**Introduction to Relational Databases in SQL**

# **Tables**

## **Query information schema tables**: information about all tables in database

SELECT table\_name

FROM information\_schema.tables

WHERE table\_schema = 'public';

## 

## **Query information schema columns**: information about all columns in all tables

SELECT column\_name, data\_type

FROM information\_schema.columns

WHERE table\_name = 'table\_name'

AND table\_schema = 'public';

## 

## **Creating simple tables:**

CREATE TABLE table\_name (

column\_a data\_type,

column\_b data\_type,

column\_c data\_type

);

## 

## **Adding column names:**

ALTER TABLE table\_name

ADD COLUMN column\_name data\_type;

## 

## **Insert distinct records from a table to a table:**

INSERT INTO target\_table

SELECT DISTINCT column\_a, column\_b

FROM source\_table

## **Insert distinct records into a table:**

INSERT INTO table\_name (column\_a, column\_b)

VALUES (“value\_a”, “value\_b”)

## **Rename a column:**

ALTER TABLE table\_name

RENAME COLUMN old\_name TO new\_name

## **Drop a column:**

ALTER TABLE table\_name

DROP COLUMN column\_name

## 

## **Drop table:**

DROP TABLE table\_name

## **Update table:**

UPDATE table\_name

SET col\_name = CONCAT (a, b);

## **Join tables:**

SELECT ...

FROM table\_a

JOIN table\_b

ON …=...

WHERE …

-- Select all professors working for universities in the city of Zurich

SELECT professors.lastname, universities.id, universities.university\_city

FROM professors

JOIN universities

ON professors.university\_id = universities.id

WHERE universities.university\_city = 'Zurich';

## **Update columns of a table based on values in another table:**

UPDATE table\_a

SET column\_to\_update = table\_b.column\_to\_update\_from

FROM table\_b

WHERE condition1 AND condition2 AND ...;

This query does the following:

1. For each row in table\_a, find the corresponding row in table\_b where condition1, condition2, etc., are met.
2. Set the value of column\_to\_update to the value of column\_to\_update\_from (from that corresponding row).
3. The conditions usually compare other columns of both tables, e.g. table\_a.some\_column = table\_b.some\_column. Of course, this query only makes sense if there is only *one* matching row in table\_b.

# **Attribute Constraints – e.g. data types on columns.**

## **Type Cast**

CREATE TABLE weather (

temperature integer, wind\_speed text);

SELECT temperature \* CAST(wind\_speed AS integer) AS wind\_chill

## **Common Data Types**

text

varchar [ (x) ] – max x characters

char [ (x) ] – fixed length string of x characters

boolean – TRUE, FALSE, NULL (unknown)

date, time, timestamp

numeric [ (x, y) ] - precision of x and scale of y (total x digits and y digits after .)

integer – whole numbers

serial - allows incrementing numbers in existing table in PostgreSQL

## **Alter data type**

ALTER TABLE table\_name

ALTER COLUMN column\_name

TYPE new\_dtype

## **Truncate or transform values before altering**

ALTER TABLE table\_name

ALTER COLUMN column\_name

TYPE new\_dtype

USING some\_function (column\_name)

## **Example – Convert types using a function**

ALTER TABLE table\_name

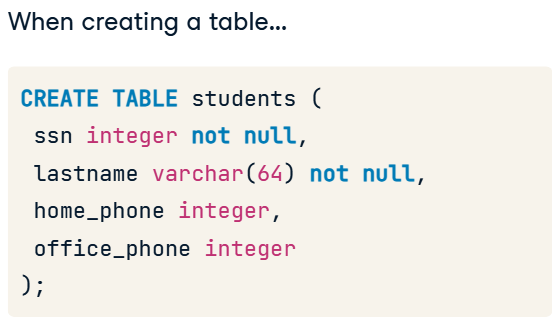
ALTER COLUMN column\_name

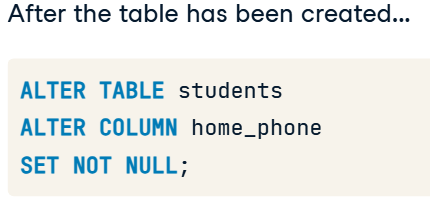
TYPE varchar(x)

