CS 257 – Database System Principles, Section 1
Group Project #2 (9 pts)
Submitted by Team 2:

Pooja Krishan (016315442)
Uzma Shaikh (016605485)
Bhargavi Chevva (016591978)
Muneeba Hashmi (016662709)

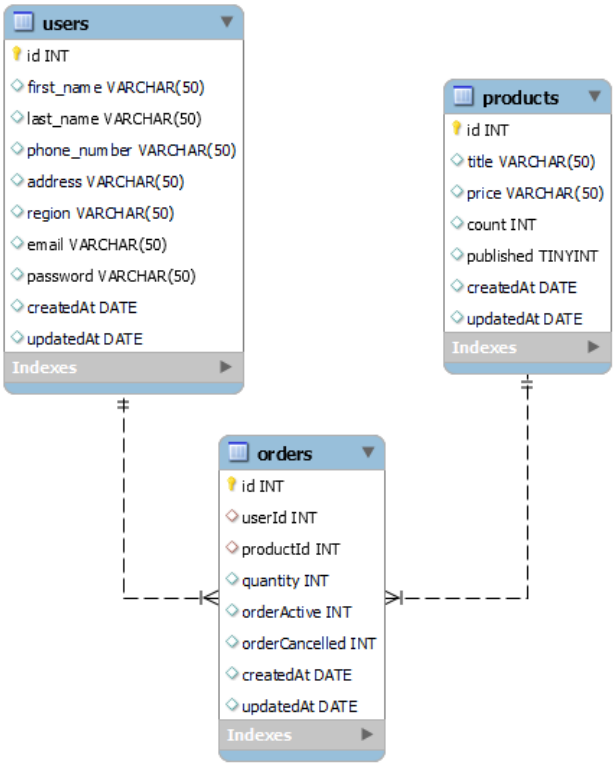
SHOPPING CART

Problem Statement:

- **To construct a schema for a shopping cart database and to reverse engineer the model to extract an EER diagram.**
- **To query the database with simple and complex operations (i.e., 2- or 3-way Join, self-join, nested queries, aggregation, Group-by, etc.).**
- **To perform Insert/Update/Delete (IUD) operations as well.**
- **To create a View and perform an insert/update on the View (trigger)**
- **To start a transaction and perform an operation that violates a constraint and demonstrate that you extended the schema so SQL runtime can maintain database consistency.**

Schema definitions (ER Diagram):

For the purpose of this project, one database called shopping_Cart containing 3 tables, users, products and orders, was used.



Database: Shopping Cart.

Database specs: Represents a store that sells grocery items.

Database definition:

CREATE DATABASE shopping_Cart;

USE shopping_Cart;

Table definitions:

1) Table1 name: Users

Table1 specs: Stores details of users of Shopping cart application.

Creating table1:

```
create table users (
id INT PRIMARY KEY,
first_name VARCHAR(50),
last_name VARCHAR(50),
phone_number VARCHAR(50),
address VARCHAR(50),
region VARCHAR(50),
email VARCHAR(50),
password VARCHAR(50),
createdAt DATE,
updatedAt DATE
);
```

Populating table1:

```
insert into users (id, first_name, last_name, phone_number, address, region, email, password, createdAt, updatedAt) values (1, 'Mattias', 'Pirozzi', '309-377-0807', '8284 Prairie Rose Park', 'Illinois', 'mpirozzi0@google.ru', 'EKiHxLkNn', '2019-11-29', '2019-06-23');
```

Users table schema:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	
first_name	varchar(50)	NO		NULL	
last_name	varchar(50)	NO		NULL	
phone_number	varchar(50)	NO		NULL	
address	varchar(50)	NO		NULL	
region	varchar(50)	NO		NULL	
email	varchar(50)	NO		NULL	
password	varchar(50)	NO		NULL	
createdAt	date	NO		NULL	
updatedAt	date	NO		NULL	

2) Table2 name — Products

Table2 specs — Stores details of products (grocery items).

Creating table2

```
create table products (
id INT PRIMARY KEY,
title VARCHAR(50),
price VARCHAR(50),
count INT,
published TINYINT,
createdAt DATE,
updatedAt DATE
);
```

Populating table2

```
insert into products (id, title, price, count,
published, createdAt, updatedAt) values (1,
'Shrimp - Black Tiger 6 - 8', '$5.94', 57, 1,
'2020-09-07', '2022-07-27');
```

Products table Schema:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	
title	varchar(50)	NO		NULL	
price	varchar(50)	NO		NULL	
count	int	YES		NULL	
published	tinyint	NO		NULL	
createdAt	date	NO		NULL	
updatedAt	date	NO		NULL	

3) Table3 name — Orders

Table3 specs — Stores details of orders made by customers.

