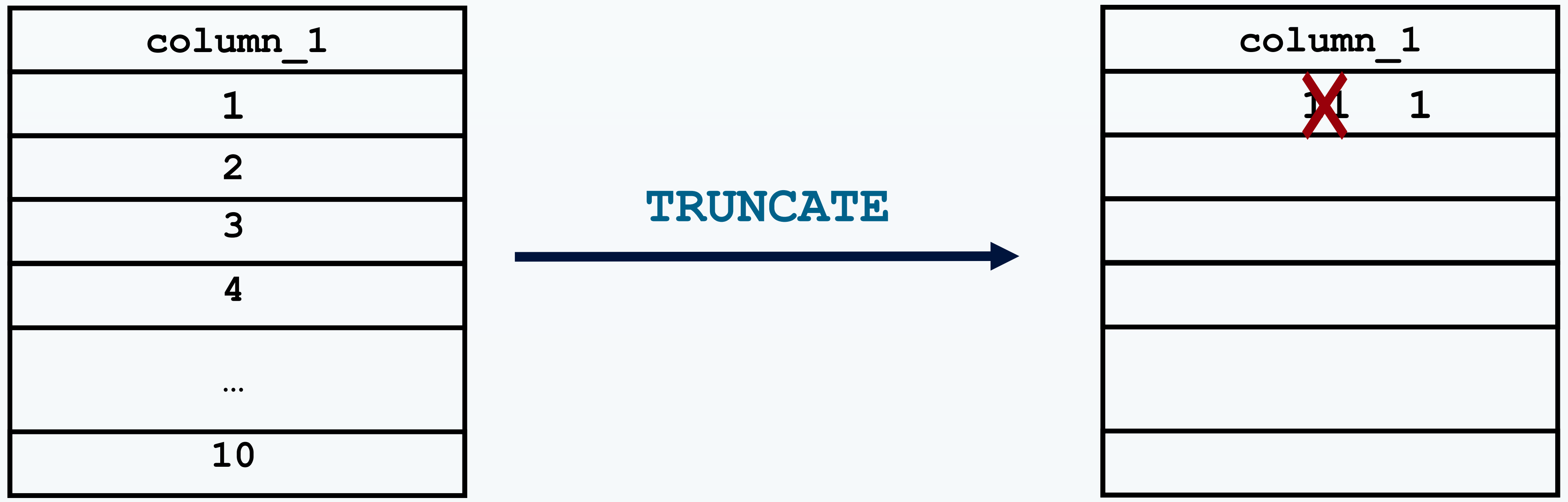
when truncating, auto-increment values will be reset



DROP vs TRUNCATE vs DELETE

TRUNCATE

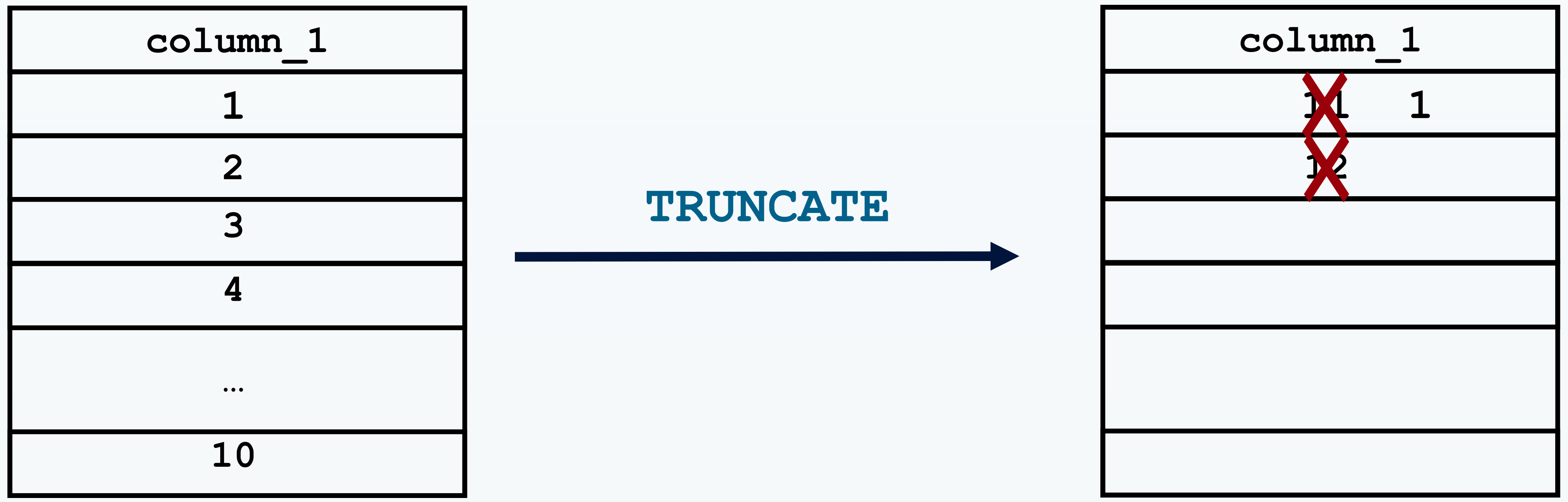
when truncating, auto-increment values will be reset



