SQL primary key, surrogate key, composite keys, foreign keys... and JPA

Franck Pachot, Developer Advocate

yugabyteDB

Franck Pachot

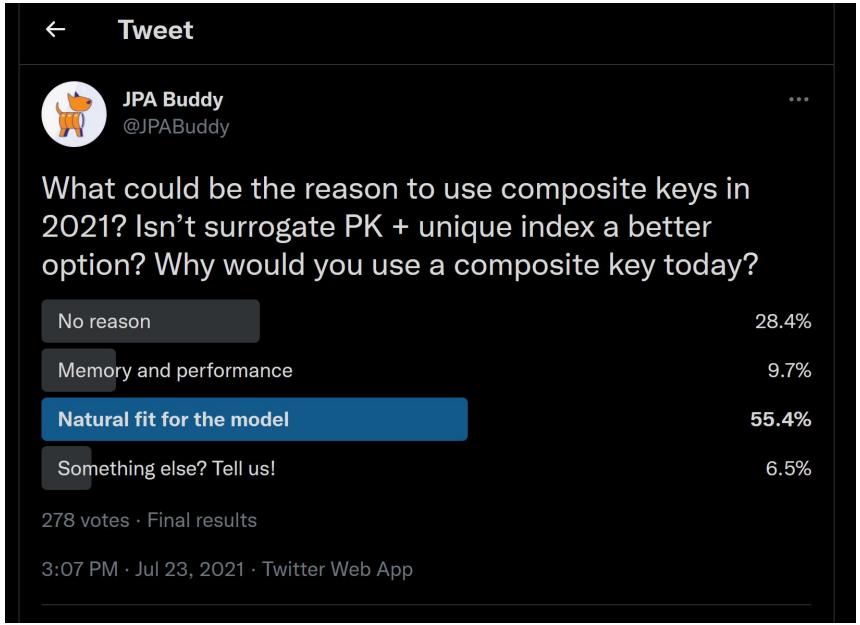
Developer Advocate on YugabyteDB
(PostgreSQL-compatible distributed database)

Past:

- 20 years in databases, dev and ops
- Oracle ACE Director, AWS Data Hero
- Oracle Certified Master, AWS Database Specialty

fpachot@yugabyte.com
dev.to/FranckPachot
@FranckPachot

This presentation idea started with...



Demo

what is a key?
what is a primary key?
what is a surrogate key?
what is a foreign key?
what is a composite key?

A table may have no key

```
yb_demo_northwind=# select * from people where last_name='Pachot';
 first_name | last_name | birth_date |     email     | twitter_handle | tax_country |  tax_id
------------+-----------+------------+---------------+----------------+-------------+----------
 Franck     | Pachot    | 1971-02-08 | me@pachot.net | @FranckPachot  | CH          | 12345689
(1 row)

yb_demo_northwind=# select * from people where last_name='Doe';
 first_name | last_name | birth_date |    email     | twitter_handle | tax_country |  tax_id
------------+-----------+------------+--------------+----------------+-------------+----------
 John       | Doe       | 1970-01-01 | john@dow.net |                | US          | 11223344
 Jane       | Doe       | 1970-01-01 | jane@dow.net |                | US          | 55667788
(2 rows)
```

Problem:
I've no guaranteed way to identify a unique person

```
create table people(
  first_name, last_name , birth_date      , email           , twitter_handle,  tax_country, tax_id
) as values
 ('Franck'  ,   'Pachot',  date '1971-02-08', 'me@pachot.net', '@FranckPachot', 'CH'      , 12345689)
,('John  '  ,   'Doe'   ,  date '1970-01-01', 'john@dow.net' ,  null          , 'US'      , 11223344)
,('Jane  '  ,   'Doe'   ,  date '1970-01-01', 'jane@dow.net' ,  null          , 'US'      , 55667788)
;
\d+ people
select * from people;
select * from people where last_name='Pachot';
select * from people where last_name='Doe';
```

