

Uniwersytet w Białymstoku

Instytut Informatyki
Kierunek: Informatyka

Rok I / II-go stopnia / semestr 1

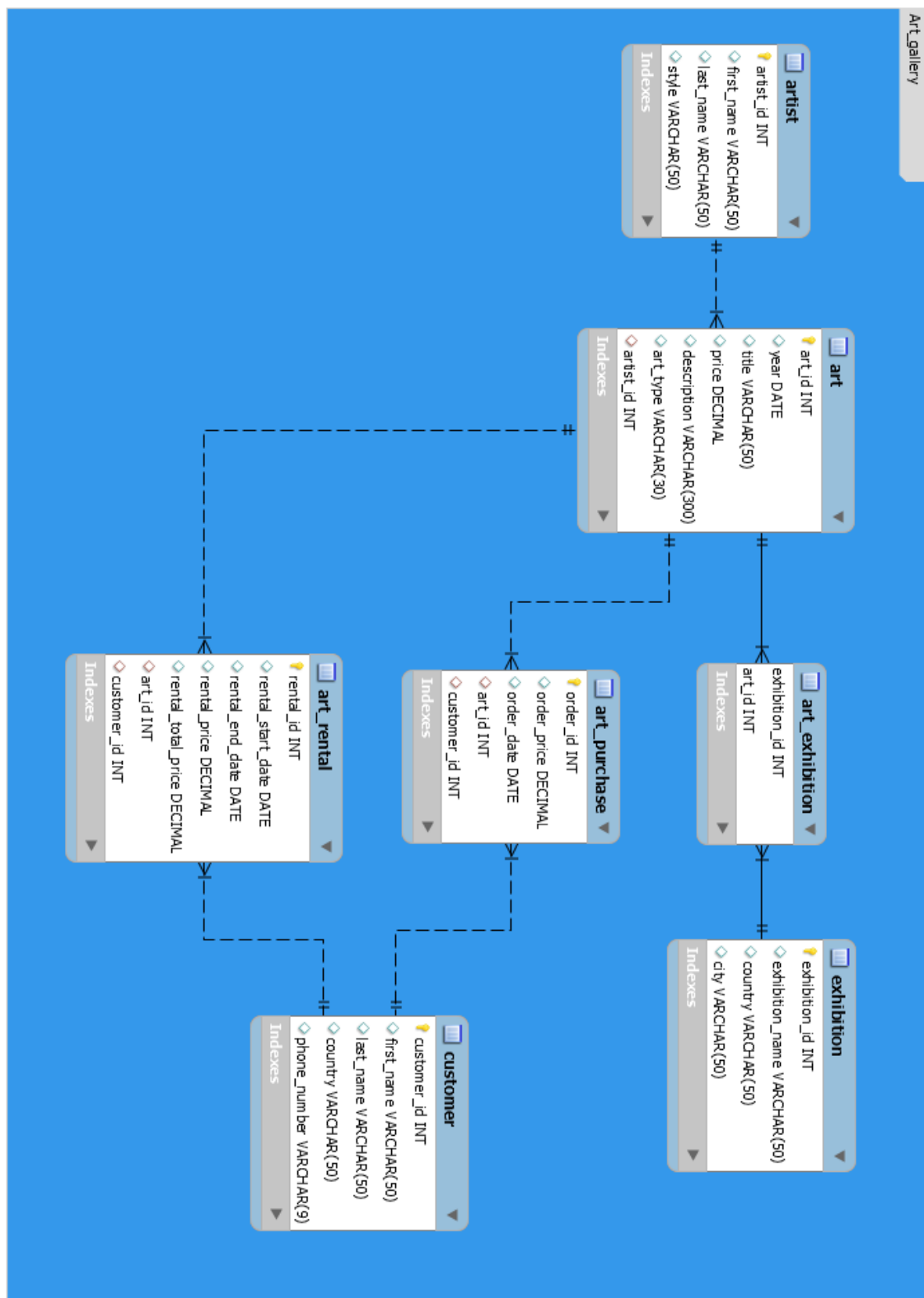
Laboratorium
Zaawansowane bazy danych

Projekt zaliczeniowy

Prowadzący:
dr hab. Agnieszka Bołtuć

Wykonał:
Wojciech Metelski

1. Schemat bazy danych



Rysunek 1 Schemat bazy danych galerii sztuki

2. Dane umieszczone w tabelach bazy

Tabela 1 Art

Art Id	Year	Title	Price	Art Type	Artist Id
1	11-Feb-1990	Conversations of Jealousy	10000	painting	1
15	04-Apr-2018	Glorious Goal	200000	painting	1
7	06-Jun-2015	Tranquil Girlfriend	17500	painting	2
3	05-Oct-2005	Exultant Curiosity	5000	painting	3
6	17-May-2007	Grieving Pride	150000	sculpture	4
8	03-Jan-1995	Invention of Balance	76000	sculpture	4
2	25-Mar-1997	Exhilarated Sanity	25000	sculpture	5
12	05-Oct-2016	Charity of Storms	35000	painting	5
13	07-Feb-2013	Strength of Solitude	14500	painting	5
10	19-Aug-2004	Everlasting Pride	45000	sculpture	6
9	11-Jul-2019	Anchored Breath	11000	painting	7
4	14-Dec-2010	Curtains of Adventure	50000	painting	8
5	25-Apr-2006	Infatuation of Authority	35000	painting	9
14	19-Jan-2012	Grace of Courage	32000	painting	10
11	12-Sep-2001	Hollow Problem	67800	sculpture	10