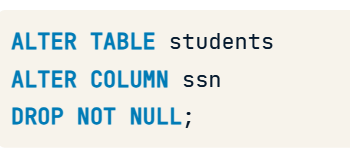
USING SUBSTRING(column\_name FROM 1 FOR x)

*Because you want to reserve only x characters for column\_name, you have to retain a SUBSTRING of every value, i.e. the first x characters of it, and throw away the rest. This way, the values will fit the varchar(x) requirement.*

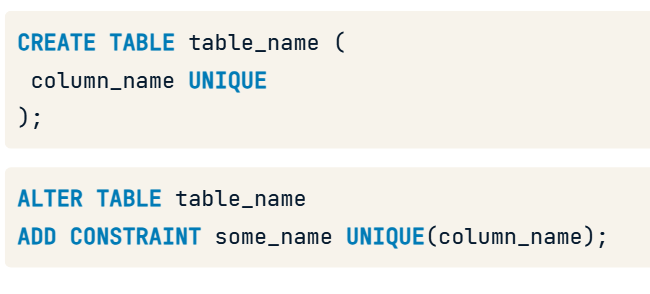
## **Add or Remove Not-Null constraint**







## **Adding Unique constraint – name the constraint**



# **Key Constraints - e.g. primary keys.**

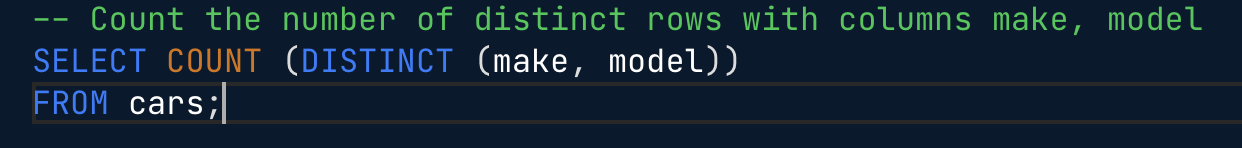
**Keys are denoted by underlined attribute names in the entity-relationship diagram**

## **Keys, superkeys and candidate keys**

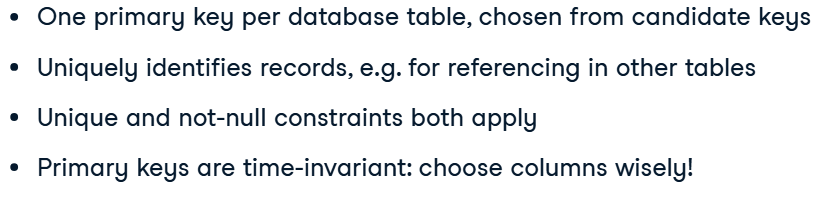
* The combination of all attributes is a key in itself.
* However, it's not called a key, but a superkey, if attributes from that combination can be removed, and the attributes still uniquely identify records.
* If all possible attributes have been removed but the records are still uniquely identifiable by the remaining attributes, we speak of a minimal superkey.
* This is the actual key. So a key is always minimal.

## **Identify keys with SELECT COUNT DISTINCT**

1. Count the distinct records for all possible combinations of columns. If the resulting number x equals the number of all rows in the table for a combination, you have discovered a superkey.
2. Then remove one column after another until you can no longer remove columns without seeing the number x decrease. If that is the case, you have discovered a (candidate) key.



