

Relational Data Integrity

Big Data Systems

Dr. Rubi Boim

Relational Data Integrity

- DB Constraints
- ACID & Transactions

DB Constraints

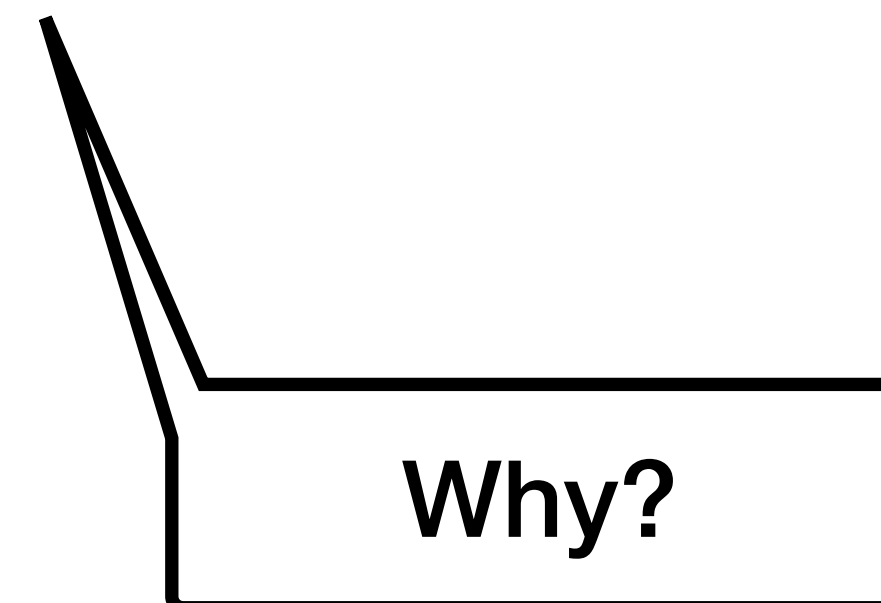
Note - Syntax can vary between relation databases

DB Constraints

Goal: ensure the accuracy and reliability of the data

- Column level
- Table level

Warning: Most of these won't apply to Wide column databases



Not NULL

Ensures that a column cannot have a NULL value

```
CREATE TABLE users (  
  user_id INT NOT NULL,  
  name VARCHAR(255),  
  city VARCHAR(255) NOT NULL,  
  birthdate DATE,  
  PRIMARY KEY (user_id)  
)
```

users

<u>user_id</u>	name	city	brithdate
101	Rubi Boim	Tel Aviv	<null>
102	Tova Milo	Tel Aviv	<null>
103	Lebron James	Los Angeles	30/12/1984
104	Michael Jordan	Chicago	17/02/1963

DEFAULT

Sets a default value for a column if no value is specified

```
CREATE TABLE users (  
  user_id INT NOT NULL,  
  name VARCHAR(255),  
  city VARCHAR(255) NOT NULL,  
  last_update DATE DEFAULT GETDATE(),  
  PRIMARY KEY (user_id)  
)
```

users

<u>user_id</u>	name	city	last_update
101	Rubi Boim	Tel Aviv	24/10/2021
102	Tova Milo	Tel Aviv	20/09/2021
103	Lebron James	Los Angeles	24/10/2021
104	Michael Jordan	Chicago	01/10/2019

CHECK

Ensures that a column satisfies a specific condition

```
CREATE TABLE users (  
  user_id INT NOT NULL,  
  name VARCHAR(255),  
  city VARCHAR(255) NOT NULL,  
  age INT,  
  PRIMARY KEY (user_id),  
  CONSTRAINT CHK_Person CHECK (age >= 18)  
)
```

<u>user_id</u>	name	city	age
101	Rubi Boim	Tel Aviv	39
102	Tova Milo	Tel Aviv	24
103	Lebron James	Los Angeles	36
104	Michael Jordan	Chicago	58

CHECK (1)

Ensures that a column satisfies a specific condition

```
CREATE TABLE users (  
  user_id INT NOT NULL,  
  name VARCHAR(255),  
  city VARCHAR(255) NOT NULL,  
  age INT,  
  PRIMARY KEY (user_id),  
  CONSTRAINT CHK_Person CHECK (age >= SELECT avg(age)  
                                FROM users)  
)
```

users

user_id	name	city	age
101	Rubi Boim	Tel Aviv	39
102	Tova Milo	Tel Aviv	24
103	Lebron James	Los Angeles	36
104	Michael Jordan	Chicago	58

CHECK (1)

Ensures that a column satisfies a specific condition

```
CREATE TABLE users (  
  user_id INT NOT NULL,  
  name VARCHAR(255),  
  city VARCHAR(255) NOT NULL,  
  age INT,  
  PRIMARY KEY (user_id),  
  CONSTRAINT CHK_Person CHECK (age >= SELECT avg(age)  
                                  FROM users)  
)
```

What would happen if we update another row that will change the average?

user_id	name	city	age
101	Rubi Boim	Tel Aviv	39
102	Tova Milo	Tel Aviv	24
103	Lebron James	Los Angeles	36
104	Michael Jordan	Chicago	58

CHECK (1)

Ensures that a column satisfies a specific condition

CREATE

Different DBs = Different behaviors
—> Don't do this...