Queries should have a key, and we can provide many keys

```
yb_demo_northwind=# alter table people add constraint unique_twitter_handle_unique unique (twitter_handle);
ALTER TABLE
yb_demo_northwind=# alter table people add constraint unique_tax_key unique (tax_country, tax_id);
ALTER TABLE
yb_demo_northwind=# alter table people add constraint unique_email unique (email);
ALTER TABLE
yb_demo_northwind=# alter table people alter column email set not null;
ALTER TABLE
yb_demo_northwind=# \d people
                Table "public.people"
     Column     |  Type   | Collation | Nullable | Default
----------------+---------+-----------+----------+---------
 first_name     | text    |           |          |
 last_name      | text    |           |          |
 birth_date     | date    |           |          |
 email          | text    |           | not null |
 twitter_handle | text    |           |          |
 tax_country    | text    |           |          |
 tax_id         | integer |           |          |
Indexes:
    "unique_email" UNIQUE CONSTRAINT, lsm (email HASH)
    "unique_tax_key" UNIQUE CONSTRAINT, lsm (tax_country HASH, tax_id ASC)
    "unique_twitter_handle_unique" UNIQUE CONSTRAINT, lsm (twitter_handle HASH)
```

Keys on non nullable column can identify any row in the table

You can have many keys

Or unique concatenation of columns

A key is a unique column

```
alter table people add constraint unique_twitter_handle_unique unique (twitter_handle);
alter table people add constraint unique_tax_key unique (tax_country, tax_id);
alter table people add constraint unique_email unique (email);
alter table people alter column email set not null;
\d+ people
```

One of the key can be the primary key

```
 tax_id          | integer |           |          |
Indexes:
    "unique_email" UNIQUE CONSTRAINT, lsm (email HASH)
    "unique_tax_key" UNIQUE CONSTRAINT, lsm (tax_country HASH, tax_id ASC)
    "unique_twitter_handle_unique" UNIQUE CONSTRAINT, lsm (twitter_handle HASH)
```

Primary key is about physical organization of rows

Some database do not **require** a primary key

- PostgreSQL: heap tables. Primary key is just a not-null unique index
- Oracle: heap tables (except Index Organized Tables)
- DB2: heap tables. Primary key is optional

Some databases **cluster** data on the primary key (B*Tree or LSM tree)

- YugabyteDB: without primary key an internal uuid is generated
- MySQL (InnoDB): the first unique index is used to cluster data
- SQL Server: clustered index (except heap tables)

A key used for organization should be immutable

```
 tax_id          | integer |           |          |
Indexes:
    "unique_email" UNIQUE CONSTRAINT, lsm (email HASH)
    "unique_tax_key" UNIQUE CONSTRAINT, lsm (tax_country HASH, tax_id ASC)
    "unique_twitter_handle_unique" UNIQUE CONSTRAINT, lsm (twitter_handle HASH)
```

A key that is used for **physical** organization

- Should not be updated (cons: index maintenance, fragmentation)

A key that is used for **logical** organization (pointer)

- Must not be updated (cons: complexity of cascading changes)

When it is about physical or logical organization, we **can** add a non-business key

- Name: (Surrogate|Artificial|Technical) key

Adding a surrogate key to be the primary key

```
yb_demo_northwind=# alter table people add column people_id int;
ALTER TABLE
yb_demo_northwind=# update people t set people_id=n.id from
yb_demo_northwind-# (select row_number()over() id,* from people ) n
yb_demo_northwind-# where (t.tax_country,t.tax_id)=(n.tax_country,n.tax_id);
UPDATE 3
yb_demo_northwind=# select * from people;
 first_name | last_name | birth_date |     email     | twitter_handle | tax_country |  tax_id  | people_id
------------+-----------+------------+---------------+----------------+-------------+----------+-----------
 John       | Doe       | 1970-01-01 | john@dow.net  |                | US          | 11223344 |         1
 Franck     | Pachot    | 1971-02-08 | me@pachot.net | @FranckPachot  | CH          | 12345689 |         2
 Jane       | Doe       | 1970-01-01 | jane@dow.net  |                | US          | 55667788 |         3
(3 rows)

yb_demo_northwind=# alter table people add constraint people_pk  primary key (people_id);
ALTER TABLE
yb_demo_northwind=#
```

```
alter table people add column people_id int;
update people t set people_id=n.id from
(select row_number()over() id,* from people ) n
where (t.tax_country,t.tax_id)=(n.tax_country,n.tax_id);
select * from people;
alter table people add constraint people_pk  primary key (people_id);
```

