Hibernate is core Java framework and that can connect with the database layers.

Hibernate is an ORM tool (Object Relational Mapping).

JPA is the Java specification and not the implementation. Hibernate is an implementation of JPA and uses common standards of Java Persistence API.

Ibatties, toplink are the other platform that uses the ORM.

Annotation in Hibernate

@Enitity: mark a class as Entity

@Table: Create the table in the database.

@Id: Create the Primary Key in the database table

@GeneratedValue: It will automatically generare the primary key value.

@Column: Give the column to the Entity attribute.

**@Transient**: not to save in the database.

**@Temporal**: add above the date field tell that this attribute need to store the date value as date, time, timestamp formate.

**@Lob:** tells that is the large object not the simple object.

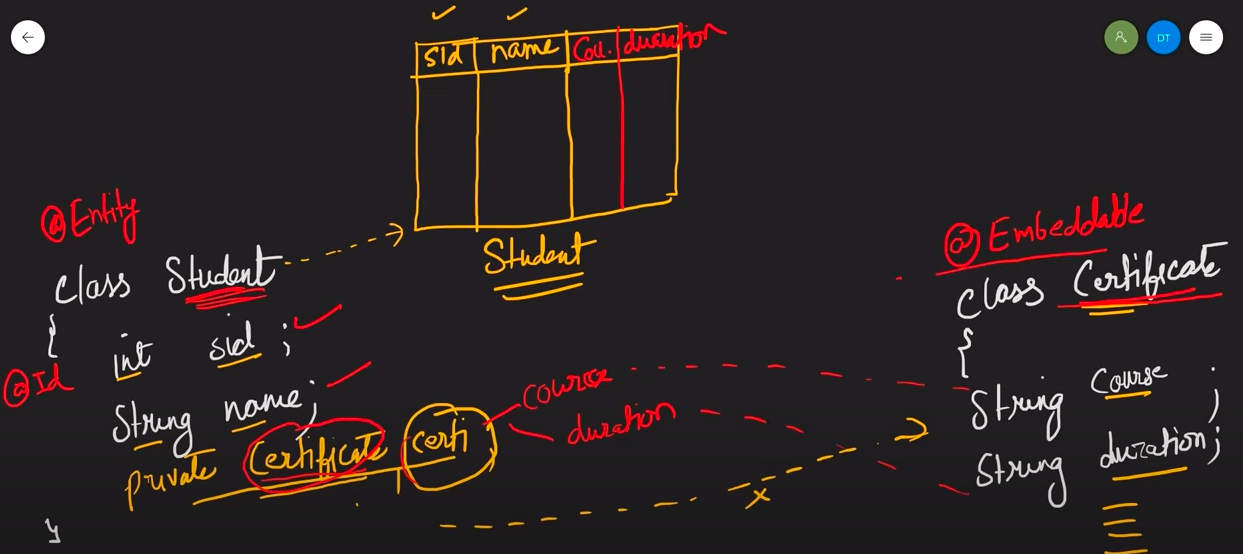
@OneToOne, @OneToMany, @ManyToOne, @ManyToMany @JoinColumn

@Embeddable:

We need to embed all the properties to the Entity in Main Entity.

Example: We have to inject the Certificate class to the Student then We need to use the Embeddable Annotation above the class.

So Student Class have four properties in the Enitity: sid, name, **course, designation**.