Creating table3

```
create table orders (
id INT PRIMARY KEY,
userId INT,
productId INT,
quantity INT,
orderActive TINYINT,
orderCancelled TINYINT,
createdAt DATE,
updatedAt DATE
);
```

Adding foreign key constraints to table3	ALTER TABLE orders ADD CONSTRAINT productIdFK FOREIGN KEY (productId) REFERENCES products(id); ALTER TABLE orders ADD CONSTRAINT userIdFK FOREIGN KEY (userId) REFERENCES users(id);
Populating table3	insert into orders (id, userId, productId, quantity, orderActive, orderCancelled, createdAt, updatedAt) values (1, '170', '766', 95, 0, 1, '2020-11-18', '2021-07-26');

Orders table schema:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	
userId	int	NO	MUL	NULL	
productId	int	NO	MUL	NULL	
quantity	int	YES		NULL	
orderActive	tinyint	NO		NULL	
orderCancelled	tinyint	NO		NULL	
createdAt	date	NO		NULL	
updatedAt	date	NO		NULL	

4)

Table 4	persons_archive
Table 4 specification	stores details of all deleted users
Table 4 creation	create table person_archive (id INT, first_name VARCHAR(50), last_name VARCHAR(50), phone_number VARCHAR(50), address VARCHAR(50), region VARCHAR(50), email VARCHAR(50), password VARCHAR(50), createdAt DATE, updatedAt DATE);

5)

Table 5	backup
Table 5 specification	stores snapshot of products at given time
Table 5 creation	create table backup (

	id INT PRIMARY KEY, title VARCHAR(50), price VARCHAR(50), count INT, published TINYINT, createdAt DATE, updatedAt DATE);

All three tables were created and populated using SQL create and insert statements in a .sql file. (attached)

List of queries and output:

1.

Find average sales made on a certain day or timeframe.	select avg(o.quantity*trim(leading '$' from p.price)) as avgtotalSales, month(o.updatedAt) as saleMonth, year(o.updatedAt) as saleYear from orders o, products p where o.productId=p.id group by saleMonth, saleYear order by avgtotalSales desc, saleYear desc, saleMonth desc;"	Constructs used: Join, Group by, Order by

Output:

avgtotalSales	saleMonth	saleYear
18072.261428571433	12	2020
17736.568461538463	3	2021
17371.656818181815	6	2022
17301.32740740741	12	2019
15998.980500000001	5	2020
15866.037142857147	9	2021
15588.05857142857	7	2021

2.

Group orders by region.	select sum(o.quantity*trim(leading '$' from p.price)) as totalOfAllOrders, u.region from orders o, products p, users u where o.userId = u.id group by region order by totalOfAllOrders desc;	Constructs used: Join, Group by, Order by

<u>Output</u>:

totalOfAllOrders	region
1515829366.3199992	California
1406992766.9599962	Texas
1077940057.6799984	Florida
668022772.2400011	New York
510616568.95999956	District of Columbia
497902012.9600008	Virginia
433312068.47999704	Colorado
424920461.519999	Missouri
393896944.8799998	Illinois
373045073.0399989	Ohio
369484997.3599992	Georgia
359059061.4399984	Pennsylvania
345835923.2000001	Massachusetts
332612784.96000004	Washington
268785713.83999944	North Carolina
248442424.2400012	Michigan
233439248.16000003	Minnesota

3. a)

Find customers who generate most sales for the store.	SELECT userId, users.first_name, users.last_name, users.email, users.address, users.region, users.phone_number, SUM(orders.quantity) AS total_orders, SUM(REPLACE(products .price,'$','') *	Constructs used: Multiway Join, Group by, Order by

	orders.quantity) AS total_sales FROM orders INNER JOIN users ON orders.userId = users.id INNER JOIN products ON orders.productId = products.id GROUP BY orders.userId ORDER BY total_sales desc;	

3. b)

Find customers who generate least sales for the store.	SELECT userId, users.first_name, users.last_name, users.email, users.address, users.region, users.phone_number, SUM(orders.quantity) AS total_orders, SUM(REPLACE(products .price,'$',") * orders.quantity) AS total_sales FROM orders INNER JOIN users ON orders.userId = users.id INNER JOIN products ON orders.productId = products.id GROUP BY orders.userId ORDER BY total_sales asc;	Constructs used: Multiway Join, Group by, Order by