DROP vs TRUNCATE vs DELETE

TRUNCATE

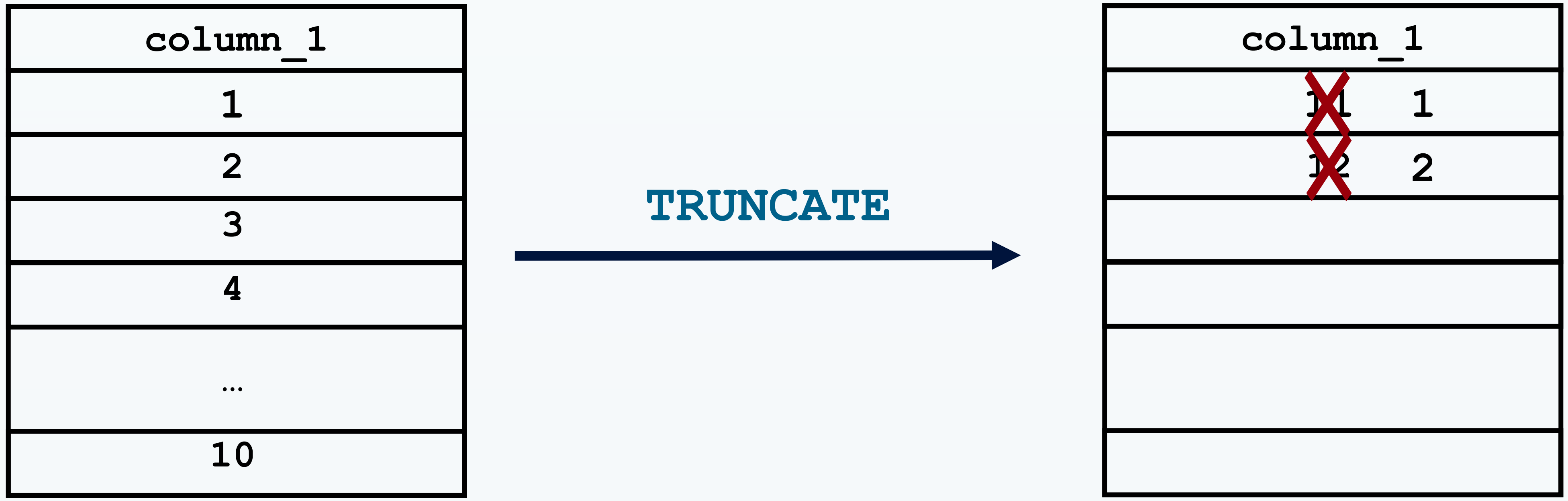
when truncating, auto-increment values will be reset



