

The ERD diagram of Cinema Database.

**Steps of creation.**

**Step 1.** First of all I created ‘Cinema’ table which have its own id, name, address, phone. Then 2 table emanate: customer and employee. Customer have his own id, first name, last name, phone number and age. And employee have own id, first name, last name, gender and phone.

**Step 2.** The client can make a reservation. The customer order table links to the ‘Booking’ table. The “booking” table has data such as booking time and all information about movie, and employee who accepted the order through foreign keys and finally getting total price.

**Step 3.** If customer came to cinema and wants buy ticket immediately, he also have information about movie and employee, and after payment – getting ticket through payment\_id keys.

**Step 4.** About movie table, here we have information about his name, date, genre, start and end times.

**Step 5.** Now the “ticket” table. It has its own id, buying date, employee information, customer\_id, payment\_id, seat\_id.

**Step 6.** ‘Seat table’ have hall, row and number of seat, also in which cinema.

**Step 7.** ‘Vip’ table talking about vip seats, and his price.

**Step 8.** And last remaining table is ‘payment’. In this table we store information about final total prices.

1. **Using the join to query the customers who bought the most expensive tickets:**

select fname, lname, final\_price

from customer

inner join booking

on customer.customer\_id=booking.customer\_id

inner join payment

on payment.payment\_id=booking.payment\_id

where payment.final\_price>5000;

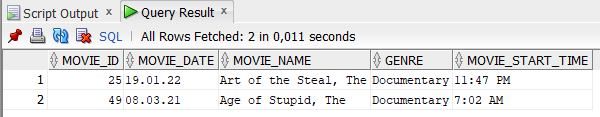
Изображение выглядит как стол

Автоматически созданное описание

1. **Movie which titles starts with letter A:**

select\*from movie

where movie\_name like 'A%';



1. **Movies released in 2020:**

select buying\_date, movie\_name

from ticket

inner join movie

on movie.movie\_id=ticket.movie\_id

where buying\_date between '01-01-2020' and '01-01-2021';

Изображение выглядит как текст

Автоматически созданное описание

1. **Child rate (customers under 18 years of age):**

select fname, lname, age

from customer

where age < 18;

Изображение выглядит как стол

Автоматически созданное описание

1. **Male workers:**

select fname, lname

from employee

where gender = 'Male'

Изображение выглядит как текст

Автоматически созданное описание

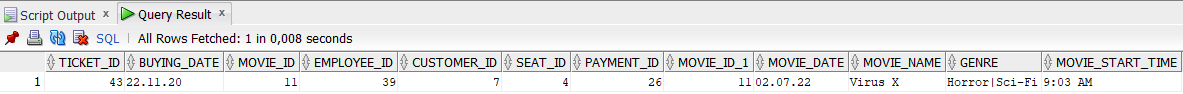
1. **All data where client id = 7:**

select \* from ticket

inner join movie

on movie.movie\_id=ticket.movie\_id

where customer\_id = 7;



1. **3 minimum prices of the vip seat:**

select seat\_id, price

from vip

order by price asc

fetch first 3 rows only;

Изображение выглядит как стол

Автоматически созданное описание

1. **Most popular genres in descending order:**

select count(\*) as quantity, genre

from movie

group by genre

order by quantity desc;

Изображение выглядит как текст

Автоматически созданное описание

1. **Customer, employee first names, final price where booking date is 7th April of 2021:**

select customer.fname c\_fname, employee.fname e\_fname, final\_price, booking\_time

from customer

inner join booking

on customer.customer\_id=booking.customer\_id

inner join employee

on employee.employee\_id=booking.employee\_id

inner join payment

on payment.payment\_id=booking.payment\_id

where booking\_time = '07-04-2021';

Изображение выглядит как стол

Автоматически созданное описание

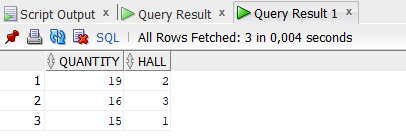
1. **Counting customers by hall:**

select count(\*) as quantity, hall

from seat

group by hall

order by quantity desc;



Descriptions:

* INSERT is an SQL statement that allows you to add rows to a table by filling them with values. Values ​​can be inserted by enumeration using the word values ​​and enumerating them in parentheses, separated by commas, or by using the select statement.
* WHERE is an SQL statement that specifies that a data management language statement should only act on records that meet certain criteria.
* The SQL ORDER BY statement sorts the output values. The SQL ORDER BY statement can be applied to both numeric and string columns.
* The SQL GROUP BY clause is used to combine the results of a selection on one or more columns. This query uses the SQL AS statement to specify a new column name.
* The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. There are two wildcards often used in conjunction with the LIKE operator: The percent sign (%) represents zero, one, or multiple characters
* JOIN is an operator of the SQL language, which is an implementation of the operation of joining tables by their foreign key.
* The COUNT() function returns the number of rows that matches a specified criterion

Procedures:

1. **Procedure that showing movies released in 2020.**

create procedure movies2020()

language plpgsql

as $$

declare m2020 record;

begin

for m2020 in

select buying\_date, movie\_name

from ticket

inner join movie

on movie.movie\_id=ticket.movie\_id

where buying\_date between '01-01-2020' and '01-01-2021'

loop

raise notice 'Movies released in 2020. Date: %, movie title: %', m2020.buying\_date, m2020.movie\_name;

end loop;

end;$$;