What would happen if we update another row that will change the average?

NULL,

```
age INT,  
PRIMARY KEY (user_id),  
CONSTRAINT CHK_Person CHECK (age >= SELECT avg(age)  
FROM users  
users  
)
```

user_id	name	city	age
101	Rubi Boim	Tel Aviv	39
102	Tova Milo	Tel Aviv	24
103	Lebron James	Los Angeles	36
104	Michael Jordan	Chicago	58

CHECK (1)

Ensures that a column satisfies a specific condition

CREATE

Different DBs = Different behaviors
→ Don't do this...

What would happen if we update another row that will change the average?

```
age INT,  
PRIMARY KEY (user_id),  
CONSTRAINT CHK_Person CHECK (age >= SELECT avg(age)  
FROM users)  
users
```

BTW - MySQL does not support
"CHECK" at all
(Accepts the syntax but does nothing)

id	name	city	age
	Rubi Boim	Tel Aviv	39
	Tova Milo	Tel Aviv	24
	Lebron James	Los Angeles	36
	Michael Jordan	Chicago	58

CHECK (1)

Ensures that a column satisfies a specific condition

CREATE

Different DBs = Different behaviors
—> Don't do this...

What would happen if we update another row that will change the average?

```
age INT,  
PRIMARY KEY (user_id),  
CONSTRAINT CHK_Person CHECK (age >= SELECT avg(age)  
FROM users)  
users
```

BTW - MySQL does not support "CHECK" at all
(Accepts the syntax but does nothing)

Use "triggers" (before insert)

	id			
104	12	Michael Jordan	Chicago	58

UNIQUE

Ensures that all values in a column are different

```
CREATE TABLE users (  
  user_id INT NOT NULL,  
  name VARCHAR(255) UNIQUE,  
  city VARCHAR(255) NOT NULL,  
  birthdate DATE,  
  PRIMARY KEY (user_id)  
)
```

Discussion

(1) - should "name" be unique?

(2) - how the DB enforce this?

users

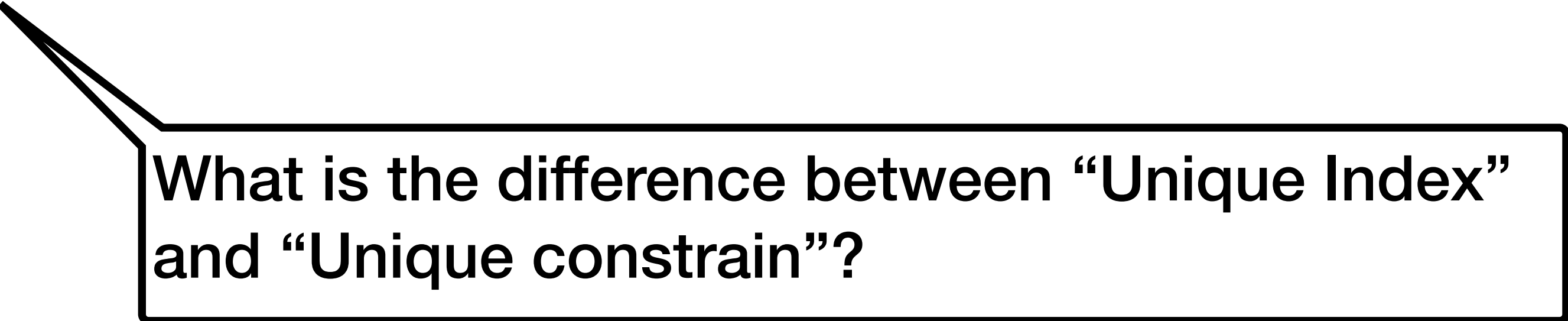
<u>user_id</u>	name	city	brithdate
101	Rubi Boim	Tel Aviv	<null>
102	Tova Milo	Tel Aviv	<null>
103	Lebron James	Los Angeles	30/12/1984
104	Michael Jordan	Chicago	17/02/1963

INDEX

Used to retrieve data from the database very quickly

```
CREATE INDEX index_name  
ON table_name (column1, column2, ...)
```

```
CREATE UNIQUE INDEX index_name  
ON table_name (column1, column2, ...)
```



What is the difference between “Unique Index” and “Unique constrain”?