Tabela 2 Artist

Artist Id	First Name	Last Name	Style
1	Fabien	Cantin	abstract
2	Aiglentina	Lang	cubizm
3	Nicolas	Moreau	surrealizm
4	Honorina	Garrido	surrealizm
5	Lucelia	Pizarro	surrealizm
6	Aloisia	Mazzanti	Abstract Expressionism
7	Bruto	Calabrese	Abstract Expressionism
8	Ernesta	Genovese	Futurism
9	Bertoldo	Bergamaschi	Futurism
10	Kathrin	Nadel	Futurism

Tabela 3 Art_customer

Customer Id	First Name	Last Name	Phone Number	Country	Email
1	Dobrosław	Zawadzki	78 259 72 99	Poland	DobroslawZawadzki@jourrapide.com
2	Bolesław	Kalinowski	66 782 33 17	Poland	BoleslawKalinowski@jourrapide.com
3	Niklas	Koenig	06782 25 34 07	Germany	NiklasKoenig@jourrapide.com
4	Amethyst	Goodbody	070 6671 5987	United Kingdom	AmethystGoodbody@armyspy.com
5	Gabriel	Whitehead	079 7852 1334	United Kingdom	GabrielWhitehead@dayrep.com
6	Melissa	Johnson	078 6283 0318	United Kingdom	MelissaJohnson@jourrapide.com
7	Freya	Osborne	079 6465 2668	United Kingdom	FreyaOsborne@teleworm.us
8	Connor	Bradley	641 224 125	Spain	ConnorBradley@jourrapide.com
9	Filip	Černý	519 012 630	Czech Republic	FilipCerny@dayrep.com
11	Tomáš	Hrubý	723 301 417	Czech Republic	TomasHruby@teleworm.us
12	Lilla	Trevisani	0331 7819948	Italy	LillaTrevisani@dayrep.com
13	Cataldo	Marino	0330 7357492	Italy	CataldoMarino@dayrep.com
10	Václav	Žák	473 914 708	Czech Republic	VaclavZak@rhyta.com

Tabela 4 Art_purchase













Edit	Order Id	Order Price	Art Id	Customer Id	Order Date
	1	15000	1	5	23-Nov-2020
	2	52500	5	3	10-Oct-2020
	3	67500	10	7	11-Feb-2021
	4	114000	8	8	14-Jan-2021
	5	101700	11	7	03-Mar-2021
	6	21750	13	1	17-Jan-2021
	7	37500	2	2	19-Nov-2020
	8	75000	4	9	10-Jan-2021
	9	7500	3	4	01-Apr-2021
	10	52500	12	6	07-May-2021
	11	67500	10	4	18-Aug-2020
	12	7500	3	9	11-Apr-2020

Tabela 5 Art_rental

Rental Id	Rental Start Date	Rental End Date	Rental Price	Art Id	Customer Id	Rental Total Price
1	19-Nov-2020	20-Mar-2021	916.67	9	12	3696.25
2	21-Mar-2020	20-Jan-2021	1458.33	7	13	14536.26
3	22-May-2020	01-Dec-2020	4166.67	4	10	26344.11
4	20-Jun-2020	26-Aug-2020	2083.33	2	4	4569.89
5	03-Mar-2020	04-Jul-2020	416.67	3	7	1680.12
6	17-Feb-2021	25-Apr-2021	2916.67	12	5	6586.03
7	19-Jan-2021	07-Mar-2021	833.33	1	8	1344.08
8	18-Nov-2020	16-Feb-2021	1208.33	13	5	3547.03
9	02-May-2021	15-Aug-2021	2666.67	14	3	9118.29
10	07-Aug-2020	13-Jan-2021	16666.67	15	6	86559.16
20	20-Jan-2019	15-May-2020	1458.33	7	4	23098.07

Tabela 6 Art_exhibition

Exhibition Id	Country	City	Name
1	France	Louvre	Alchemical Chemistry: The Dysfunction of Progress
2	France	Paris	Mediating Charm: A Juried Show of Progress
3	Spain	Madrid	Fantastic Dreams: 15 Years of Dilettantism
4	Spain	Barcelona	Arbitrary Sustainability: Defying Interactivity
5	Spain	Bilbao	The Bureaucracies of Dissent: The Disjunction of Complacency
6	Italy	Florence	Parsing Dreams: Media Art and Progress
7	Italy	Mediolan	In Search of Relevance: A Retrospective of Change
8	Italy	Venice	Extravagant Relevance: Defying Remediation
9	Britain	London	After the Sustainability: A Remix of the Status Quo
10	Poland	Warsaw	After the Dreams: Deconstructing Aesthetic Forms and Their Opposites

Tabela 7 Art_at_exhibition

Exhibition Id	Art Id
1	2
1	15
2	4
2	5
3	7
4	3
4	13
5	6
5	14
6	2
6	9
6	10
7	1
7	5
8	11

3. Kod poszczególnych elementów projektu

Wykorzystane elementy języka PLSQL

Projekt rozszerza funkcjonalność bazy danych poprzez wykorzystanie wszystkich wymaganych przez projekt elementów PLSQL:

- Funkcja/procedura – elementy pakietu **ART_GALLERY**
- Wyzwalacz – **ART_PURCHASE_PRICE** oraz **ART_RENTAL_PRICE**
- Obsługa wyjątków - w wyzwalaczu **ART_RENTAL_PRICE**, funkcjach/procedurach **get_art_info()**, **purchase_income_from_art()**, **rental_income_from_art()**, **add_record()**
- rekord jawnie zdefiniowany - **ART_DATA**

oraz elementów dodatkowych:

- pakiet własny o wybranej funkcjonalności – definicja pakietu **ART_GALLERY**
- dowolna kolekcja i jej zastosowanie – definicja tabeli zagnieżdżonej **STRING_LIST** oraz wykorzystanie jej w funkcjach/procedurach **GET_ART_EXHIBITIONS()** oraz **ADD_RECORD()**
- przykład zapytania w formie dynamicznego sql – procedura **ADD_RECORD()**

Specyfikacja pakietu "ART_GALLERY"

```
create or replace package ART_GALLERY as
```

```
function purchase_income_from_art(  
    art INTEGER,  
    d1 Date,  
    d2 Date  
)  
return Number;
```

```
function rental_income_from_art(  
    art INTEGER,  
    d1 Date,  
    d2 Date  
)  
return Number;
```

```
function total_income_from_art(  
    art INTEGER,  
    d1 Date,  
    d2 Date  
)  
return Number;
```

```
function rental_income_from_artist(  
    artist INTEGER,  
    d1 Date,  
    d2 Date  
)  
return Number;
```

```
function purchase_income_from_artist(  
    artist INTEGER,  
    d1 Date,  
    d2 Date  
)  
return Number;
```

```
function total_income_from_artist(  
    artist INTEGER,  
    d1 Date,  
    d2 Date  
)  
return Number;
```

```
function rental_income(  
    d1 Date,  
    d2 Date  
)  
return Number;
```

```

function purchase_income(
    d1 Date,
    d2 Date
)
return Number;

function total_income(
    d1 Date,
    d2 Date
)
return Number;

function get_art_exhibitions(v_art integer)
return string_list;

function get_art_info(v_art integer)
return art_data;

procedure get_art_from_artist(v_artist integer);

procedure get_customer_art(customer integer);

procedure add_record(
    table_name varchar2,
    fields string_list,
    f_values string_list
);

end;

```

Całość projektu została zawarta w pakiecie, w którym zostały zdefiniowane wszystkie funkcje oraz procedury rozszerzające funkcjonalność bazy danych.

Definicja obiektu „ART_DATA”

```
CREATE OR REPLACE EDITIONABLE TYPE "ART_DATA" is object
(
  art_title varchar2(100),
  art_type varchar(50),
  art_nominal_price number,
  art_exhibitions string_list
)
```

Definicja obiektu „STRING_LIST”

```
CREATE OR REPLACE EDITIONABLE TYPE "STRING_LIST" as table of VARCHAR2(50)
```

Na poziomie bazy danych zostały zadeklarowane typy pomocnicze, które będą używane w stworzonych funkcjach/procedurach:

- **ART_DATA** – rekord zadeklarowany jawnie (na poziomie bazy danych nie da się zadeklarować rekordu, więc użyty został typ obiektowy)
- **STRING_LIST** – tablica zagnieżdżona varcharów

Ciało procedury get_customer_art()

```
procedure get_customer_art(customer integer)
as
    cursor customer_purchase is
    (
        select art.title, art_purchase.order_price
        from art_purchase inner join art
        on art.art_id = art_purchase.art_id
        where art_purchase.customer_id = customer
    );
    cursor customer_rental is
    (
        select art.title, art_rental.rental_price
        from art_rental inner join art
        on art.art_id = art_rental.art_id
        where art_rental.customer_id = customer
    );
begin
    dbms_output.put_line('Purchased art:');
    for elem in customer_purchase loop
        dbms_output.put_line('Title: ' || elem.title ||
                               ' | Price: ' || elem.order_price);
    end loop;

    dbms_output.put_line('Rented art:');
    for elem in customer_rental loop
        dbms_output.put_line('Title: ' || elem.title ||
                               ' | Rental: ' || elem.rental_price);
    end loop;
end;
```

Procedura pozwala wyświetlić dzieła sztuki, które zostały kupione/dzierżawione przez konkretnego klienta. W procedurze zostały wykorzystane dwa kursory wskazujące na tabele „art_purchase” oraz „art_rental”.

SQL Commands

Rows 10 ? Clear Command

```
begin
  art_gallery.get_customer_art(3);
end;
```

Results

Explain

Describe

Saved SQL

History

Purchased art:
Title: Infatuation of Authority | Price: 52500
Rented art:
Title: Grace of Courage | Rental: 2666.67

Statement processed.

0.00 seconds

SQL Commands

Rows 10 ? Clear Command

```
begin
  art_gallery.get_customer_art(5);
end;
```

Results

Explain

Describe

Saved SQL

History

Purchased art:
Title: Conversations of Jealousy | Price: 15000
Rented art:
Title: Charity of Storms | Rental: 2916.67
Title: Strength of Solitude | Rental: 1208.33

Statement processed.

0.01 seconds

Rysunek 2 Przykłady działania procedury `get_customer_art()`

Ciało funkcji `get_art_exhibitions()`

```
function get_art_exhibitions(v_art integer)
return string_list
as
    exhib_list string_list;
begin
    select name bulk collect into exhib_list
    from (art_exhibition inner join art_at_exhibition
        on art_exhibition.exhibition_id = art_at_exhibition.exhibition_id)
    where art_at_exhibition.art_id = v_art;
    return exhib_list;
end;
```

Funkcja zwraca listę nazw wystaw sztuki, na których można znaleźć dane dzieło sztuki. Nazwy przechowywane są w zadeklarowanym typie **STRING_LIST** będącym tabelą zagnieżdżoną.