## **Primary keys**



## **Specify Primary Key**

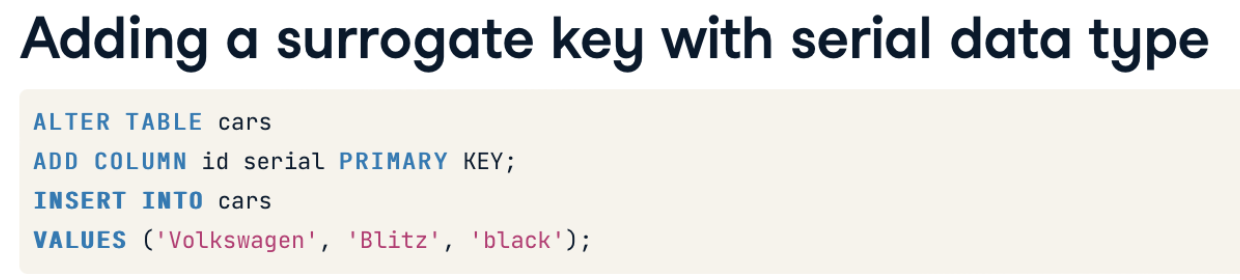
## 

## 

## **Surrogate Key**

* Artificial primary key - not based on a native column in data, but on a column that just exists for the sake of having a primary key.
* A primary key is ideally constructed from as few columns as possible and should never change over time.
* Defining an artificial primary key (ideally consisting of a unique number or string) - can be sure that this stays the same for each record.
* Other attributes might change, but the primary key always has the same value for a given record.

## **Add surrogate key**



## 

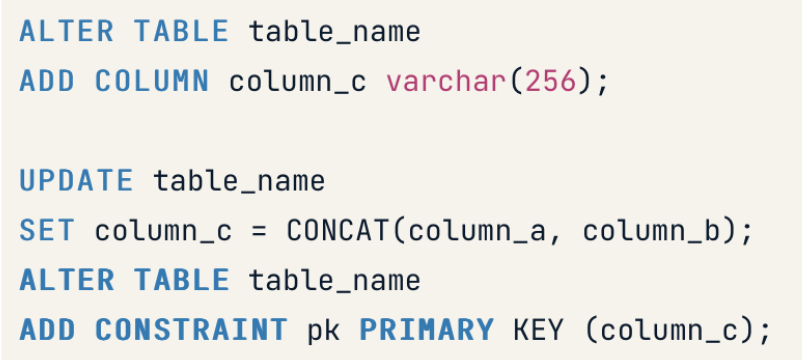
## 

## 

## 

## **Another type of surrogate key**

* Combine two existing columns into a new one.
* In this example, ADD new column with the "varchar" data type.
* UPDATE that column with the concatenation of two existing columns.
* (CONCAT function concatenates values of two or more existing columns).
* Turn that new column into a surrogate primary key.



-- Count the number of distinct rows with columns make, model

SELECT COUNT(DISTINCT(make, model))

FROM cars;

-- Add the id column

ALTER TABLE cars

ADD COLUMN id varchar(128);

-- Update id with make + model

UPDATE cars

SET id = CONCAT(make, model);

-- Make id a primary key

ALTER TABLE cars

ADD CONSTRAINT id\_pk PRIMARY KEY (id);

# **Referential integrity Constraints – enforced through foreign keys.**

* A record referencing another record in another table must always refer to an existing record.
* In other words: A record in table A cannot point to a record in table B that does not exist.
* Referential integrity is held from A to B.
* Referential integrity is a constraint that always concerns two tables, and is enforced through foreign keys,