Having a generated key for the surrogate

Generated != Surrogate

When exposed to the user (screen, bills, url) it becomes a natural key

- not immutable. e.g: you assign customer id 666 to your best customer and he wants to change
- leaks information on the internals

```
 tax_id         | integer |           |          |                                  | plain    |
 people_id      | integer |           | not null | nextval('people_id'::regclass) | plain    |
Indexes:
    "people_pk" PRIMARY KEY, lsm (people_id HASH)
    "unique_email" UNIQUE CONSTRAINT, lsm (email HASH)
```

```
create sequence people_id start with 4;
alter table people alter column people_id set default( nextval('people_id') );
\d+ people;
```

Performance is the same: all key are unique index

```
yb_demo_northwind=# explain select * from people where people_id=1;
                               QUERY PLAN
------------------------------------------------------------------------
 Index Scan using people_pk on people  (cost=0.00..4.11 rows=1 width=172)
   Index Cond: (people_id = 1)
(2 rows)

yb_demo_northwind=# explain select * from people where email='me@pachot.net';
                                QUERY PLAN
---------------------------------------------------------------------------
 Index Scan using unique_email on people  (cost=0.00..4.12 rows=1 width=172)
   Index Cond: (email = 'me@pachot.net'::text)
(2 rows)
```

```
explain select * from people where people_id=1;
explain select * from people where email='me@pachot.net';
insert into people (email) select format('spam%s@gmail.com',n) from generate_series(1,1000) n;
explain analyze select * from people where people_id=1;
explain analyze select * from people where twitter_handle='@FranckPachot';
\d+ people;
```

Immutable?

There is no problem to update the key, primary or not.
As long as nothing references it (foreign key)

```
Indexes:
    "people_pk" PRIMARY KEY, lsm (people_id HASH)
    "unique_email" UNIQUE CONSTRAINT, lsm (email HASH)
    "unique_tax_key" UNIQUE CONSTRAINT, lsm (tax_country HASH, tax_id ASC)
    "unique_twitter_handle_unique" UNIQUE CONSTRAINT, lsm (twitter_handle HASH)

yb_demo_northwind=# update people set people_id=0 where email='me@pachot.net';
UPDATE 1
yb_demo_northwind=#
```

```
\d+ people;
update people set people_id=0 where email='me@pachot.net';
```

Foreign key can reference any key - primary or just unique

```
yb_demo_northwind=# create table tax_reports (tax_country text, tax_id int, year int, amount numeric,
yb_demo_northwind(# foreign key (tax_country,tax_id)
yb_demo_northwind(# references people(tax_country,tax_id)
yb_demo_northwind(# );
insert into tax_reports values ('CH',12345689,2021,999.99);
CREATE TABLE
yb_demo_northwind=# insert into tax_reports values ('CH',12345689,2021,999.99);
INSERT 0 1
yb_demo_northwind=# \d tax_reports
              Table "public.tax_reports"
   Column    |  Type   | Collation | Nullable | Default
-------------+---------+-----------+----------+---------
 tax_country | text    |           |          |
 tax_id      | integer |           |          |
 year        | integer |           |          |
 amount      | numeric |           |          |
Foreign-key constraints:
    "tax_reports_tax_country_fkey" FOREIGN KEY (tax_country, tax_id) REFERENCES people(tax_country, tax_id)
```

```
create table tax_reports (tax_country text, tax_id int, year int, amount numeric,
foreign key (tax_country,tax_id)
references people(tax_country,tax_id)
);
insert into tax_reports values ('CH',12345689,2021,999.99);
```

Primary key is just the default if you don't mention columns (because there is only one primary key)

Update the parent key

We have seen lot of agility but here we have a problem

```
yb_demo_northwind=# update people set tax_id=666666 where email='me@pachot.net';
ERROR:  update or delete on table "people" violates foreign key constraint "tax_reports_tax_country_fkey" on table "tax_reports"
DETAIL:  Key (tax_country, tax_id)=(CH, 12345689) is still referenced from table "tax_reports".
yb_demo_northwind=# update tax_reports set tax_id=666666 where tax_id=12345689;
ERROR:  insert or update on table "tax_reports" violates foreign key constraint "tax_reports_tax_country_fkey"
DETAIL:  Key (tax_country, tax_id)=(CH, 666666) is not present in table "people".
```

```
update people set tax_id=666666 where email='me@pachot.net';
update tax_reports set tax_id=666666 where tax_id=12345689;
```