DROP vs TRUNCATE vs DELETE

TRUNCATE

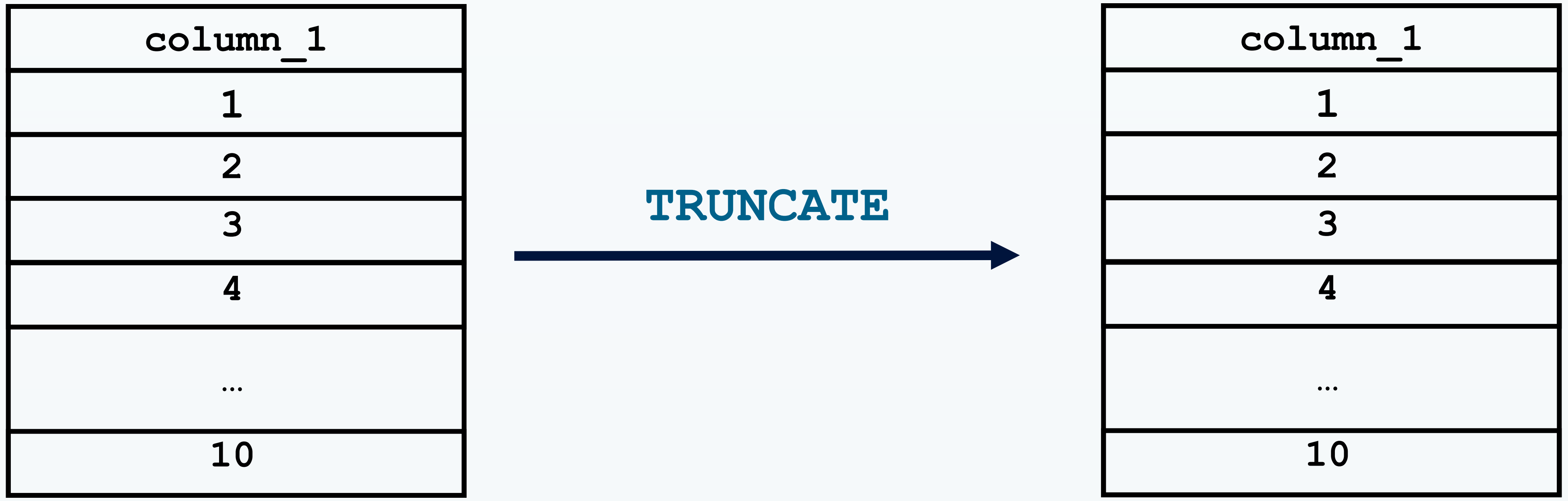
when truncating, auto-increment values will be reset



DROP vs TRUNCATE vs DELETE

TRUNCATE

when truncating, auto-increment values will be reset



DROP vs TRUNCATE vs DELETE

DELETE

removes records *row by row*

SQL

```
DELETE FROM table_name
WHERE conditions;
```

TRUNCATE ~ DELETE without WHERE

DROP vs TRUNCATE vs DELETE

TRUNCATE vs DELETE without WHERE

- the SQL optimizer will implement different programmatic approaches when we are using TRUNCATE or DELETE

↓

TRUNCATE delivers the output much *quicker* than DELETE

row by row *row by row*

DROP vs TRUNCATE vs DELETE

TRUNCATE vs DELETE without WHERE

- the SQL optimizer will implement different programmatic approaches when we are using TRUNCATE or DELETE