PRIMARY KEY

Uniquely identifies each row in a table

- A combination of a NOT NULL and UNIQUE
- A table can have only ONE primary key
- can consist of single or multiple columns

```
CREATE TABLE views (  
  user_id INT NOT NULL,  
  movie_id INT NOT NULL,  
  view_time DATETIME,  
  PRIMARY KEY (user_id, movie_id)  
)
```

<u>user_id</u>	<u>movie_id</u>	<u>view_time</u>
101	2003	24/02/2021
101	2004	25/02/2021
103	2004	26/02/2021
103	2005	26/02/2021

FOREIGN KEY

Prevents actions that would destroy links between tables

- A **foreign key** field(s) in one table refers to the **primary key** in another table

users			cities	
<u>user_id</u>	name	city_id	<u>id</u>	name
101	Rubi Boim	5	5	Tel Aviv
102	Tova Milo	5	6	Los Angeles
103	Lebron James	6	7	Chicago
104	Michael Jordan	7		

FOREIGN KEY (1)

users			cities	
<u>user_id</u>	name	city_id	<u>id</u>	name
101	Rubi Boim	5	5	Tel Aviv
102	Tova Milo	5	6	Los Angeles
103	Lebron James	6	7	Chicago
104	Michael Jordan	7		
105	Luka Doncic	8		

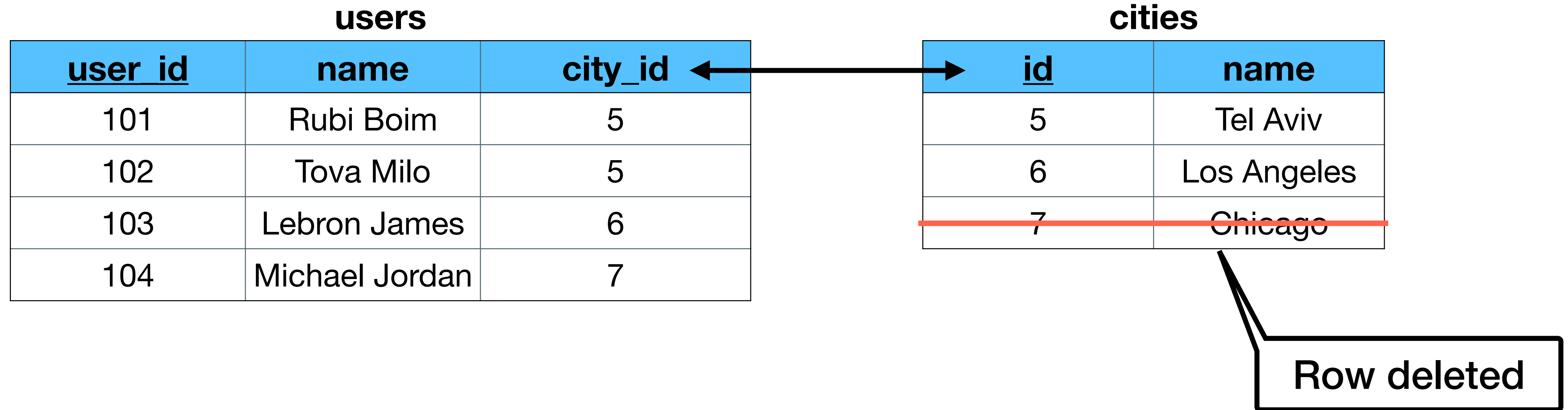
New row is inserted

FOREIGN KEY (1)