Output:

userId	first_name	last_name	email	address	region	phone_number	total_orders	total_sales
362	Mirabel	Grinstead	mgrinsteada1@nationalgeographic.com	447 Cherokee Plaza	Georgia	912-720-6392	246	110925.26
407	Fallon	Crosham	fcroshamba@cam.ac.uk	22 Rigney Circle	Florida	904-923-2723	151	74049.20999999999
101	Ringo	Gumley	rgumley2s@amazon.de	985 Maple Wood Terrace	New York	718-794-4000	228	72586.4
865	Davy	Alishoner	dalishonero0@hhs.gov	271 Burrows Avenue	Florida	772-760-4430	222	72119.54
292	Isidoro	Scoterbosh	iscoterbosh83@microsoft.com	17 Sycamore Court	Oregon	503-532-7491	203	68017.27
141	Gawen	Dunmuir	gdunmuir3w@ftc.gov	0 Doe Crossing Crossing	Minnesota	218-376-9196	136	65902.08
976	Dori	Leonarde	dleonarder3@webmd.com	4 Lerdahl Center	Florida	407-423-6476	170	64146.479999999996
98	Quintilla	Kinnach	qkinnach2p@gizmodo.com	653 Mayer Road	Georgia	404-195-7198	148	61858.03999999999
827	Alvin	Javes	ajavesmy@storify.com	9 Mendota Lane	Texas	214-303-3615	125	60423.92

userId	first_name	last_name	email	address	region	phone_number	total_orders	total_sales
104	Patsy	Kobes	pkobes2v@a8.net	6983 Garrison Lane	Indiana	260-228-2261	6	115.08
781	Barbe	Dewicke	bdewickelo@nature.com	07 Hanover Hill	Delaware	302-659-2943	11	144.32
293	Maryjane	Ahlin	mahlin84@wsj.com	21811 Oakridge Way	Missouri	816-509-1720	1	171.28
358	Ly	Pitsall	lpitsall9x@freewebs.com	8 Dayton Plaza	California	858-576-2288	17	184.62
595	Katrine	Spriggen	kspriggengi@lycos.com	152 Troy Terrace	Georgia	770-808-6416	16	228
332	Valentin	Blackstock	vblackstock97@disqus.com	55 Farragut Plaza	Wisconsin	608-269-2208	15	246.15
782	Karleen	Darkott	kdarkottlp@hc360.com	103 Thackeray Hill	New York	212-660-1758	7	293.23
763	Kendra	Pakenham	kpakenhaml6@privacy.gov.au	656 Iowa Street	Colorado	303-223-3580	47	303.15000000000003
880	Rikki	Pinchon	rpinchonof@naver.com	3150 Susan Junction	Georgia	678-322-3785	51	346.8

4.

Find users who bought the same product.	SELECT userId, users.first_name, users.last_name, users.region, orders.productId, products.title, orders.quantity, products.price FROM users INNER JOIN orders ON users.id = orders.userId INNER JOIN products ON orders.productId = products.id HAVING orders.productId IN (SELECT productId FROM users INNER JOIN orders ON users.id = orders.userId INNER JOIN products ON orders.productId = products.id GROUP BY (productId) HAVING count(orders.productId) > 1) ORDER BY orders.productId asc;	Constructs used: Multiway Join, Sub query, aggregate functions

Output:

userId	first_name	last_name	region	productId	title	quantity	price
473	Guido	Perfect	Washington	5	Pasta - Cannelloni, Sheets, Fresh	95	$187.34
965	Benson	Driuzzi	Tennessee	5	Pasta - Cannelloni, Sheets, Fresh	45	$187.34
245	Colly	Pozzi	North Carolina	12	Pork - European Side Bacon	61	$94.08
459	Reider	Vasyutochkin	South Carolina	12	Pork - European Side Bacon	52	$94.08
761	Leonard	Wainscoat	New York	14	Beef - Bones, Marrow	98	$193.31
309	Nils	Darnody	District of Columbia	14	Beef - Bones, Marrow	15	$193.31
110	Valida	Lyver	Texas	18	Soup - Campbells Bean Medley	10	$323.06

5.