**FetchType:**

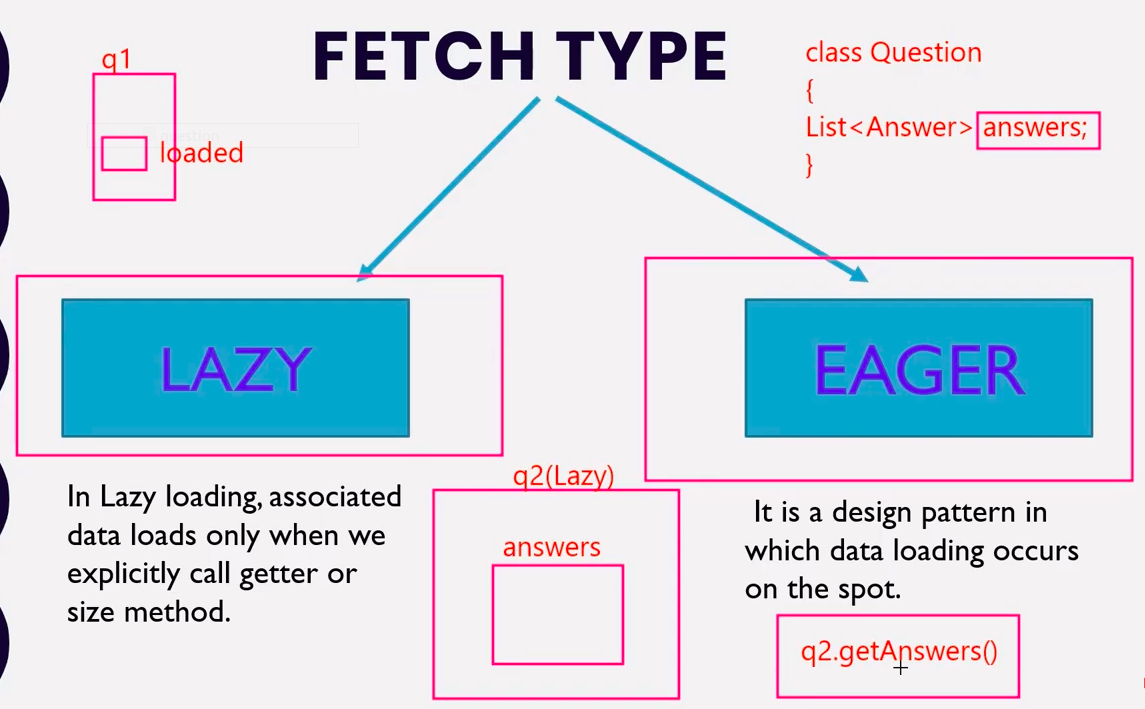
* Lazy: on time when we call the function.
* Eager: before time it load the data before call the function.

OneToMany: LAZY

ManyToOne: EAGER

ManyToMany: LAZY

OneToOne: EAGER



**Hibernate Lifecycle States: (TPDR)**

**Transient State:** when we create object and initilaized value in that then that object goes into this state.

**Persistance State:** when the object is store and save in the session that time the object comes into the Persistance state. And data is store inside the database.

**Detached State:** When the session is closed then object is delete from the session and comes in the detached state.

**Removed State:** when user delete the data from the database and close the session that time it comes in that phase.

**HQL: (Hibernate Query Language)**

HQL is the database independent so we can create the query for any database.

**HQL Query:** from Entity where EntityVariable = ‘[Value]’

from Entity where EntityValuable = :X (get the value form the user dynamically)

Ex: SELECT: from Student s where s.name = ‘Pritesh’

DELETE: delete from Student s where s.name = ‘Pritesh’

UPDATE: update Student s SET s.name=’PriteshUpdate’ where s.name=’Pritesh’

INNERJOIN:

select q.questionId, q.question, a.answerId, a.answer from Question as q INNER JOIN q.answer as a;

Select q.\*, a.\* from Question as q inner join Answer as a on q.id = a.qid where q.type=’java’ order by q.id asc;

group by q.type;

Limit 10 offset 2

**Question:**

1. **Hibernate Vs JDBC:**

Hibernate is the ORM tool. In hibernate we are using the HQL.

In Jdbc we are using the SQL.

Hibernate if we change the database then only dialect of the database need to change in the application properties file.

Jdbc use the sql query so we need to change the query as per the database we are using.

Transaction and caching are inbuilt implemented on the Hibernet.

Jdbc we explicitly write a code for the transaction.

Hibernet handle all the checked exception.

In sql we need to write a code in the try catch block.

1. **Session Factory vs Session:**

Session Factory provide the object of the session.

In threading environment Session Factory is used by the multiple threads at the same time.

Where only single session can perform the activity at the same time in the Session Factory.

SessionFactory is the heavy weight.

Session is the light weight.