Foreign keys should reference immutable keys

```
yb_demo_northwind=# alter table tax_reports add column people_id int references people;


ALTER TABLE
yb_demo_northwind=# update tax_reports set people_id=0 where tax_id=12345689;
UPDATE 1
yb_demo_northwind=# \d tax_reports
              Table "public.tax_reports"
   Column    |  Type   | Collation | Nullable | Default
-------------+---------+-----------+----------+---------
 tax_country | text    |           |          |
 tax_id      | integer |           |          |
 year        | integer |           |          |
 amount      | numeric |           |          |
 people_id   | integer |           |          |
Foreign-key constraints:
    "tax_reports_people_id_fkey" FOREIGN KEY (people_id) REFERENCES people(people_id)
    "tax_reports_tax_country_fkey" FOREIGN KEY (tax_country, tax_id) REFERENCES people(tax_country, tax_id)
```

```
alter table tax_reports add column people_id int references people;
update tax_reports set people_id=0 where tax_id=12345689;
\d tax_reports
```

A primary key can include the surrogate key from the parent

```
yb_demo_northwind=# alter table tax_reports drop constraint tax_reports_tax_country_fkey;
ALTER TABLE
yb_demo_northwind=# alter table tax_reports alter column people_id set not null;
ALTER TABLE
yb_demo_northwind=# alter table tax_reports add primary key (people_id,year);
ALTER TABLE
yb_demo_northwind=# \d tax_reports
              Table "public.tax_reports"
   Column    |  Type   | Collation | Nullable | Default
-------------+---------+-----------+----------+---------
 tax_country | text    |           |          |
 tax_id      | integer |           |          |
 year        | integer |           | not null |
 amount      | numeric |           |          |
 people_id   | integer |           | not null |
Indexes:
    "tax_reports_pkey" PRIMARY KEY, lsm (people_id HASH, year ASC)
Foreign-key constraints:
    "tax_reports_people_id_fkey" FOREIGN KEY (people_id) REFERENCES people(people_id)
```

Typical many-to-one
Do you need another surrogate here?

```
alter table tax_reports drop constraint tax_reports_tax_country_fkey;
alter table tax_reports alter column people_id set not null;
alter table tax_reports add primary key (people_id,year);
\d tax_reports
```

In short...

SQL may not need a key... but JPA requires an @Id
Natural keys can change 👉 it is easier to generate a key
Exposed values may be updated 👉 it is safer with surrogate key
Referenced keys (by FK) should be immutable
What about composite keys?

Composite keys

As we can always add a surrogate key,
do we need to support multi-column keys?

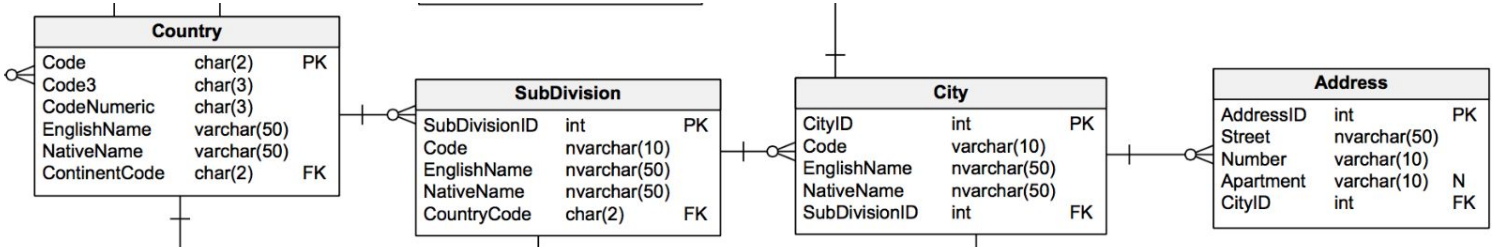
Primary keys on association tables will be composite

https://vladmihalcea.com/the-best-way-to-map-a-many-to-many-association-with-extra-columns-when-using-jpa-and-hibernate

many-to-many: the concatenation of the foreign keys is the primary key

No need to add a surrogate key (and another index to maintain) that will never be used