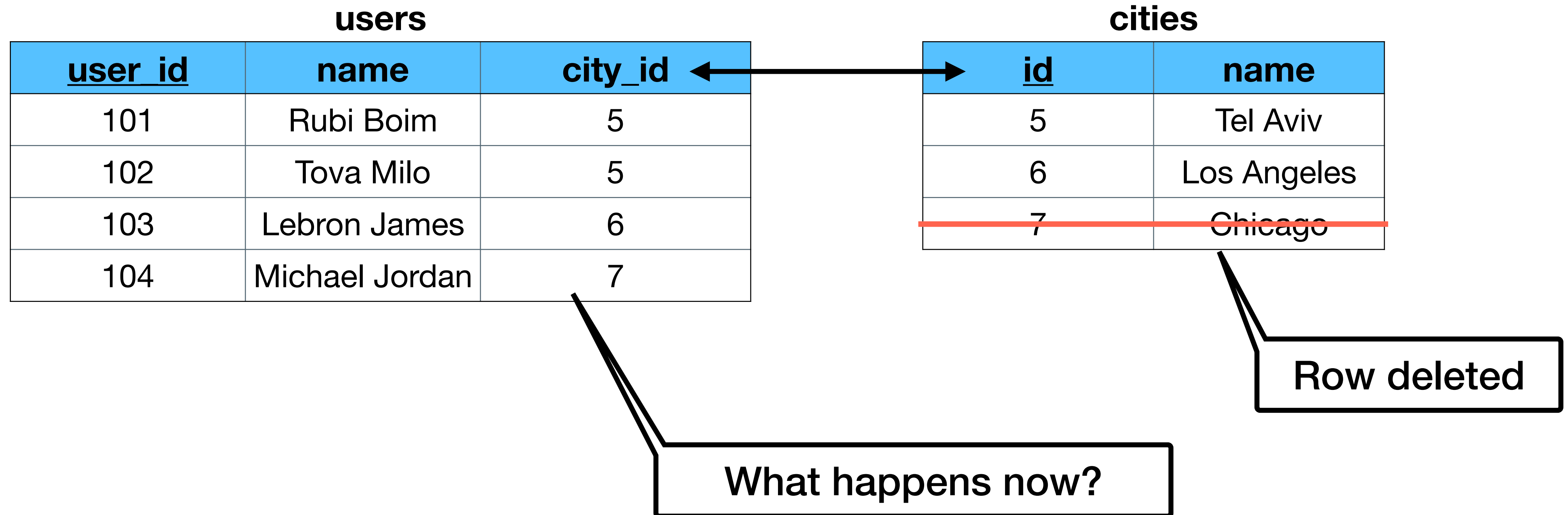
users			cities	
<u>user_id</u>	name	city_id	<u>id</u>	name
101	Rubi Boim	5	5	Tel Aviv
102	Tova Milo	5	6	Los Angeles
103	Lebron James	6	7	Chicago
104	Michael Jordan	7		
105	Luka Doncic	8		

What happens now?

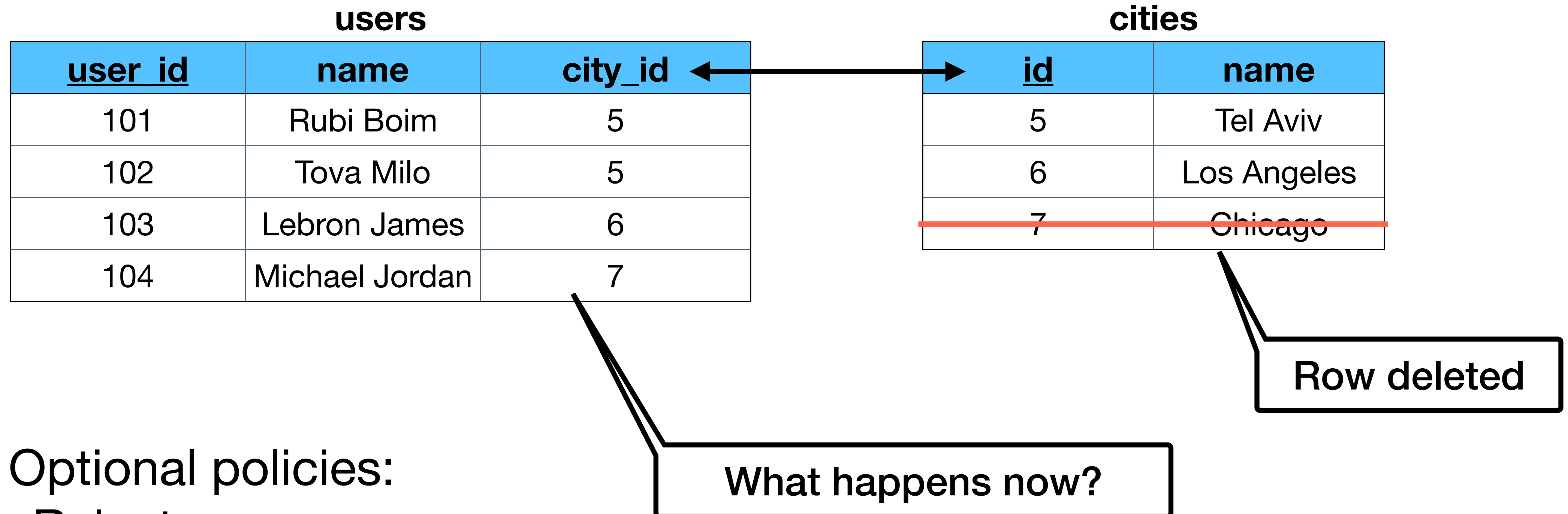
FOREIGN KEY (2)



FOREIGN KEY (2)



FOREIGN KEY (2)




Optional policies:

- Reject
- Set NULL
- Cascade (!)