Ciało funkcji `get_art_info()`

```
function get_art_info(v_art integer)
return art_data
as
    artist_exists integer;
    art_info art_data;
begin
    select art_data(title, art_type, price, get_art_exhibitions(art_id))
    into art_info from art
    where art.art_id = v_art;
    return art_info;
exception
    when no_data_found then
        dbms_output.put_line('No art found!');
end;
```

Funkcja zwraca wybrane informacje o danym dziele sztuki. Informacje zostają przypisane do zmiennej rekordowej typu **ART_DATA**, która zostaje zwrócona przez funkcję. Funkcja **get_art_info()** wywołuje wewnątrz siebie funkcję **get_art_exhibition()**, zwracającą listę wystaw, która zostaje zapisana w rekordzie.

Definicja procedury get_art_from_artist()

```
procedure get_art_from_artist(v_artist integer)
as
    cursor arts is (
        select art_id
        from art
        where art.artist_id = v_artist
    );
    v_art arts%rowtype;
    v_art_data art_data;
    ex_t varchar2(100);
begin
    for v_art in arts loop
        v_art_data := get_art_info(v_art.art_id);
        dbms_output.put_line(v_art_data.art_title || ' | '
            || v_art_data.art_type || ' | ' || v_art_data.art_nominal_price);
        dbms_output.put('Exhibitions: ');
        for i in 1..v_art_data.art_exhibitions.count loop
            dbms_output.put(v_art_data.art_exhibitions(i) || ' | ');
        end loop;
        DBMS_OUTPUT.NEW_LINE;
    end loop;
end;
```

Procedura wyświetla informacje o dziełach sztuki stworzonych przez danego artystę. Procedura wywołuje wewnątrz pętli funkcję **get_art_info()**, a następnie wyświetla poszczególne informacje.

SQL Commands

Rows 10 ? Clear Command

```
begin
  art_gallery.get_customer_art(3);
end;
```

Results

Explain

Describe

Saved SQL

History

Purchased art:

Title: Infatuation of Authority | Price: 52500

Rented art:

Title: Grace of Courage | Rental: 2666.67

Statement processed.

0.00 seconds

SQL Commands

Rows 10 ? Clear Command Find Tables

```
begin
  art_gallery.get_art_from_artist(5);
end;
```

Results

Explain

Describe

Saved SQL

History

Exhilarated Sanity | sculpture | 25000

Exhibitions: Alchemical Chemistry: The Dysfunction of Progress | Parsing Dreams: Media Art and Progress | After the Sustainability: A Remix of the Status Quo |

Charity of Storms | painting | 35000

Exhibitions: After the Sustainability: A Remix of the Status Quo |

Strength of Solitude | painting | 14500

Exhibitions: Arbitrary Sustainability: Defying Interactivity | Extravagant Relevance: Defying Remediation |

Statement processed.

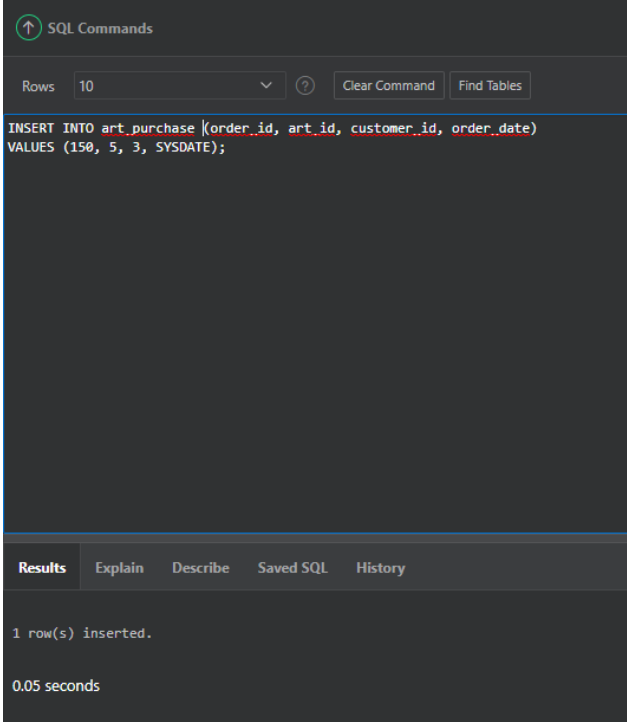
0.00 seconds

Rysunek 3 Przykład działania procedury `get_art_from_artist()`

Definicja wyzwalacza ART_PURCHASE_PRICE

```
create or replace trigger "ART_PURCHASE_PRICE"
BEFORE
update or insert on "ART_PURCHASE"
for each row
declare
    v_price NUMBER(8, 2);
begin
    select price into v_price from art
    where art.art_id = :new.art_id;
    :new.order_price := 1.5 * v_price;
end;
```

Wyzwalacz **ART_PURCHASE_PRICE** ustawia automatycznie cenę zakupu danego dzieła sztuki na podstawie jego ceny nominalnej. Cena zakupu wynosi 1,5 ceny nominalnej.



The screenshot shows the SQL Developer interface. At the top, there's a tab labeled "SQL Commands". Below it, a toolbar contains a "Rows" dropdown set to "10", a help icon, and buttons for "Clear Command" and "Find Tables". The main text area contains the following SQL command:

```
INSERT INTO art_purchase (order_id, art_id, customer_id, order_date)
VALUES (150, 5, 3, SYSDATE);
```

Below the command area, there's a tabbed interface with "Results" selected. The results pane shows:

```
1 row(s) inserted.

0.05 seconds
```