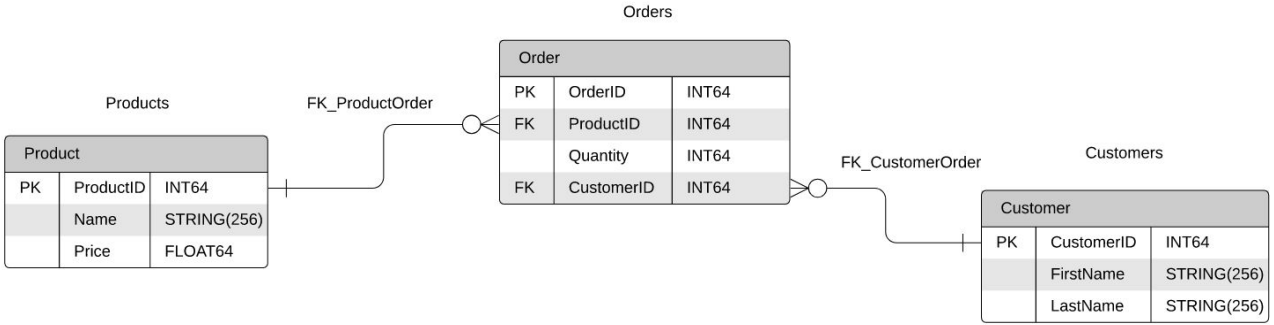
When replacing all composite keys by surrogate:



Compare this with (countryID,SubDivisionID,cityID,addressID) as PK of ADDRESS

Additional index to maintain (for the surrogate key)
Additional joins on queries (users query by natural ID)
Reduces partitioning possibilities (how to partition ADDRESS by country?)
Hides optimizer statistics on business values (predicate selectivity)

Some entities look like association tables but are not

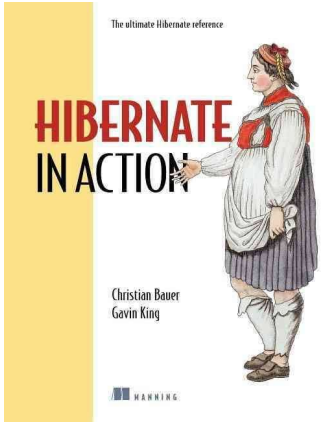


Here we need a key for the association between products and customers, it is a business entity, and the concatenation of foreign keys is not unique -> need a key (here generated but not surrogate)

JPA

Composite key mapping

“Legacy schemas and composite keys”



“Mapping from legacy databases”

JPA

2.4 Primary Keys and Entity Identity

Every entity must have a primary key.

The primary key must be defined on the entity class that is the root of the entity hierarchy or on a mapped superclass that is a (direct or indirect) superclass of all entity classes in the entity hierarchy. The primary key must be defined exactly once in an entity hierarchy.

A primary key corresponds to one or more fields or properties (“attributes”) of the entity class.

- A simple (i.e., non-composite) primary key must correspond to a single persistent field or property of the entity class. The `Id` annotation or `id` XML element must be used to denote a simple primary key. See Section 11.1.18.

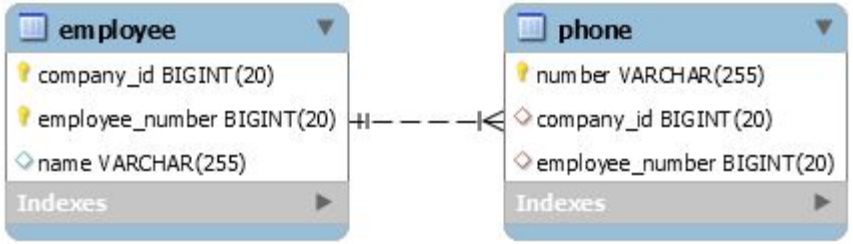
- A composite primary key must correspond to either a single persistent field or property or to a set of such fields or properties as described below. A primary key class must be defined to represent a composite primary key. Composite primary keys typically arise when mapping from legacy databases when the database key is comprised of several columns. The `EmbeddedId` or `IdClass` annotation is used to denote a composite primary key. See Sections 11.1.15 and 11.1.19.

Problem with the model or with the tool?