Can we create the multiple sessionFactory? And where we use?

Yes, we can create the multiple session Factory. We can create then multiple session factory where the multiple database are used.

For Each database we are create the one session factory object.

1. **Equivalent object of sesion in JPA.**

Entity Manager is the interface use in the JPA equivalent to the session.

Session: save(), merge(), delete(), detach(), close(), clear(), get(), load()

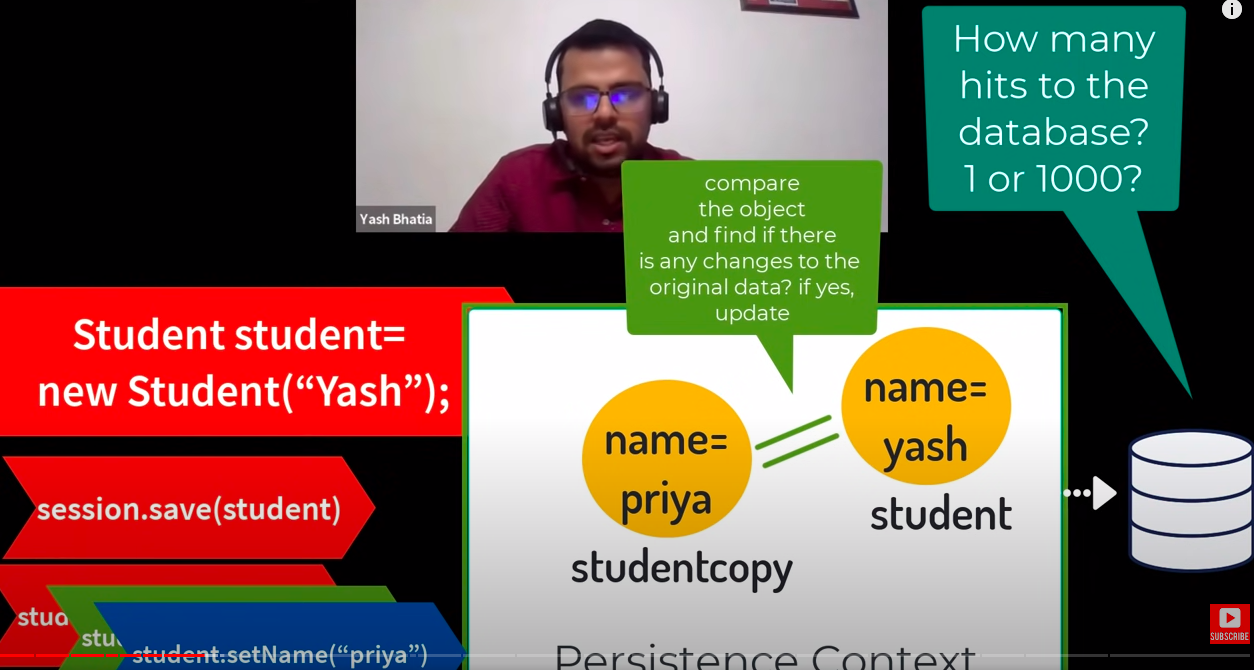
EntityManager: Persist(), merge(), remove(), evit(), close(), clear(), find(), getReference()

1. **Persistence Context and what we use in hibernet:**

Persistence context is **created when ever we create a session object**. **Maintan the state of the object** in the persistence context block. Also have the session object that will coordinate with the database.

1. **What if we can get the entity object from the database and change the name of student in 1000 time and then at the end we can write commit?**

**Answer - only hit the query last time.**



Answer: when we fetch the object from the database it store in the persistence context.

Then we perform some name updation 1000 time it will create the copy of the object store in the persistence context and update that 1000 time value to the copy object(studentcopy).

After session.commit() both the object in the persistence context are compared if their is the any change in the studentcopy then it will update in the original object and database.

It is also called as a first level of caching.

Persistence Context(Session) is the first level cache.

1. **Get() vs Load() in Hibernate:**

Get is from JPA.

Load() is from hibernate.

Ex: suppose we need to find the user with the userId 1.

In get() method: session.get(Student.class, 1);

It will directly call the database when we call the get method.