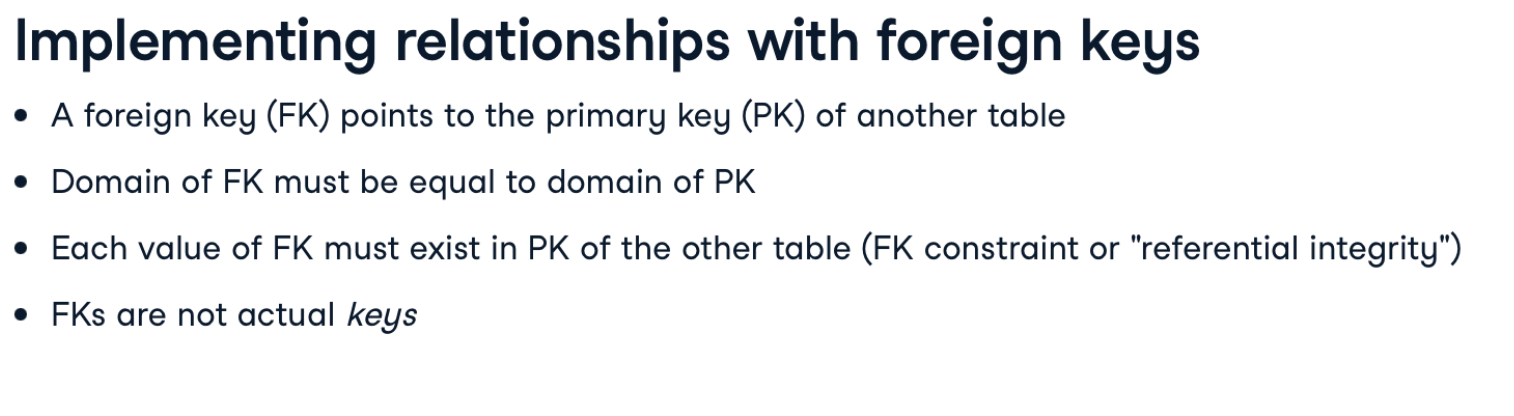
## **1:N relationships with foreign keys**