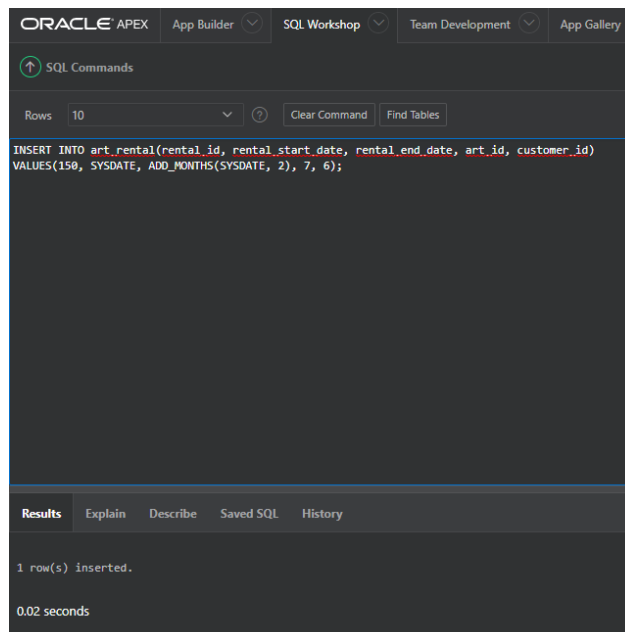
ART_PURCHASE													<div>+ v</div>
Table	Data	Indexes	Model	Constraints	Grants	Statistics	UI Defaults	Triggers	Dependencies	SQL	REST		
Query		Count Rows	Insert Row										
Data													
EDIT		ORDER_ID		ORDER_PRICE		ART_ID		CUSTOMER_ID		ORDER_DATE			
		3		67500		10		7		11-Feb-2021			
		4		114000		8		8		14-Jan-2021			
		5		101700		11		7		03-Mar-2021			
		7		37500		2		2		19-Nov-2020			
		8		75000		4		9		10-Jan-2021			
		10		52500		12		6		07-May-2021			
		12		7500		3		9		11-Apr-2020			
		1		15000		1		5		23-Nov-2020			
		2		52500		5		3		10-Oct-2020			
		6		21750		13		1		17-Jan-2021			
		9		7500		3		4		01-Apr-2021			
		11		67500		10		4		18-Aug-2020			
		150		52500		5		3		28-Jan-2021			

Rysunek 4 Przykład działania wyzwalacza ART_PUCTHASE_PRICE

Definicja wyzwalacza ART_RENTAL_PRICE

```
create or replace trigger "ART_RENTAL_PRICE"
BEFORE
update or insert on "ART_RENTAL"
for each row
declare
v_price NUMBER(8, 2);
invalid_rental_date exception;
begin
    if :new.rental_start_date >= :new.rental_end_date then
        raise invalid_rental_date;
    end if;
    SELECT price into v_price from art
    where art.art_id = :new.art_id;
    :new.rental_price := v_price / 12;
    :new.rental_total_price := months_between(
        :new.rental_end_date,
        :new.rental_start_date) * :new.rental_price;
    exception
        when invalid_rental_date then
            dbms_output.put_line('Start date is greater than end date');
end;
```

Wyzwalacz **ART_RENTAL_PRICE** ustawia automatycznie cenę wynajmu danego dzieła sztuki na podstawie jego ceny nominalnej. Cena za miesiąc wynajmu to 1/12 ceny nominalnej. Na podstawie okresu wynajmu zostaje też policzona całkowita cena za dzierżawienie dzieła sztuki.



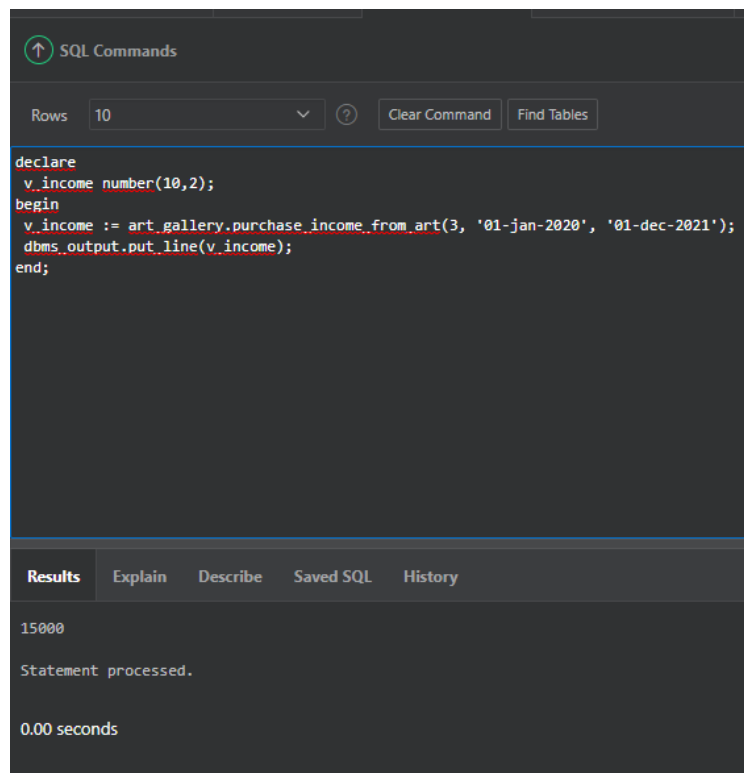
ART_RENTAL												+ v		
Table	Data	Indexes	Model	Constraints	Grants	Statistics	UI Defaults	Triggers	Dependencies	SQL	REST			
Query														
	Count Rows	Insert Row												
Data														
EDIT	RENTAL_ID	RENTAL_START_DATE				RENTAL_END_DATE		RENTAL_PRICE	ART_ID	CUSTOMER_ID		RENTAL_TOTAL_PRICE		
	2	21-Mar-2020				20-Jan-2021		1458.33	7	13		14536.26		
	3	22-May-2020				01-Dec-2020		4166.67	4	10		26344.11		
	4	20-Jun-2020				26-Aug-2020		2083.33	2	4		4569.89		
	5	03-Mar-2020				04-Jul-2020		416.67	3	7		1680.12		
	6	17-Feb-2021				25-Apr-2021		2916.67	12	5		6586.03		
	9	02-May-2021				15-Aug-2021		2666.67	14	3		9118.29		
	20	20-Jan-2019				15-May-2020		1458.33	7	4		23098.07		
	150	28-Jan-2021				28-Mar-2021		1458.33	7	6		2916.66		
	1	19-Nov-2020				20-Mar-2021		916.67	9	12		3696.25		
	7	19-Jan-2021				07-Mar-2021		833.33	1	8		1344.08		
	8	18-Nov-2020				16-Feb-2021		1208.33	13	5		3547.03		
	10	07-Aug-2020				13-Jan-2021		16666.67	15	6		86559.16		