In Load() method:

Student stu = session.load(Student.class, 1); // that is the proxy object (No real object))

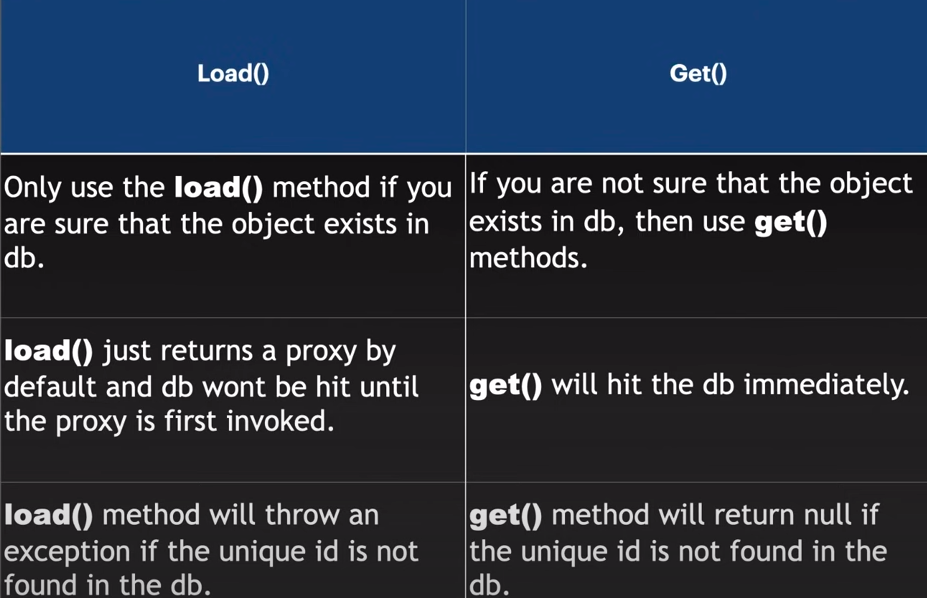
System.out.println(**stu**.getName()); // here stu is the real object

It will not call the database when we call it call the database when we using the attributes of that object. When we call “stu.getName()” that time the database will hit and get the record from the database.

It no data found then Exception are throw like,

get(): NULL POINTER EXCEPTION

load(): No Row with the given identifiers found.



1. **Transient State in the hibernation/JPA:**

If we create the object with out opening the session before then that object will called as a transient starte.

Ex: Student stu = new Student(“Pritesh”) then stu is call the **Transient state.** Because that objet is not associate with the session.

If we create the object of session and call the save method then that state is called **persistence state**.

Ex: Session session = sessionFactory.openSession();

session.save(stu);

Persistence context always have the identifier (PK) for communicate with the database.

1. **Detached Starte in hibernate/JPA:**

If we can close the **session.close()** then all the object from the persistence state is come to the detached state. Also done with the **session.evict()** also go to the detached state without session.close().

1. **How to move the transient state to persistence state.**

We have the get, load, save and update method to move the transient state to persistence state.

1. **What is connection pooling.**

Connection pooling is noting but the bunch of the connections that is reusable in the threading.

Ex: thread1 can call the query of the student using connection pool connecation1.

thread2 can access the same database using the connection pool connecction2.

In Spring boot we have the provider to implement the conncetion pooling.

* DBCP (Provided by the apache)
* Tomcat
* C3PO
* HikariCP (use Default by spring boot)

What interface are used to manage the connection pooling: DataSource

1. **@GeneratedValue and stretorgies:**

There is the four stretorgis: Identity, Sequence Table, Auto

Auto: default implementation define by the hibernate. (Default is the sequence)

Identity: Is managed by the datatype. Giving the primary generation capacity to database and database generate key and that use by hibernate.

Sequence: use when we need to start key generation fro 2 and the next key should be 4, 6,8 and so on at that time we use sequence.

Table: that can manage one table for managing the all the entity primary key and maintain all the keys of the entity in the one table.