**Drawn with a rhombus in the Entity Relationship Diagram**

**small numbers denote cardinality**

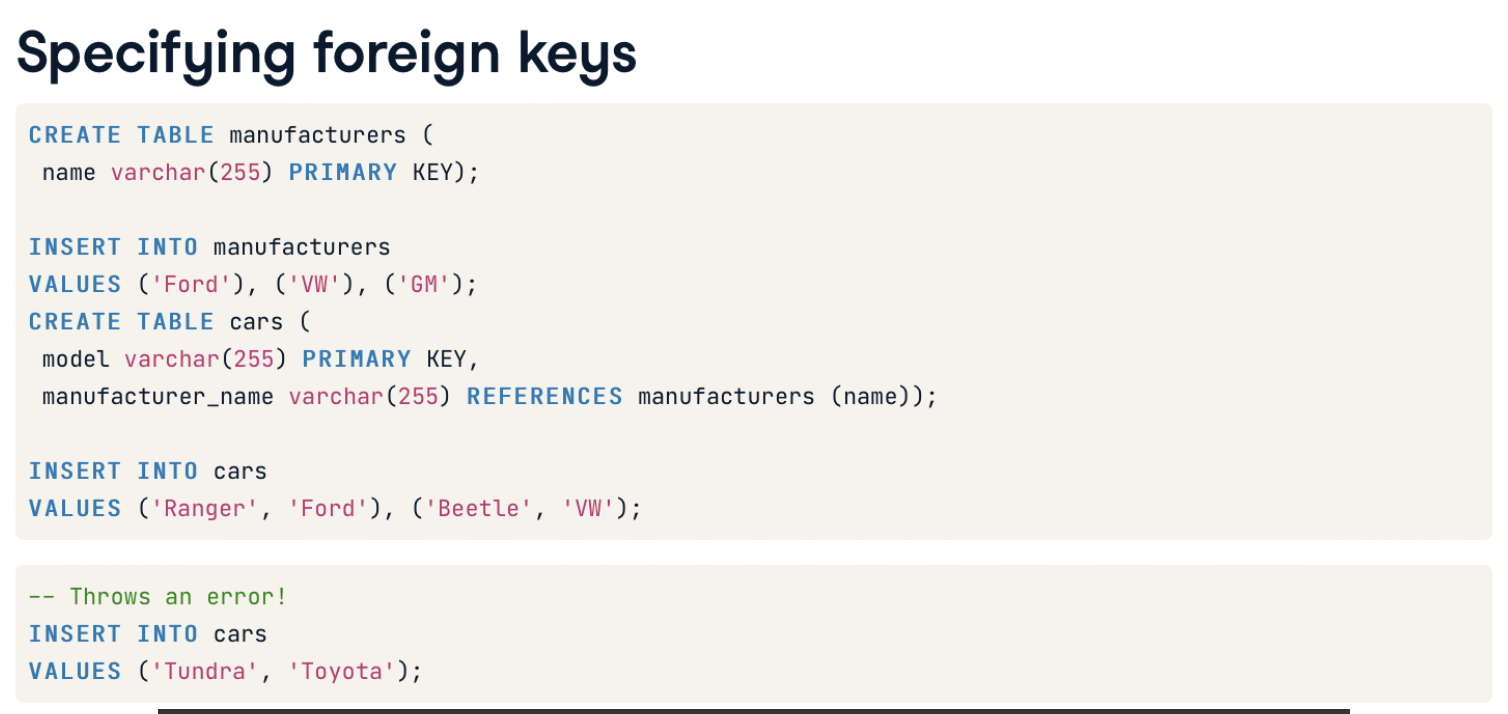
## 

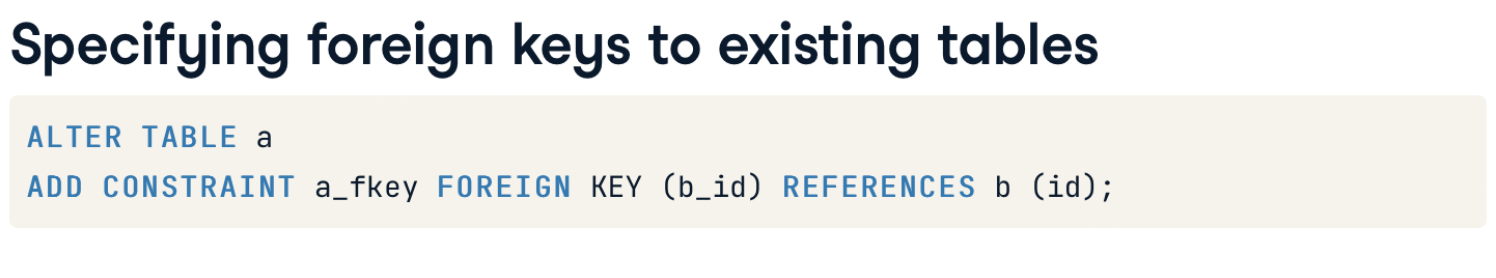
## **Foreign Keys - REFERENCES keyword**

****

## **Specify foreign key**

**naming convention** employed here: Usually, a foreign key referencing another primary key with name id is named x\_id, where x is the name of the referencing table in the singular form.