Find products usually bought by the same user.	SELECT userId, orders.productId, orders.quantity, products.title, products.price FROM users INNER JOIN orders ON users.id = orders.userId INNER JOIN products ON orders.productId = products.id HAVING orders.userId IN (SELECT orders.userId FROM users INNER JOIN orders ON users.id = orders.userId INNER JOIN products ON orders.productId = products.id GROUP BY (orders.userId) HAVING count(orders.userId) > 1) ORDER BY orders.userId asc;	Constructs used: Multiway Join, Sub query, aggregate functions

Output:

userId	productId	quantity	title	price
3	432	64	Rice - 7 Grain Blend	$296.31
3	754	67	Wasabi Paste	$251.29
7	522	22	Cheese - Valancey	$96.73
7	438	93	Napkin White	$14.22
10	676	75	Watercress	$81.34
10	636	24	Pork - Back, Short Cut, Boneless	$198.14
15	535	76	Alize Red Passion	$224.22
15	640	66	Pastry - Banana Tea Loaf	$350.32
16	585	53	Chips - Potato Jalapeno	$489.35

6.

Find products not being bought by customers.	SELECT t1.* FROM products AS t1 LEFT JOIN orders AS t2 ON t1.id=t2.productId WHERE t2.productId IS NULL;	Constructs used: Left join, IS NULL operator

Output:

id	title	price	count	published	createdAt	updatedAt
1	Tomato Paste	$141.79	277	1	2020-03-13	2021-01-06
10	Beef - Cow Feet Split	$433.28	452	1	2021-04-19	2021-06-30
17	Ham - Proscuitto	$149.78	293	0	2021-02-11	2022-05-27
26	Wine - Fontanafredda Barolo	$353.36	241	0	2020-02-18	2022-06-04
29	Roe - White Fish	$102.43	206	0	2021-04-12	2019-12-26
34	Beer - Heinekin	$359.97	346	1	2019-02-06	2020-05-09
36	Wine - Red, Harrow Estates, Cab	$10.23	318	0	2021-09-10	2020-07-30

7.

Find orders whose total amount is in a certain range.	SELECT *, SUM(REPLACE(products.price,'$','') * orders.quantity) AS total_sales FROM orders INNER JOIN products ON orders.productId = products.id GROUP BY orders.userId HAVING SUM(REPLACE(products.price,'$','') * orders.quantity) BETWEEN 500 AND 1000;	Constructs used: Inner join, Sub query, BETWEEN operator

Output:

id	userId	productId	quantity	orderActive	orderCancelled	createdAt	updatedAt	id	title	price	count	published	createdAt	updatedAt	total_sales
2	450	367	24	0	0	2019-04-08	2022-04-29	367	Wine - Magnotta, White	$41.39	140	0	2021-11-23	2020-03-20	993.36
82	583	218	9	0	1	2019-10-09	2022-08-17	218	Cheese - Brie, Triple Creme	$87.49	265	0	2021-08-07	2020-07-26	787.41
107	499	472	2	0	1	2019-12-20	2020-04-25	472	Pepperoni Slices	$445.41	409	0	2020-04-13	2019-08-28	890.82
199	586	373	86	1	1	2021-07-29	2020-07-25	373	Filo Dough	$8.92	443	1	2022-08-28	2022-07-25	767.12
338	779	558	21	1	0	2019-03-20	2021-04-10	558	Lettuce - Belgian Endive	$35.93	196	0	2020-02-07	2021-08-05	754.53
484	567	709	21	1	1	2022-09-09	2021-10-23	709	Shrimp - 16 - 20 Cooked, Peeled	$33.15	106	0	2022-05-01	2020-04-05	696.15
571	626	852	21	1	1	2022-02-09	2019-01-23	852	Lentils - Green, Dry	$40.33	499	0	2020-03-05	2021-06-03	846.93
597	472	55	44	0	1	2022-04-06	2020-09-17	55	Island Oasis - Ice Cream Mix	$17.65	361	1	2020-03-16	2020-11-19	776.5999999999999
612	184	120	2	0	0	2022-03-22	2020-02-02	120	Bread - Roll, Soft White Round	$419.23	175	0	2019-10-13	2021-05-24	838.46
728	793	725	30	1	0	2020-11-10	2021-10-28	725	Cleaner - Pine Sol	$18.34	125	1	2020-11-10	2020-10-13	550.2
734	756	422	18	0	1	2020-10-16	2022-06-16	422	Butter - Salted, Micro	$34.17	335	1	2022-09-27	2019-04-16	615.0600000000001
750	822	383	74	1	1	2019-04-27	2020-03-15	383	Cream - 35%	$7.19	248	1	2020-10-30	2019-08-01	532.0600000000001

8.

Alter table query.	ALTER TABLE orders ADD CONSTRAINT productIdFK FOREIGN KEY (productId) REFERENCES products(id) ON DELETE cascade; ALTER TABLE orders ADD CONSTRAINT userIdFK FOREIGN KEY (userId) REFERENCES users(id) ON DELETE cascade;	Constructs used: Check consistency of DB

Output:

5	16:26:13	ALTER TABLE orders ADD CONSTRAINT productIdFK FOREIGN KEY (productId...	1000 row(s) affected Records: 1000	Duplicates: 0...	0.041 sec
6	16:26:18	ALTER TABLE orders ADD CONSTRAINT userIdFK FOREIGN KEY (userId) REFE...	1000 row(s) affected Records: 1000	Duplicates: 0...	0.031 sec

9.