Example of table: If we have two table student and course when we insert value in the student table first then the pk is generated as 1 and add in the seq\_table. When we create the course entries tha time it check from the seq\_table table 1 is already there then the next primary key is 2 so add 2 as a primary key for the course table.

1. **@NamedQuery:**

Nameed query is added when we have need the query frequvantly in user.

Nameed Query is added above the Entity.

Ex:

@NameedQuery(name=”myapp.allStudent”, query=”from Student”)

@Entity

@Table

Class Student {

}

Use in logic:

List<Student> studentList = session.getNamedQuery(“myapp.allStudent”);

Query<Student> studentList = session.createNamedQuery(“myapp.allStudent”, Student.class);

1. **Criteria API:**

We can witer the more complex query using the criteria API.

That is basically use for the joins.

Example:

Criteria cr = session.createCriteria(Employee.class);

cr.add(Restrictions.eq("salary", 2000));

List results = cr.list();

**Concurrency in the database:** One or more thead/proecss can access the same database data at the same time then that is called concurrent in database.

Main problems in using Concurrency: Updates will be lost, Inconsistent retrievals

For Solution we need the locking stretorgy:

Two type of lock:

Shared lock: only available for the read the data.

Exclusive lock: available for read and write the data.

In spring boot we have Annotation to achieve the locking the transaction.

@Transactional

public void processOrder(Long orderId) {

Order order = orderRepository.findById(orderId);

entityManager.lock(order, LockModeType.PESSIMISTIC\_WRITE);

// perform some operations on the order

orderRepository.save(order);

}

### or other alternative

@Repository

public interface MyRepository extends JpaRepository<MyEntity, Long> {

@Lock(LockModeType.PESSIMISTIC\_WRITE)

MyEntity findById(Long id);

}

**Database Normalization:**

1st Normal Form:

No MualtiValue attribute, only single value

Ex: Student Course: stu1 -> java, c // not allow comma value in the course field. Only single value are allowed.

2nd Normal Form:

No Partial Dependancy, only full dependency are allowed.

EX: AB -> C , if AB are the candidate key and c is dependent on that then only A is not allow to dependant on C and only B is also not. Require both AB to depencacy. (Full Dependancy)

3rd Normal Form:

No Transitive Dependancy.

Non unique table can not point to another nonuniqe table.

Ex: A->B->C A has PK, B is non prime (Not have unique value), c(Also not have unique value).

BCNF:

LHS must be a candidate Key or Super Key.

Ex: A->B //A must have the candidate key or super key as B.

4th Nomal Form:

No MultiValued Dependacy.

Ex: Pritesh have -> phno1 phno2 phno3 -> email1 email2 email3

Create two table one for the phone number and one for the email.

5th Normal Form:

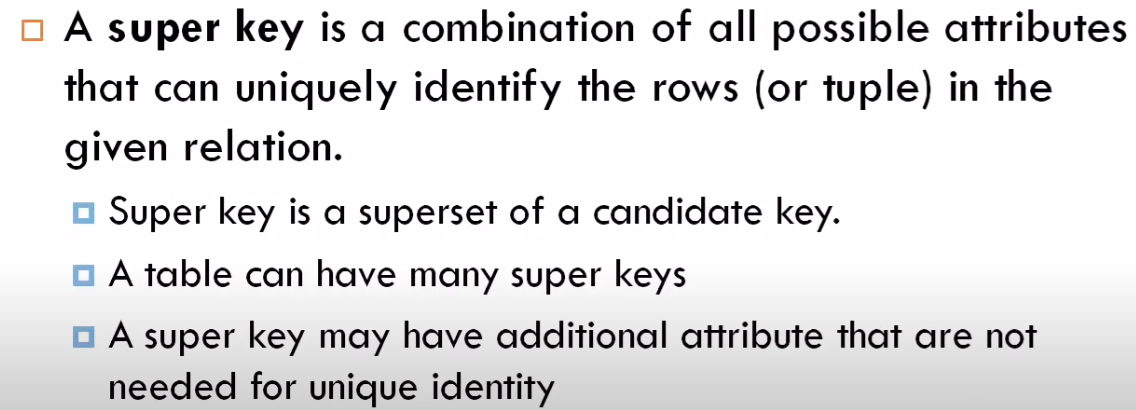
Loss less decomposission.