TRUNCATE delivers the output much *quicker* than DELETE

~~*row by row*~~ *row by row*

DROP vs TRUNCATE vs DELETE

TRUNCATE vs DELETE without WHERE

- auto-increment values are *not* reset with DELETE

DROP vs TRUNCATE vs DELETE

TRUNCATE vs DELETE without WHERE

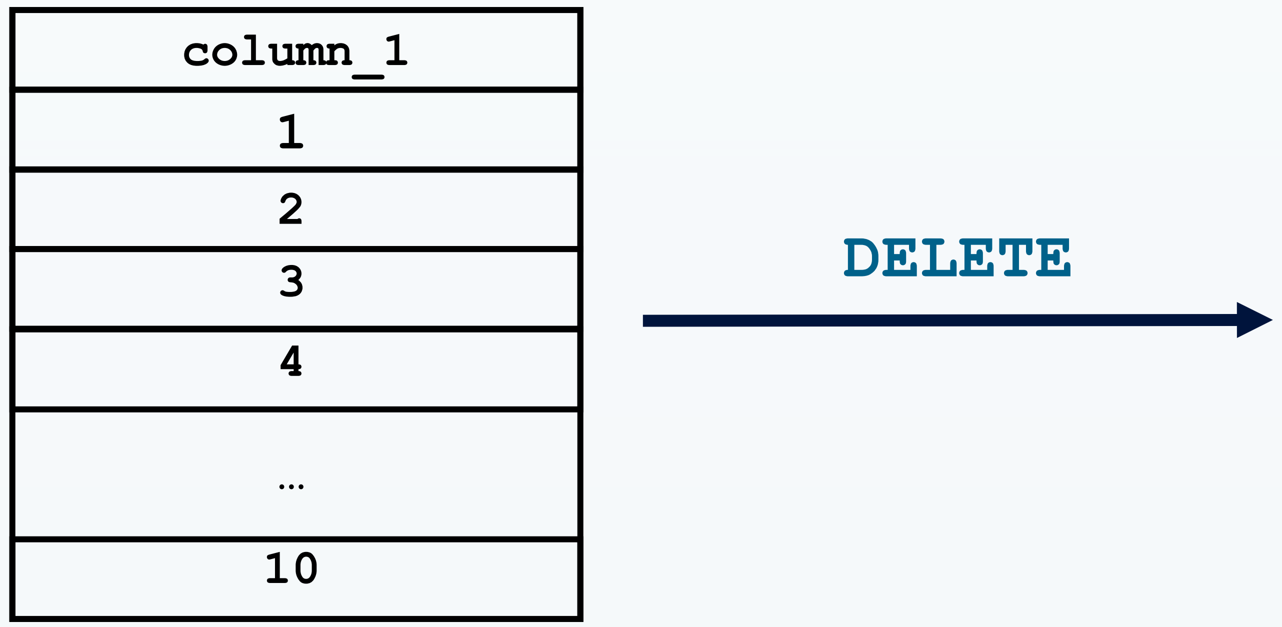
- auto-increment values are *not* reset with DELETE