Find orders made in the last six months.	select * from orders where createdAt > DATE_SUB(now(), INTERVAL 6 MONTH);	Constructs used: Date operation function

Output:

id	userId	productId	quantity	orderActive	orderCancell...	createdAt	updatedAt
12	940	817	19	0	0	2022-08-22	2020-12-13
18	39	90	30	1	0	2022-07-28	2021-04-26
22	18	622	4	0	1	2022-05-29	2022-01-12
25	846	269	8	0	1	2022-07-29	2021-12-28
32	311	742	92	1	1	2022-06-18	2022-10-04
45	678	105	60	1	0	2022-05-06	2020-01-20
49	231	760	47	0	0	2022-07-24	2022-02-12

10.

Update and delete examples.	update products set price="$4.00" where id=2; delete from orders where id=1;	Constructs used: Check consistency of DB

Output:

17	18:20:57	update products set price="$4.00" where id=2	1 row(s) affected Rows matched: 1 Changed: 1 War...	0.0071 sec
18	18:21:01	delete from orders where id=1	0 row(s) affected	0.00100 sec

11.

Demonstrate Commit/Rollback after CRUD operations.	SET autocommit = 0; commit; UPDATE users SET region = "Michigan" WHERE id = 3; SELECT * FROM users; rollback; SELECT * FROM users; SET autocommit = 0; SAVEPOINT s1; SELECT * FROM orders; DELETE FROM orders WHERE id = 1;	Constructs used: Foreign Key Constraints

	SELECT * FROM orders; ROLLBACK TO s1; SELECT * FROM orders; RELEASE SAVEPOINT s1;	

Output:

id	first_name	last_name	phone_number	address	region	email	password	createdAt	updatedAt
3	Nat	Birney	971-644-1319	8447 Sutherland Road	Oregon	nbirney2@engadget.com	tsqC5S	2019-05-06	2019-12-27
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

id	first_name	last_name	phone_number	address	region	email	password	createdAt	updatedAt
3	Nat	Birney	971-644-1319	8447 Sutherland Road	California	nbirney2@engadget.com	tsqC5S	2019-05-06	2019-12-27
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

18:21:21 rollback 0 row(s) affected 0.015 sec

id	first_name	last_name	phone_number	address	region	email	password	createdAt	updatedAt
3	Nat	Birney	971-644-1319	8447 Sutherland Road	Michigan	nbirney2@engadget.com	tsqC5S	2019-05-06	2019-12-27
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

id	first_name	last_name	phone_number	address	region	email	password	createdAt	updatedAt
3	Nat	Birney	971-644-1319	8447 Sutherland Road	California	nbirney2@engadget.com	tsqC5S	2019-05-06	2019-12-27
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

12.

Order/Group products according to region.	SELECT products.id AS products_id, products.title, users.region, products.price, products.count, products.published FROM users INNER JOIN orders ON users.id = orders.userId INNER JOIN products ON orders.productId = products.id ORDER BY region asc; SELECT products.id AS products_id, count(products.id) AS product_count, products.title, users.region, products.price, products.count, products.published FROM users INNER JOIN orders ON users.id = orders.userId INNER JOIN products ON orders.productId = products.id GROUP BY region ORDER BY product_count desc;	Constructs used: Multiway Join, Group by, order by, aggregate functions

	select orders.productId, users.region, count(productId) AS product_count from users join orders on orders.userId=users.id group by users.region;	

Output:

products...	title	region	price	count	published
564	Water - Spring 1.5lit	Alabama	$410.95	294	1
295	Soup - Cream Of Broccoli, Dry	Alabama	$373.71	493	0
479	Sauerkraut	Alabama	$494.95	322	1
44	Bread - 10 Grain	Alabama	$349.00	334	0
758	Chocolate - Chips Compound	Alabama	$254.51	222	1
771	Aspic - Amber	Alabama	$346.29	366	1

13.