A have two child B, C. We split A in two table B and C. When we have merge in one then there will be extra columns.

We need to create common value as a candidate key to avoid that problem.

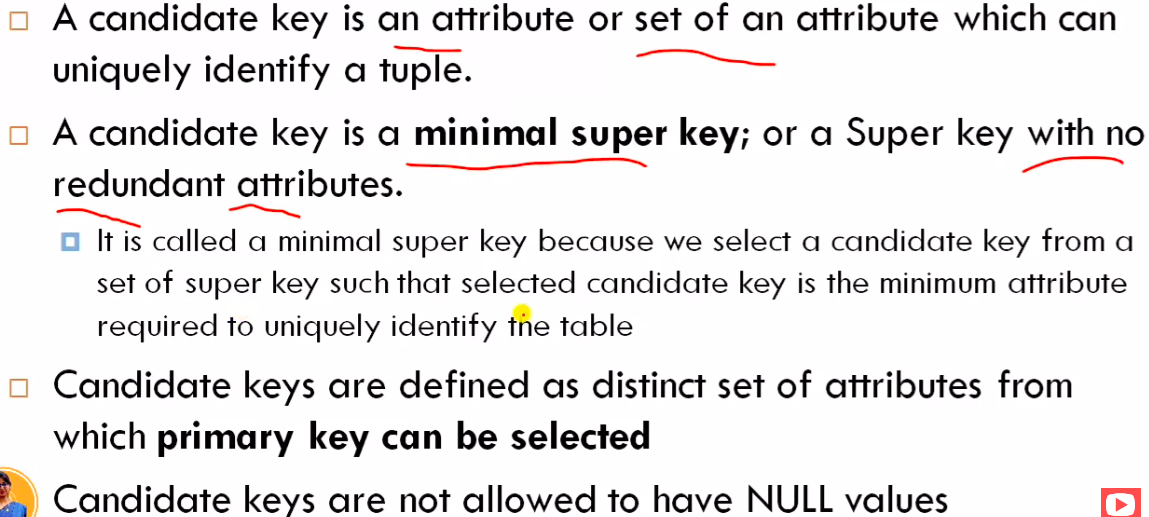
**Database Keys - (SCPAFC)**

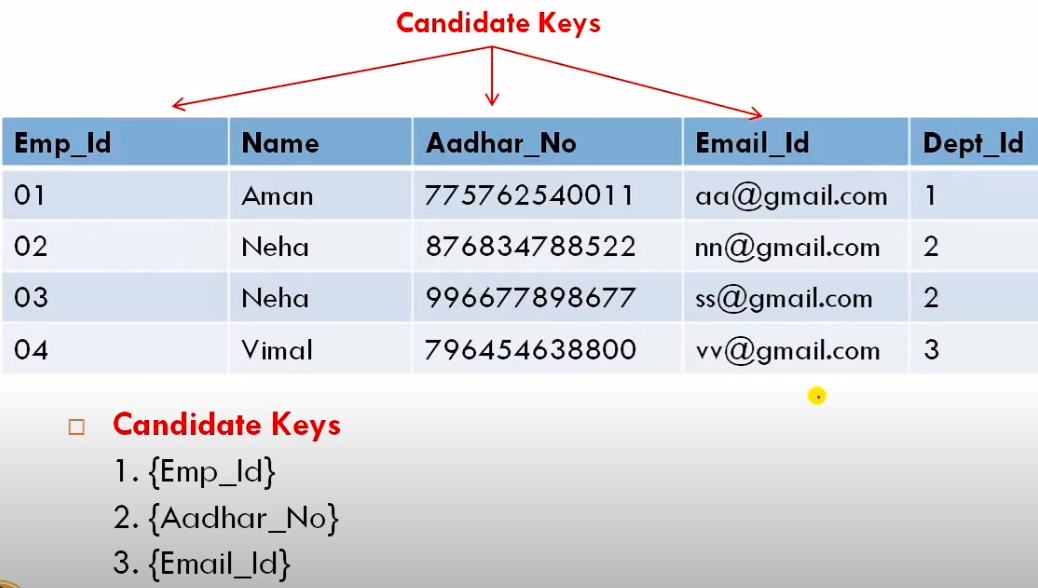
Super Key





Candidate Key





That three is the Key from the super key that have the unique value and not repeat in the attribute list.