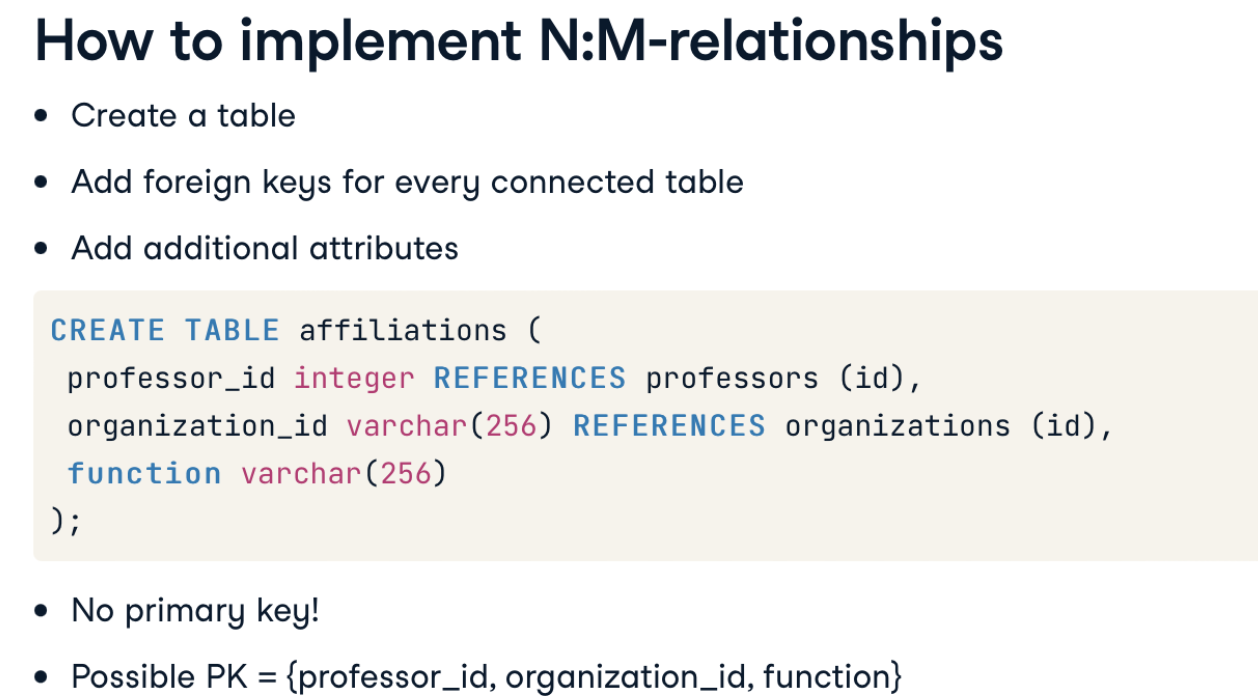
****

****

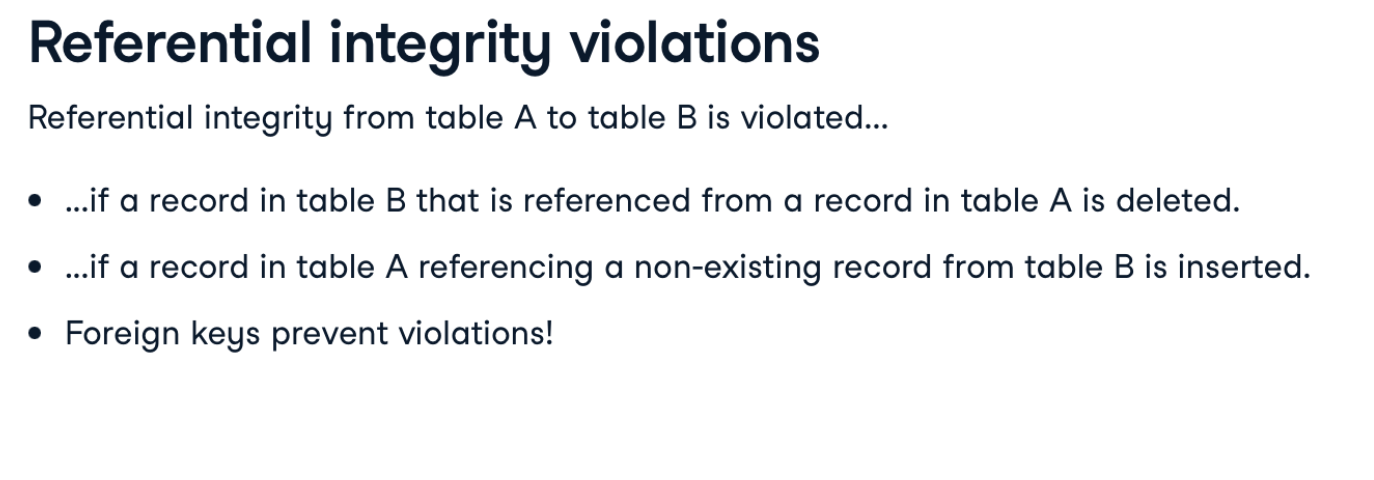
## **N:M relationships with foreign keys**