Find order total and total amount for users.	SELECT users.id, users.first_name, users.last_name, users.address, products.title, SUM(orders.quantity) AS order_total, SUM(REPLACE(products.price,'$','') * orders.quantity) AS amount_total FROM orders INNER JOIN products ON orders.productId = products.id INNER JOIN users on orders.userId = users.id GROUP BY orders.userId ORDER BY orders.userId;	Constructs used: Multiway Join

Output:

id	first_name	last_name	address	order_total	amount_total
1	Mattias	Pirozzi	8284 Prairie Rose Park	54	21127.5
2	Emelen	Pennicard	782 Daystar Circle	57	10884.15
3	Nat	Birney	8447 Sutherland Road	131	35800.270000000004
4	Rozella	Blackston	89 Sloan Circle	30	1593.9
7	Peggi	Spenley	5475 Debs Alley	115	3450.52

14.

Create a cursor to take a backup of the product table.	DELIMITER //
	CREATE PROCEDURE backupProduct()
	BEGIN
	DECLARE done INT DEFAULT 0;
	DECLARE productID, productCount INTEGER;
	DECLARE productPublished TINYINT;
	DECLARE productTitle, productPrice VARCHAR(100);
	DECLARE createdAt, updatedAt DATE;
	DECLARE cur CURSOR FOR SELECT * FROM products;
	DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
	OPEN cur;
	label: LOOP
	FETCH cur INTO productID, productTitle, productPrice, productCount, productPublished, createdAt, updatedAt;
	INSERT INTO backup VALUES(productID, productTitle, productPrice, productCount, productPublished, createdAt, updatedAt);
	IF done = 1 THEN LEAVE label;
	END IF;
	END LOOP;
	CLOSE cur;
	END//
	DELIMITER ;
	CALL backupProduct();
	SELECT * from backup;
		Constructs used: Select from a view

Output:

id	title	price	count	published	createdAt	updatedAt
1	Tomato Paste	$141.79	277	1	2020-03-13	2021-01-06
2	Bulgar	$4.00	114	0	2021-12-29	2020-01-12
3	Mushrooms - Honey	$419.13	415	1	2019-05-16	2020-10-23
4	Nut - Pistachio, Shelled	$198.18	186	1	2020-02-09	2021-09-06
5	Pasta - Cannelloni, Sheets, Fresh	$187.34	363	1	2022-10-07	2021-09-15
6	Sandwich Wrap	$454.28	312	1	2022-09-03	2022-08-08
7	Syrup - Monin - Granny Smith	$320.69	277	1	2021-05-21	2020-06-26

15. a)

Perform a transaction violating foreign key constraints to check database consistency.	insert into orders (id, userId, productId, quantity, orderActive, orderCancelled, createdAt, updatedAt) values (1001, '2001', '3001', 95, 0, 1, '2020-11-18', '2021-07-26'); update products set count = -1 where id = 2;	Constructs used: Foreign Key Constraints

15. b)

Perform a transaction violating check constraints to check database consistency.	update products set count = -1 where id = 2;	

Output:

Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`shopping_cart`.`orders`, CONSTRAINT `productIdFK` FOREIGN KEY (`productId`) REFERENCES `products` (`id`) ON DELETE CASCADE)

Error Code: 3819. Check constraint 'products_chk_1' is violated.

16.

Insert old users into the archive by using trigger before delete.	create table person_archive (id INT PRIMARY KEY, first_name	Constructs used: trigger before delete

	VARCHAR(50), last_name VARCHAR(50), phone_number VARCHAR(50), address VARCHAR(50), region VARCHAR(50), email VARCHAR(50), password VARCHAR(50), createdAt DATE, updatedAt DATE); delimiter // CREATE TRIGGER user_bd BEFORE DELETE ON users FOR EACH ROW insert into person_archive(id, first_name, last_name, phone_number, address, region, email, password, createdAt, updatedAt) values (old.id, old.first_name, old.last_name, old.phone_number, old.address, old.region, old.email, old.password, old.createdAt, old.updatedAt); // delimiter ; -- test DELETE FROM users WHERE id = 1; SELECT * FROM person_archive;	

Output:

id	first_name	last_name	phone_number	address	region	email	password	createdAt	updatedAt
1	Mattias	Pirozzi	309-377-0807	8284 Prairie Rose Park	Illinois	mpirozzi0@google.ru	EKiHxLkNn	2019-11-29	2019-06-23
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

17.

Trigger error if quantity in ordercarts is greater than count in products. (trigger before insert).	delimiter //
	create trigger before_insert_order before insert on orders for each row
	begin
	DECLARE c INT;
	SET c = (select count from products where id=new.productId);
	if(new.quantity > c) then SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'not enough units of this product in stock. Pls reduce quantity and try again.';
	end if;
	end; //
	delimiter ;
	
	insert into orders (id, userId, productId, quantity, orderActive, orderCancelled, createdAt, updatedAt) values (1005, '170', '1', 95, 0, 1, '2020-11-18', '2021-07-26');
		Constructs used: Trigger before insert

Output:

74... 17:34:08 in Error Code: 1644. Not enough units of this product in stock. Please reduce quantity and try again.