column_1
1
2
3
4
…
10

DROP vs TRUNCATE vs DELETE

TRUNCATE vs DELETE without WHERE

- auto-increment values are *not* reset with DELETE

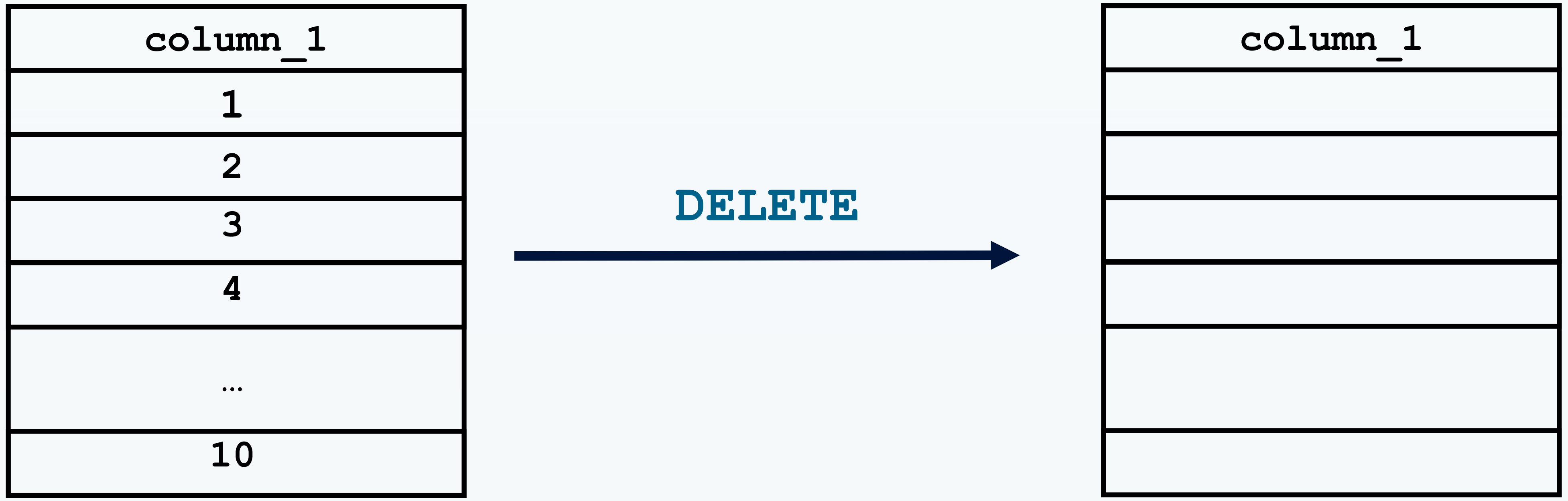


column_1
1
2
3
4
…
10

DROP vs TRUNCATE vs DELETE

TRUNCATE vs DELETE without WHERE

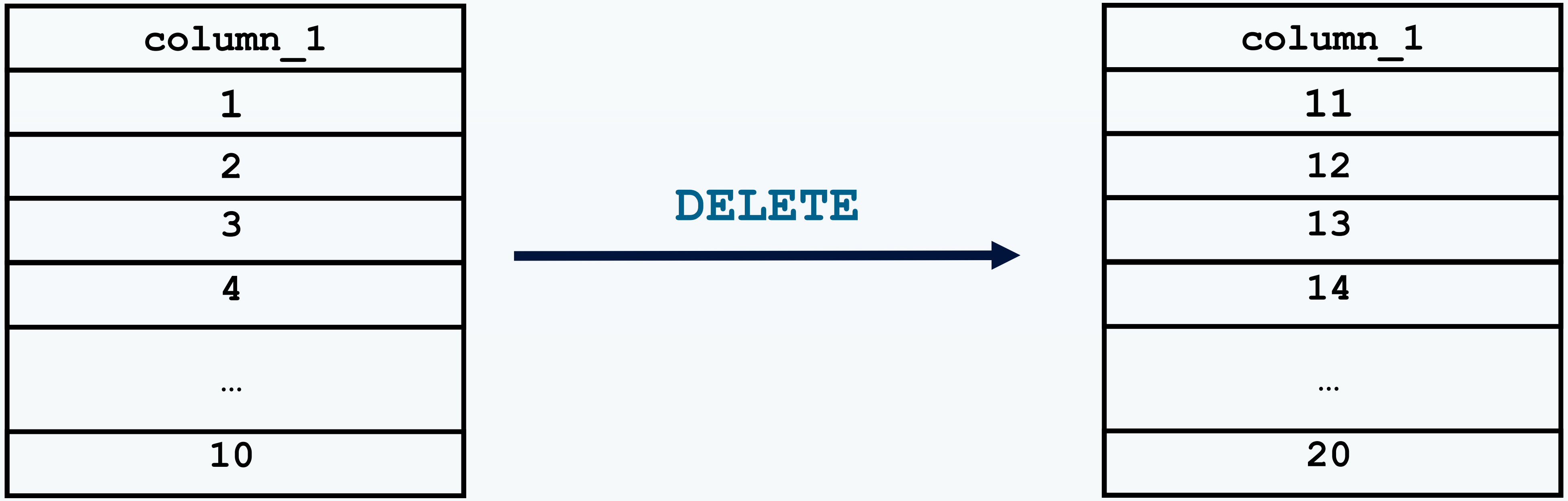
- auto-increment values are *not* reset with DELETE



DROP vs TRUNCATE vs DELETE

TRUNCATE vs DELETE without WHERE

- auto-increment values are *not* reset with DELETE


column_1
1
2
3
4
…
10
DELETE
column_1
11
12
13
14
…
20