Primary Key

We can select one key from the Candidate key that must have not null value and uniquely identify, not update one it assign, only one primary key in the table.

{Emp\_id} we select as a Primary Key.

Alternate Key

All the Key in the Candidate Key other than primary key is called the Alternate key.

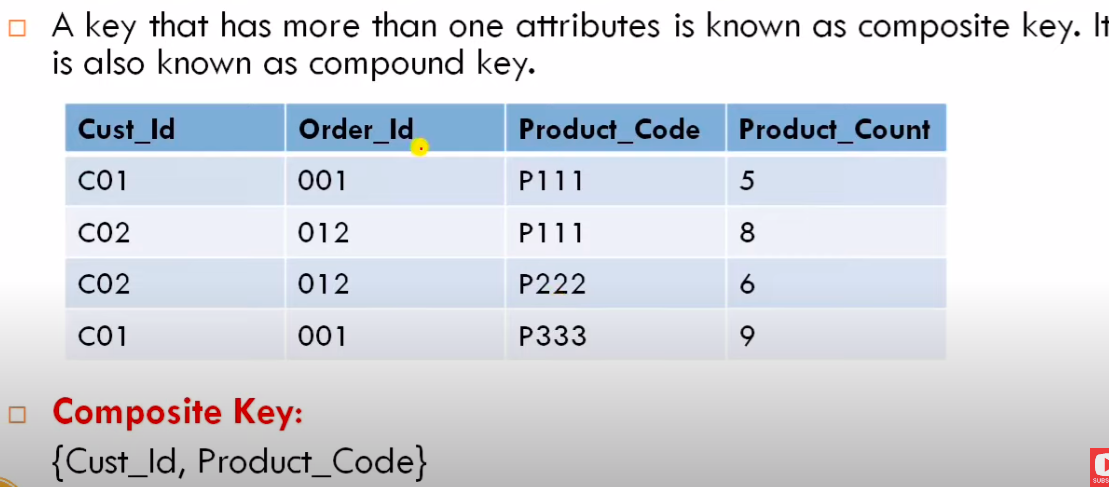
Here, {Aadhar\_No} and {Email\_Id} is the Alternate Key

Foreige Key

Here the {Dept\_Id} as the Foreige of the Department table.

Composite | Compound Key

More than one attribute combination key is called the Composite Key.



There is no unique attribute for the primary key so we can merge two column and create the Composite Key.

**SQL Querys:**

<https://www.edureka.co/blog/interview-questions/sql-query-interview-questions>

2nd highest salary:

select max(salary) from employee where salary < (select max(salary) from employee where salary>=0);

Nth highest salary: (6th highest salary)

SELECT \* FROM employee where salary!=0 order by salary desc limit 1 offset 6;

Remove duplicate records of email

select email from Person group by email having count(email) > 1;

SELECT \* FROM Customers

WHERE Country IN ('Germany', 'France', 'UK');

SELECT \* FROM Customers

WHERE Country NOT IN ('Germany', 'France', 'UK');

SELECT column\_name(s)

FROM table\_name

WHERE condition

GROUP BY column\_name(s)

HAVING condition

ORDER BY column\_name(s);

SELECT column\_name(s)

FROM table\_name

WHERE **EXISTS**

(SELECT column\_name FROM table\_name WHERE condition);

SELECT column\_name(s)

FROM table\_name

WHERE column\_name operator **ALL|ANY**

(SELECT column\_name

FROM table\_name

WHERE condition);

SELECT CustomerName, City, Country

FROM Customers

ORDER BY

(CASE

WHEN City IS NULL THEN Country

ELSE City

END);

CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode nvarchar(10)

AS

SELECT \* FROM Customers WHERE City = @City AND PostalCode = @PostalCode

GO;

EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1 1DP';