18.

Find repeated purchases.	SELECT orders.productId, products.title, count(orders.productId) AS	Constructs used: Multiway Join, Group by, order by, aggregate functions

	product_count, products.price, products.count FROM orders JOIN products ON orders.productId = products.id GROUP BY orders.productId HAVING count(orders.productId) > 1 ORDER BY orders.productId;	

Output:

productId	title	product_co...	price	count
5	Pasta - Cannelloni, Sheets, Fresh	2	$187.34	363
12	Pork - European Side Bacon	2	$94.08	393
14	Beef - Bones, Marrow	2	$193.31	181
18	Soup - Campbells Bean Medley	2	$323.06	269
19	Rice - Aborio	2	$188.55	203
20	Oil - Cooking Spray	3	$332.57	325

19.

Find products frequently being purchased from a locality.	SELECT products.id, count(products.id) AS product_count, products.title, users.region, products.price, products.count, products.published FROM users INNER JOIN orders ON users.id = orders.userId INNER JOIN products ON orders.productId = products.id GROUP BY products.id, region HAVING count(products.id) > 1 ORDER BY region asc;	Constructs used: Multiway Join, Group by, order by, aggregate functions

Output:

id	product_co...	title	region	price	count	published
682	2	Appetizer - Veg Assortment	California	$495.37	152	1
84	3	Oil - Cooking Spray	California	$489.80	389	1
428	2	Pasta - Bauletti, Chicken White	California	$185.48	421	0
383	2	Cream - 35%	Florida	$7.19	248	1
667	2	Pickerel - Fillets	Florida	$396.24	382	0

20.

Check inventory at the end of each month.	select b.x-a.x from inventory b inner join inventory a on b.year=a.year and b.month=a.month-1;	Constructs used: Join

Output:

b.x-a.x
8
-9
2
18
5
1
-7
-13

21.

Create a view for inventory.	create view inventory as select count(id) as x, YEAR(createdAt) as year, MONTH(createdAt) as month from products group by YEAR(createdAt), MONTH(createdAt); select * from inventory;	Constructs used: creating view

Output:

x	year	month
26	2020	3
15	2019	5
28	2020	2
7	2022	10
25	2022	9
24	2021	5
19	2019	7
22	2020	12

22.

Trigger: update count in products after insertion in orders.	create trigger after_insert_order after insert on orders for each row update products set count = count-new.quantity where id = new.productId; select * from products where id = 1; insert into orders (id, userId, productId, quantity, orderActive, orderCancelled, createdAt, updatedAt) values (1005, '170', '1', 1, 0, 1, '2020-11-18', '2021-07-26'); select * from products where id = 1;	Constructs used: Trigger after insert

Output:

id	title	price	count	published	createdAt	updatedAt
1	Tomato Paste	$141.79	276	1	2020-03-13	2021-01-06
NULL	NULL	NULL	NULL	NULL	NULL	NULL

id	title	price	count	published	createdAt	updatedAt
1	Tomato Paste	$141.79	271	1	2020-03-13	2021-01-06
NULL	NULL	NULL	NULL	NULL	NULL	NULL

23.

Create results of joining users and orders to find active orders as a view..	create view active_orders as select o.id, o.userId, u.first_name, u.last_name, u.region, o.orderActive from orders o, users u where orderActive=1 and o.userId=u.id; select * from active_orders;	Constructs used: join and create a view

Output:

id	userId	first_name	last_name	region	orderActive
3	290	Farlee	Nixon	Florida	1
6	180	Seumas	Aland	Massachusetts	1
7	853	Clemmy	Persian	Texas	1
9	965	Benson	Driuzzi	Tennessee	1

24.

Trigger update of active_orders view when an order is delivered.	delimiter //
	create trigger afer_update_orders
	after update on orders
	for each row
	begin
	if (old.orderActive ='1' and new.orderActive = '0') then
	delete from active_orders where id=old.id;
	end if;
	end; //
	delimiter ;
		Constructs used: Trigger after update

	select * from active_orders; update orders set orderActive=0 where id=3; select * from active_orders where id=3;	

Output:

id	userId	first_name	last_name	region	orderActive
3	290	Farlee	Nixon	Florida	1
6	180	Seumas	Aland	Massachusetts	1
7	853	Clemmy	Persian	Texas	1
9	965	Benson	Driuzzi	Tennessee	1

id	userId	first_name	last_name	region	orderActive
3	290	Farlee	Nixon	Florida	1

25.