**Drawn with an oval in the Entity Relationship Diagram**

## **Implement N:M\_relationships**

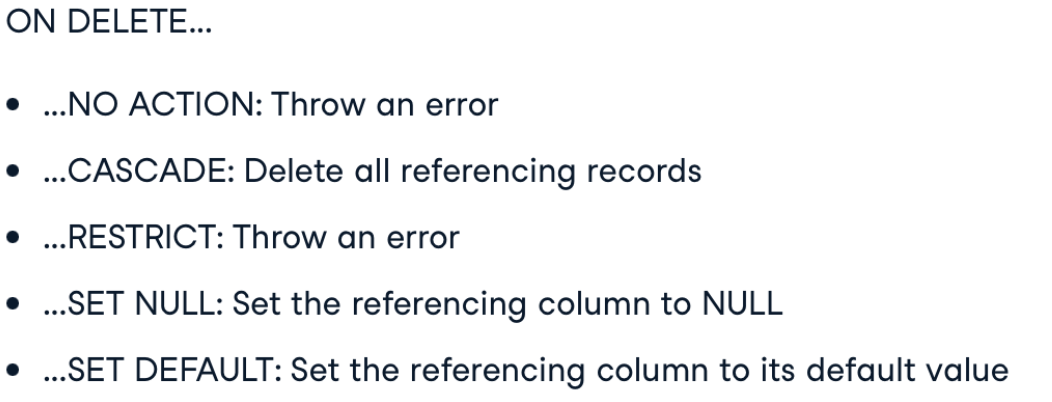


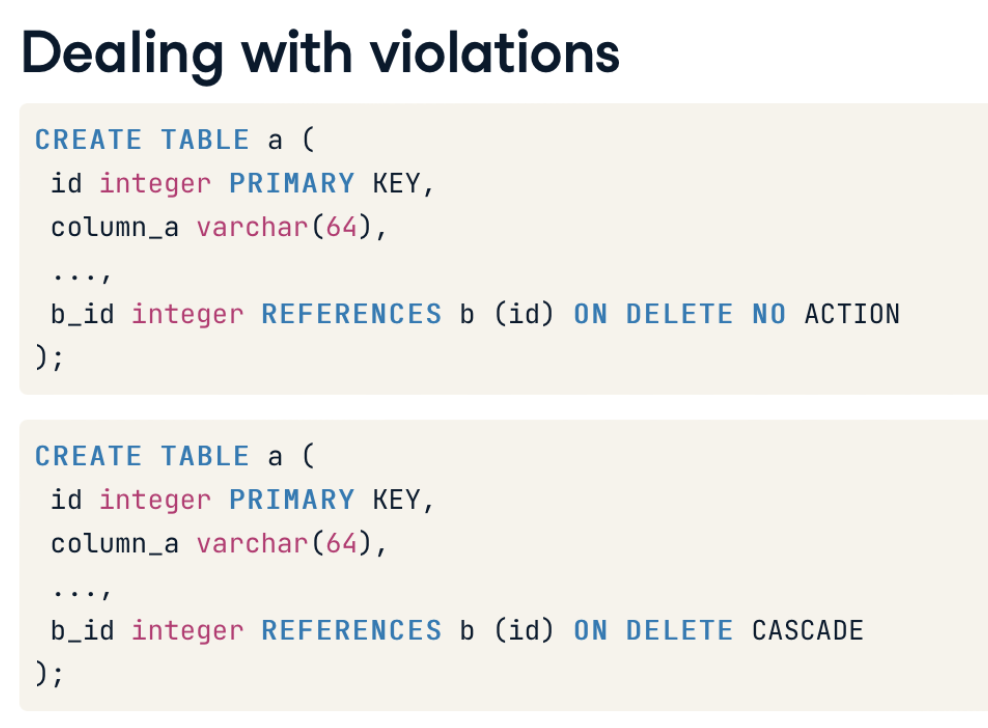
## **Referential Integrity Violations**

****

## **Dealing with violations**

ON DELETE NO ACTION IS DEFAULT



****

# **Identify constraints from table schema**

## **Identify the correct constraint name**

SELECT constraint\_name, table\_name, constraint\_type

FROM information\_schema.table\_constraints

WHERE constraint\_type = 'FOREIGN KEY';