Изображение выглядит как стол

Автоматически созданное описание

1. **A procedure that shows all data about customer with an input id.**

create procedure cust\_info(

input\_id int

)

language plpgsql

as $$

declare cust\_info record;

begin

for cust\_info in

select \* from ticket

inner join movie

on movie.movie\_id=ticket.movie\_id

where customer\_id = input\_id

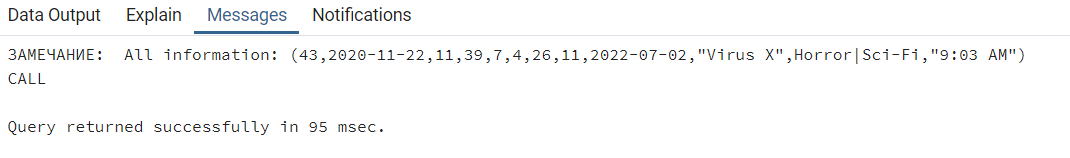
loop

raise notice 'All information: %', cust\_info;

end loop;

end;$$;

Example with id = 7:



1. **A procedure that shows all data by input date.**

create or replace procedure by\_date()

language plpgsql

as $$

declare info\_by\_date record;

begin

for info\_by\_date in

select customer.fname, employee.fname, final\_price, booking\_time

from customer

inner join booking

on customer.customer\_id=booking.customer\_id

inner join employee

on employee.employee\_id=booking.employee\_id

inner join payment

on payment.payment\_id=booking.payment\_id

where booking\_time = '07-04-2021'

loop

raise notice '%', info\_by\_date;

end loop;

end;$$;

Изображение выглядит как текст

Автоматически созданное описание

1. **A procedure that shows male workers.**

create procedure male()

language plpgsql

as $$

declare male record;

begin

for male in

select fname, lname

from employee

where gender = 'Male'

loop

raise notice 'Male workers. First name: %, last name: %', male.fname, male.lname;

end loop;

end;$$;

Изображение выглядит как текст, стол

Автоматически созданное описание

1. **A procedure that shows popularity of genres in descending order.**

create procedure popular\_genres()

language plpgsql

as $$

declare pg record;

begin

for pg in

select count(\*) as quantity, genre

from movie

group by genre

order by quantity desc

loop

raise notice '%', pg;

end loop;

end;$$;

Изображение выглядит как текст

Автоматически созданное описание

Cursor that shows the data the name of the client, employee, final price by date, in the example I used the date 7-04-2021

Cursors:

**Cursor that shows the data: the first name of the customer, employee, final price by date, in the example I used the date 07-04-2021.**

SET SERVEROUTPUT ON

DECLARE

CURSOR by\_date IS

select customer.fname, employee.fname, final\_price, booking\_time

from customer

inner join booking

on customer.customer\_id=booking.customer\_id

inner join employee

on employee.employee\_id=booking.employee\_id

inner join payment

on payment.payment\_id=booking.payment\_id

where booking\_time = '07-04-2021';

cfn customer.fname%TYPE;

efn employee.fname%TYPE;

fp payment.final\_price%TYPE;

bt booking.booking\_time%TYPE;

BEGIN

OPEN by\_date; LOOP

FETCH by\_date INTO cfn, efn, fp, bt;

EXIT WHEN by\_date%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customers first name: '|| cfn || ',

Employees first name:' || efn || ',

Final price:' || fp || ',

Booking time:' || bt);

end loop; close by\_date; end;

**Cursor that shows the data the buying date and movie title in 2020.**

SET SERVEROUTPUT ON

DECLARE

CURSOR movies2020 IS

select buying\_date, movie\_name

from ticket

inner join movie

on movie.movie\_id=ticket.movie\_id

where buying\_date between '01-01-2020' and '01-01-2021';

bd ticket.buying\_date%TYPE;

mn movie.movie\_name%TYPE;

BEGIN

OPEN movies2020; LOOP

FETCH movies2020 INTO bd, mn;

EXIT WHEN movies2020%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Buying date: ' || bd || ', movie title: ' || mn);

end loop; close movies2020; end;

Triggers:

**The trigger shown in this example saves information about the current user in that row whenever a row is added or changed. In addition, it requires that the employee's name be provided:**

CREATE FUNCTION check\_customer() RETURNS trigger AS $check\_customer$

BEGIN

IF NEW.fname IS NULL THEN

RAISE EXCEPTION 'fname cannot be null';

END IF;

RETURN NEW;

END;

$check\_customer$ LANGUAGE plpgsql;

CREATE TRIGGER check\_customer BEFORE INSERT OR UPDATE ON customer

FOR EACH ROW EXECUTE PROCEDURE check\_customer();

Изображение выглядит как текст

Автоматически созданное описание

**The trigger shown in this example ensures that any addition, change, or deletion of a row in the customer table is committed to the changes table:**

CREATE OR REPLACE FUNCTION changes() RETURNS TRIGGER AS $changes$

BEGIN

IF (TG\_OP = 'DELETE') THEN

INSERT INTO customer SELECT 'D', now(), user, OLD.\*;

RETURN OLD;

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO customer SELECT 'U', now(), user, NEW.\*;

RETURN NEW;

ELSIF (TG\_OP = 'INSERT') THEN

INSERT INTO customer SELECT 'I', now(), user, NEW.\*;

RETURN NEW;

END IF;

RETURN NULL;

END;

$changes$ LANGUAGE plpgsql;

CREATE TRIGGER changes

AFTER INSERT OR UPDATE OR DELETE ON customer

FOR EACH ROW EXECUTE PROCEDURE changes();