“many legacy schemas use (natural) composite key”
This is too vague. The real problem is:
JPA needs an identifier with hashcode() and equals()
Natural key is a candidate, but there’s rarely one
Surrogate key is a candidate: generate an object Id and then put data

What about keys composed of surrogate keys?
(aggregations, compositions, associations)

Composite keys can be mapped in JPA



The best way to map a Composite Key with JPA and Hibernate:
https://vladmihalcea.com/the-best-way-to-map-a-composite-primary-key-with-jpa-and-hibernate/

Solution: composite key as Embeddable

```
@Embeddable
public class EmployeeId implements Serializable {
      @ManyToOne
      @JoinColumn(name = "company_id")
      private Company company; @Column(name = "employee_number")
      private Long employeeNumber;   public EmployeeId() {
      }
      public EmployeeId(Company company, Long employeeId) {
      this.company = company;
      this.employeeNumber = employeeId;
      }
      public Company getCompany() { return company; }
      public Long getEmployeeNumber() { return employeeNumber; }
      @Override
      public boolean equals(Object o) {
      if (this == o) return true;
      if (!(o instanceof EmployeeId)) return false;
      EmployeeId that = (EmployeeId) o;
      return Objects.equals(getCompany(), that.getCompany()) &&
            Objects.equals(getEmployeeNumber(), that.getEmployeeNumber());
      }
      @Override
      public int hashCode() { return Objects.hash(getCompany(), getEmployeeNumber()); }
```

- The primary key class must be serializable.
- The primary key class must define `equals` and `hashCode` methods. The semantics of value equality for these methods must be consistent with the database equality for the database types to which the key is mapped.

Solution: composite key as Embeddable

```
@Entity(name = "Employee")
@Table(name = "employee")
public class Employee {

      @EmbeddedId
      private EmployeeId id;

      private String name;
      public EmployeeId getId() {
      return id;
      }
      public void setId(EmployeeId id) {
      this.id = id;
      }
      public String getName() {
      return name;
      }
      public void setName(String name) {
      this.name = name;
      }
}
```

If mapped by (foreign key):

```
@ManyToOne
@JoinColumns (foreignKey = @ForeignKey(name = "FK_LAPTOP_EMP"),
 value = {
  @JoinColumn(name="company_id",referencedColumnName = "company_id"),
  @JoinColumn(name="employee_number",referencedColumnName="employee_number")
 })
private Employee laptopOwner;
```

Generated keys

Sequence or UUID

Generated keys: UUID or sequence?

Generated <> Surrogate

You can generate a key that becomes a natural key

Example: Customer ID is not immutable because exposed to customers

But a surrogate key is generated

Generated key requires unique values

- With a single point of truth (problem: scale)
- or With a large random generator (problem: size)

Sequences

Do not try to have no-gap sequences!

If you need it, it must be after-commit (on query or batch updated)

Sequences

Sequences can scale with cache (application or DB)

https://dev.to/yugabyte/uuid-or-cached-sequences-42fi

➖ Security: May leak some information about your data

➕ smaller and faster than a UUID

GenerationType.SEQUENCE

```
yugabyte=# select * from pg_sequences;
 schemaname |    sequencename     | sequenceowner | data_type | start_value | min_value |      max_value      | increment_by | cycle | cache_size | last_value
------------+---------------------+---------------+-----------+-------------+-----------+---------------------+--------------+-------+------------+------------
 public     | hibernate_sequence | yugabyte      | bigint    |           1 |         1 | 9223372036854775807 |            1 | f     |        100 |
 public     | my_sequence        | yugabyte      | bigint    |          42 |         1 | 9223372036854775807 |          666 | f     |        100 |
(3 rows)

yugabyte=#
```

SequenceGenerator allocationSize

YugabyteDB default db-side cache

```
@SequenceGenerator(name="myseq", initialValue=42, allocationSize=666)

public class Company {
      @Id
      @GeneratedValue(strategy=GenerationType.SEQUENCE,generator="myseq")
      private Long id;  private String name;    public Long getId() {
      return id;
      }
```

UUID

UUID will always be larger than a sequence (16 bytes)

➖ Requires more CPU than a cached sequence

➕ does not leak information (GDPR)

➕ still unique when merging databases

🙏 Do not store UUID as VARCHAR(32)

Thank You

Join us on Slack:
www.yugabyte.com/slack
Star us on GitHub:
github.com/yugabyte/yugabyte-db

fpachot@yugabyte.com
dev.to/FranckPachot
@FranckPachot

Core message:

- Surrogate key is almost always needed
- We cannot ignore composite keys in a relational database
- Think about the business meaning and then performance/agility for your RDBMS

yugabyteDB