Join products and users to see who purchased what items.	SELECT users.id AS user_id, users.first_name, users.last_name, users.region, products.id AS product_id, products.title, products.price, products.count FROM users INNER JOIN orders ON users.id = orders.userId INNER JOIN products ON orders.productId = products.id ORDER BY orders.userId;	Constructs used: Join, Order by

Output:

user_id	first_name	last_name	region	product_...	title	price	count
1	Mattias	Pirozzi	Illinois	525	Napkin - Dinner, White	$391.25	199
2	Emelen	Pennicard	Michigan	966	Tuna - Sushi Grade	$190.95	453
3	Nat	Birney	Michigan	432	Rice - 7 Grain Blend	$296.31	252
3	Nat	Birney	Michigan	754	Wasabi Paste	$251.29	285
4	Rozella	Blackston	Florida	207	Beef - Tongue, Fresh	$53.13	480

26.

Group orders by date purchased.	SELECT users.id AS user_id, users.first_name, users.last_name, users.region, products.id AS product_id, products.title, products.price, products.count, orders.createdAt, orders.updatedAt FROM users INNER JOIN orders ON users.id = orders.userId INNER JOIN products ON orders.productId = products.id ORDER BY YEAR(orders.createdAt) desc, MONTH(orders.createdAt) desc;	Construct used: group by, join, order by

Output:

user_id	first_name	last_name	region	product_...	title	price	count	createdAt	updatedAt
447	Nicky	Gorring	Texas	400	Mushroom - Lg - Cello	$398.73	490	2022-10-11	2022-01-08
445	Donnie	Coveley	Texas	418	Bread - Bistro Sour	$138.47	415	2022-10-01	2021-11-24
401	Cassius	Briffett	Missouri	337	Food Colouring - Pink	$334.87	402	2022-10-10	2021-09-22
996	Florina	Axtonne	Kentucky	420	Pastrami	$149.99	139	2022-10-08	2022-10-14
110	Valida	Lyver	Texas	191	Spinach - Spinach Leaf	$415.43	331	2022-10-02	2020-09-26

27.

Group orders by “is active” in ordercarts using active_orders view.	select region, count(*) as numberOfActiveOrders from active_orders group by region order by region;	Constructs used: Group by

Output:

region	numberOfActiveOrd...
Alabama	6
Alaska	1
Arizona	9
Arkansas	1
California	64
Colorado	20
Connecticut	3
Delaware	3

28.

Group orders created at the same date.	select o1.id as OrderId, o1.userId as Customer, o1.productId as Product, o1.quantity as quantity, o1.orderActive as Delivered, o1.orderCancelled as Cancelled, o1.createdAt as OrderDate, o2.id as OrderId, o2.userId as Customer, o2.productId as Product, o2.quantity as quantity, o2.orderActive as Delivered, o1.orderCancelled as Cancelled, o2.createdAt as OrderDate from orders o1, orders o2 where o1.createdAt = o2.createdAt and o1.userId <> o2.userId group by o1.createdAt order by o1.createdAt;	Constructs used: Self Join

Output:

OrderId	Customer	Product	quantity	Delivered	Cancelled	OrderDate	OrderId	Customer	Product	quantity	Delivered	Cancelled	OrderDate
590	739	902	82	0	0	2019-01-02	177	316	443	97	1	0	2019-01-02
655	333	975	48	1	0	2019-01-03	586	935	66	92	1	0	2019-01-03
589	910	745	58	1	0	2019-01-06	121	648	473	19	1	0	2019-01-06
837	788	957	87	0	1	2019-01-07	811	53	773	96	1	1	2019-01-07
687	357	208	45	0	1	2019-01-18	394	795	347	20	0	1	2019-01-18
233	890	825	90	0	0	2019-01-24	120	866	505	98	0	0	2019-01-24
983	137	55	99	0	0	2019-02-02	906	431	914	16	0	0	2019-02-02
967	165	540	37	1	0	2019-02-03	520	17	298	80	0	0	2019-02-03
273	949	554	41	0	0	2019-02-05	33	820	756	11	0	0	2019-02-05
928	738	104	9	0	0	2019-02-07	498	383	192	34	0	0	2019-02-07
936	292	419	23	0	1	2019-02-10	578	565	81	24	1	1	2019-02-10
755	610	279	8	1	0	2019-02-11	37	867	875	29	1	0	2019-02-11
708	93	237	58	1	1	2019-03-05	442	154	716	60	1	1	2019-03-05