FOREIGN KEY (3)

users			cities	
<u>user_id</u>	name	city_id	<u>id</u>	name
101	Rubi Boim	5	5	Tel Aviv
102	Tova Milo	5	6	Los Angeles
103	Lebron James	6	7	Chicago
104	Michael Jordan	7		



```
CREATE TABLE users (  
  user_id INT NOT NULL,  
  name VARCHAR(255),  
  city_id INT NOT NULL,  
  PRIMARY KEY (user_id),  
  FOREIGN KEY (city_id)  
  REFERENCES cities(id) ON DELETE CASCADE  
)
```

FOREIGN KEY (4)

users			cities	
<u>user_id</u>	name	city_id	<u>id</u>	name
101	Rubi Boim	5	5	Tel Aviv
102	Tova Milo	5	6	Los Angeles
103	Lebron James	6	7	Chicago
104	Michael Jordan	7		

```
CREATE TABLE users (  
  user_id INT NOT NULL,  
  name VARCHAR(255),  
  city_id INT NOT NULL,  
  PRIMARY KEY (user_id),  
  CONSTRAINT CHK_city CHECK (city_id IN  
    (SELECT id FROM cities))  
)
```

Is "CHECK" the same as foreign key?

ACID and Transactions

Probably the best feature of RDBMS



Why do we need transactions?

- If only a single user use the system —>



hakuna matata

- Clearly this is not the case.

Actually this is not true...

Transactions solves two issues

- Isolation from concurrent access
- Recovery from failures

Transaction properties: ACID

ACID

- **Atomicity**
- **Consistency**
correctness / referential integrity (foreign key) - **NOT like in CAP**
- **Isolation**
level can be configured
- **Durability**

Transactions - example (1)

START TRANSACTION

```
UPDATE table1 SET column = "..."
```

```
UPDATE table2 SET column = "..."
```

...

COMMIT

Transactions - example (2)

```
START TRANSACTION
```

```
SELECT @A:=count(balance) FROM bank_accounts WHERE uid=123
```

```
UPDATE table2 SET column = @A
```

```
...
```

```
COMMIT
```

Transactions - example (3)

```
START TRANSACTION
```

```
SELECT @A:=count(balance) FROM bank_accounts WHERE uid=123
```

```
UPDATE table2 SET column = @A
```

```
...
```

```
ROLLBACK
```

Isolation levels (Transactions)

- Read uncommitted
dirty reads
- Read committed
non repeatable reads
- Repeatable reads
phantom reads
- Serializable
default

Read uncommitted (dirty reads)

Transaction 1

```
// query 1  
SELECT age FROM users WHERE id=1;  
// will read 36
```

```
// query 1  
SELECT age FROM users WHERE id=1;  
// will read 20
```

Transaction 2

```
// query 2  
UPDATE users SET age=20 WHERE id=1;  
// no commit yet
```

```
// query 2  
ROLLBACK;
```

users

<u>id</u>	name	age
1	Lebron James	36
2	Michael Jordan	58

Read committed (non repeatable reads)

Transaction 1

```
// query 1  
SELECT age FROM users WHERE id=1;  
// will read 36
```

```
// query 1  
SELECT age FROM users WHERE id=1;  
// will read 20
```

Transaction 2

```
// query 2  
UPDATE users SET age=20 WHERE id=1;  
// no commit yet
```

```
// query 2  
COMMIT;
```

users

id	name	age
1	Lebron James	36
2	Michael Jordan	58

Repeatable reads (phantom reads)

Transaction 1

```
// query 1  
SELECT * FROM users WHERE age>20;
```

```
// query 1  
SELECT * FROM users WHERE age>20;
```

Transaction 2

```
// query 2  
INSERT INTO users VALUES (3, "aa", 40);  
COMMIT;
```

users

<u>id</u>	name	age
1	Lebron James	36
2	Michael Jordan	58

Serializable

Transaction 1

```
// query 1  
SELECT * FROM users WHERE age>20;  
...  
SELECT * FROM users WHERE age>20;
```

Transaction 2

```
// query 2  
INSERT INTO users VALUES (3, "aa", 40);  
COMMIT;
```

The highest level of isolation

users		
<u>id</u>	name	age
1	Lebron James	36
2	Michael Jordan	58