Rysunek 5 Przykład działania wyzwalacza **ART_RENTAL_PRICE**

Definicja funkcji `purchase_income_from_art()`

```
function purchase_income_from_art(  
    ART IN INTEGER,  
    d1 Date,  
    d2 Date  
) return NUMBER  
as  
    total NUMBER := 0;  
begin  
    SELECT sum(order_price) INTO total  
    FROM art_purchase  
    WHERE (art_purchase.art_id = art) and  
          (art_purchase.order_date between d1 and d2);  
    return total;  
exception  
    when no_data_found then  
        DBMS_OUTPUT.PUT_LINE('No art found!');  
end ;
```

Funkcja ***purchase_income_from_art()*** zwraca sumę zysków ze sprzedaży danego dzieła, z przedziału czasowego podanego na wejściu.



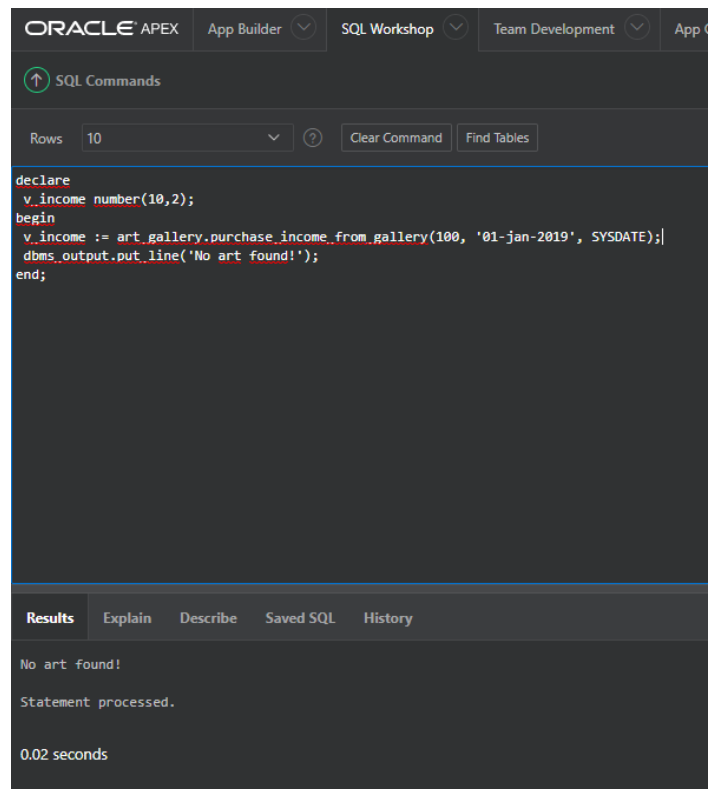
The screenshot shows a SQL command window with the following content:

```
SQL Commands  
Rows: 10  
declare  
  v_income number(10,2);  
begin  
  v_income := art_gallery.purchase_income_from_art(3, '01-jan-2020', '01-dec-2021');  
  dbms_output.put_line(v_income);  
end;
```

Below the command window, there is a tabbed interface with the following tabs: Results, Explain, Describe, Saved SQL, History. The Results tab is selected, showing the output:

```
15000  
Statement processed.  
0.00 seconds
```

Rysunek 6 Przykład działania ***purchase_income_from_art()***



Rysunek 7 Przykład obsługi wyjątku `no_data_found` w `purchase_income_from_art()`

Definicja funkcji purchase_income_from_artist()

```
function purchase_income_from_artist(  
    ARTIST IN INTEGER,  
    d1 Date,  
    d2 Date  
) return NUMBER  
as  
    total NUMBER;  
begin  
    SELECT sum(purchase_income_from_art(art_id, d1, d2)) INTO total  
    FROM art  
    WHERE art.artist_id = artist;  
    return total;  
end ;
```

Funkcja ***purchase_income_from_artist()*** zwraca sumę zysków ze sprzedaży dzieł danego artysty. Funkcja wywołuje ***purchase_income_from_art()*** dla każdego dzieła artysty i zwraca sumę.

Definicja funkcji purchase_income()

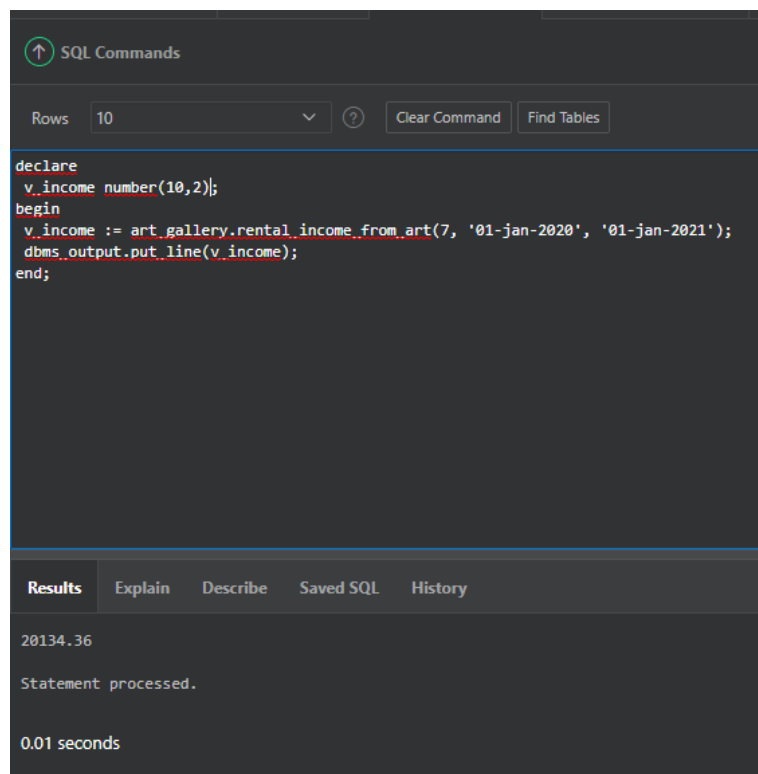
```
function purchase_income(  
    d1 Date,  
    d2 Date  
)  
return NUMBER  
as  
    total NUMBER := 0;  
begin  
    SELECT sum(order_price) INTO total  
    FROM art_purchase  
    WHERE art_purchase.order_date between d1 and d2;  
    return total;  
end;
```

Funkcja ***purchase_income()*** zwraca zysk ze sprzedaży wszystkich dzieł sztuki w danym okresie czasu.

Definicja funkcji rental_income_from_art()

```
function rental_income_from_art(  
    ART IN INTEGER,  
    d1 Date,  
    d2 Date  
) return NUMBER  
as  
    total NUMBER := 0;  
begin  
    SELECT sum(months_between(  
        least(rental_end_date, d2),  
        greatest(rental_start_date, d1)  
    ) * rental_price  
    )  
    INTO total  
    FROM art_rental  
    WHERE art_rental.art_id = art;  
    return total;  
exception  
    when no_data_found then  
        DBMS_OUTPUT.PUT_LINE('No art found!');  
end ;
```

Funkcja ***rental_income_from_art()*** zwraca sumę zysków z dzierżawienia danego dzieła sztuki w danym okresie czasu.



The screenshot shows a SQL IDE interface. At the top, there's a tab labeled 'SQL Commands'. Below it, a dropdown menu shows 'Rows' set to '10'. To the right are buttons for 'Clear Command' and 'Find Tables'. The main text area contains the following SQL code:

```
declare  
v_income number(10,2);  
begin  
v_income := art_gallery.rental_income_from_art(7, '01-jan-2020', '01-jan-2021');  
dbms_output.put_line(v_income);  
end;
```

Below the code area, there's a 'Results' tab selected, showing the output of the query:

```
20134.36  
  
Statement processed.  
  
0.01 seconds
```

Rysunek 8 Przykład działania ***rental_income_from_art()***

Definicja funkcji rental_income_from_artist()

```
function rental_income_from_artist(  
    ARTIST IN INTEGER,  
    d1 Date,  
    d2 Date  
) return NUMBER  
as  
    total NUMBER;  
begin  
    SELECT sum(rental_income_from_art(art_id, d1, d2)) INTO total  
    FROM art  
    WHERE art.artist_id = artist;  
    return total;  
end ;
```

Funkcja **rental_income_from_artist()** zwraca sumę zysków z dzierżawy dzieł danego artysty. Funkcja wywołuje **rental_income_from_art()** dla każdego dzieła artysty i zwraca sumę.

Definicja funkcji rental_income()

```
function rental_income(  
    d1 Date,  
    d2 Date  
)  
return NUMBER  
as  
    total NUMBER;  
begin  
    SELECT sum(months_between(  
        least(rental_end_date, d2),  
        greatest(rental_start_date, d1)  
    ) * rental_price  
    )  
    into total  
    FROM art_rental;  
    return total;  
end;
```

Funkcja **rental_income()** zwraca zysk ze sprzedaży wszystkich dzieł sztuki w danym okresie czasu.

Ciało funkcji `total_income_from_art()`

```
function total_income_from_art(  
    art INTEGER,  
    d1 Date,  
    d2 Date  
) return Number  
as  
    total NUMBER;  
begin  
    total := rental_income_from_art(art, d1, d2) +  
        purchase_income_from_art(art, d1, d2);  
    return total;  
end;
```

Funkcja **`total_income_from_art()`** zwraca całkowity zysk ze sprzedaży danego dzieła.

Ciało funkcji `total_income_from_artist()`

```
function total_income_from_artist(  
    ARTIST IN INTEGER,  
    d1 Date,  
    d2 Date  
) return NUMBER  
as  
    total NUMBER;  
begin  
    total := rental_income_from_artist(artist, d1, d2) +  
        purchase_income_from_artist(artist, d1, d2);  
    return total;  
end;
```

Funkcja **`total_income_from_artist()`** zwraca całkowity zysk ze sprzedaży dzieł danego artysty.

Ciało funkcji `total_income()`

```
function total_income(  
    d1 Date,  
    d2 Date  
)  
return NUMBER  
as  
    total NUMBER;  
begin  
    total := rental_income(d1, d2) +  
        purchase_income(d1, d2);  
    return total;  
end;
```

Funkcja **`total_income()`** zwraca całkowity zysk ze sprzedaży wszystkich dzieł sztuki.

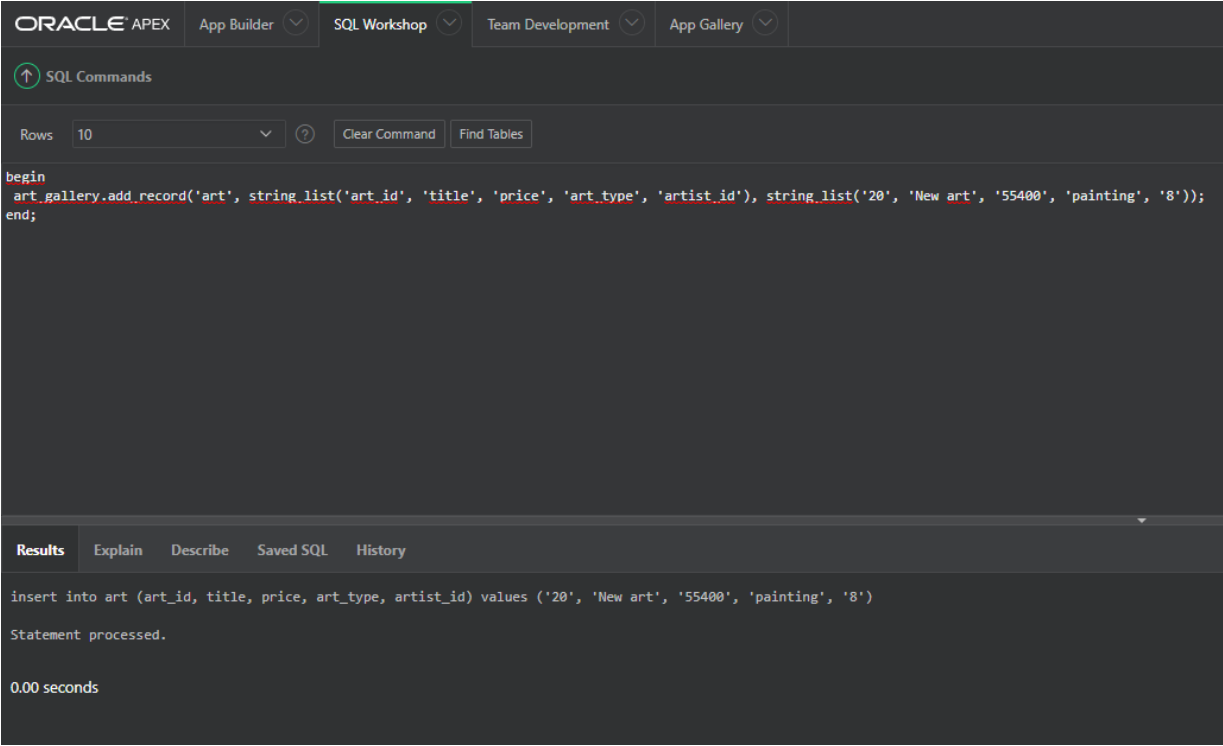
Ciało procedury `add_record()`

```
procedure add_record(
    table_name varchar2,
    fields string_list,
    f_values string_list
)
as
    sql_task varchar2(300) := 'insert into ';
    i integer;
    t_names string_list := string_list(
        'art',
        'artist',
        'art_rental',
        'art_purchase',
        'customer',
        'art_at_exhibition'
    );
    not_art_gallery exception;
begin
    if table_name not member of t_names then
        raise not_art_gallery;
    end if;
    sql_task := sql_task || table_name || ' (';
    for i in 1..fields.count()-1 loop
        sql_task := sql_task || fields(i) || ', ';
    end loop;
    sql_task := sql_task || fields(fields.count());
    sql_task := sql_task || ') ' || 'values (';
    for i in 1..f_values.count()-1 loop
        sql_task := sql_task || ''' ' || f_values(i) || ''' ' || ', ';
    end loop;
    sql_task := sql_task || ''' ' || f_values(f_values.count()) || ''';
    sql_task := sql_task || ')';
    dbms_output.put_line(sql_task);
    execute immediate sql_task;
exception
when not_art_gallery then
    dbms_output.put_line('Not art gallery table');
when others then
    dbms_output.put_line('Creating record failed');
end;
```

Procedura **`add_record()`** służy do wstawiania nowych rekordów do tabeli będących częścią **ART_GALLERY**. Na podstawie wartości podanych w tablicach budowane jest polecenie sql postaci:

INSERT INTO ***table_name*** (***fields(0)***, ***fields(1)***, ...) VALUES ("***f_values(0)***", "***f_values(1)***", ...),

które jest później wywoływane dynamicznym sql'em.



The screenshot shows the Oracle APEX Data tab for the 'art' table. The table has columns: ART_ID, YEAR, TITLE, PRICE, ART_TYPE, and ARTIST_ID. The data is as follows:

EDIT	ART_ID	YEAR	TITLE	PRICE	ART_TYPE	ARTIST_ID
	20	-	New art	55400	painting	8

At the bottom right, there is a 'Download' button and a pagination indicator: 'Previous row(s) 16 - 16 of 16'.

Rysunek 9 Przykład działania `add_record()`

ORACLE APEX

App Builder

SQL Workshop

Team Development

App Gallery

SQL Commands

Rows10Clear CommandFind Tables

```
begin
  art_gallery.add_record('tabela', string_list('art_id', 'title', 'price', 'art_type', 'artist_id'), string_list('30', 'Next new art', '23000', 'painting', 20));
end;
```

Results

Explain

Describe

Saved SQL

History

Not art gallery table

Statement processed.

0.01 seconds

Rysunek 10 Przykład obsługi wyjątku *not_art_gallery* w *add_record()*