CASE STUDY

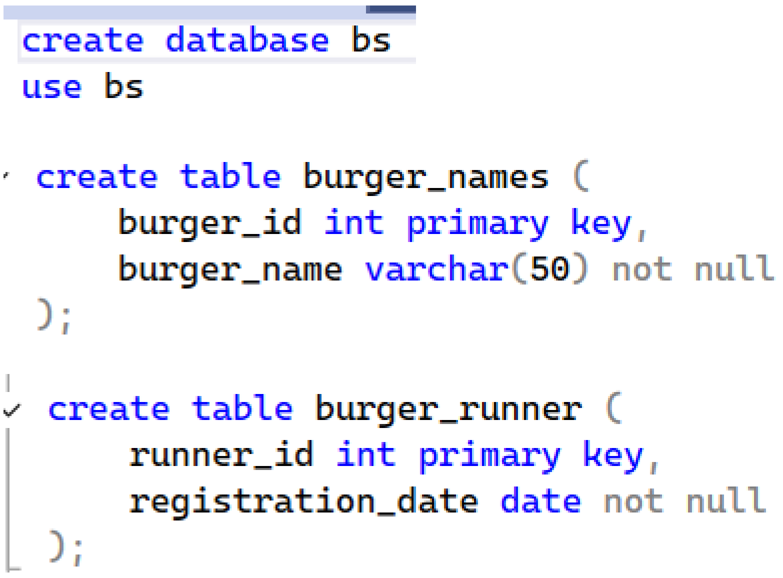
BURGER BASH

INTRODUCTION:

I have a started a new business of selling burger because I read on my Instagram feed that 'Burger Is the Future!

But I knew that burger alone was not going to help me get seed funding to expand my new Burger Empire - so I had one more genius idea to combine with it - I was going to Uberize it - and so Burger Runner was launched!

I started by recruiting "runners" to deliver fresh burger from Burger Runner Headquarters and also maxed out my credit card to pay freelance developers to build a mobile app to accept orders from customers.



```
create database bs
use bs

create table burger_names (
     burger_id int primary key,
     burger_name varchar(50) not null
);

create table burger_runner (
     runner_id int primary key,
     registration_date date not null
);
```

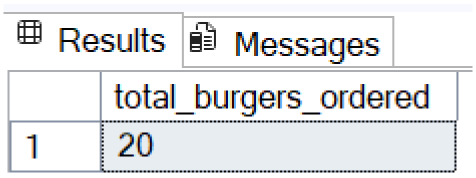
```sql
create table customer_orders (
    order_id int primary key,
    customer_id int not null,
    burger_id int not null,
    exclusions varchar(100) not null,
    extras varchar(100) not null,
    order_time datetime not null,
    foreign key (burger_id) references burger_names(burger_id)
);

create table runner_orders (
    order_id int primary key,
    runner_id int not null,
    pickup_time datetime not null,
    distance varchar(20) not null,
    duration varchar(20) not null,
    cancellation varchar(100) not null,
    foreign key (runner_id) references burger_runner(runner_id),
    foreign key (order_id) references customer_orders(order_id)
);

insert into burger_names (burger_id, burger_name) values
(1, 'veggie delight'),
(2, 'chicken supreme'),
(3, 'paneer tikka burger'),
(4, 'mushroom melt');


insert into burger_runner (runner_id, registration_date) values
(101, '2023-06-01'),
(102, '2023-06-02'),
(103, '2023-06-03'),
(104, '2023-06-04'),
(105, '2023-06-05'),
(106, '2023-06-06'),
(107, '2023-06-07'),
(108, '2023-06-08'),
(109, '2023-06-09'),
(110, '2023-06-10');
```

```sql
insert into customer_orders (order_id, customer_id, burger_id, exclusions, extras, order_time) values
(201, 1001, 1, 'no onions', 'extra cheese', '2023-07-01 12:00:00'),
(202, 1002, 2, 'no mayo', 'double patty', '2023-07-01 12:10:00'),
(203, 1003, 3, 'no lettuce', 'cheese burst', '2023-07-01 12:20:00'),
(204, 1004, 4, 'no tomato', 'extra mushroom', '2023-07-01 12:30:00'),
(205, 1005, 2, 'no pickles', 'extra spicy', '2023-07-02 13:00:00'),
(206, 1006, 1, 'no cheese', 'jalapenos', '2023-07-02 13:15:00'),
(207, 1007, 3, 'no mustard', 'crispy onions', '2023-07-02 13:30:00'),
(208, 1008, 4, 'no ketchup', 'extra paneer', '2023-07-02 14:00:00'),
(209, 1009, 2, 'no tomato', 'fried egg', '2023-07-02 14:15:00'),
(210, 1010, 1, 'no sauce', 'extra lettuce', '2023-07-02 14:30:00'),
(211, 1011, 4, 'no jalapenos', 'cheese dip', '2023-07-03 11:00:00'),
(212, 1012, 3, 'no mayo', 'spicy sauce', '2023-07-03 11:10:00'),
(213, 1013, 1, 'no onions', 'vegan cheese', '2023-07-03 11:20:00'),
(214, 1014, 2, 'no cucumber', 'grilled patty', '2023-07-03 11:30:00'),
(215, 1015, 4, 'no olives', 'cream cheese', '2023-07-03 11:40:00'),
(216, 1016, 2, 'no capsicum', 'stuffed patty', '2023-07-03 11:50:00'),
(217, 1017, 3, 'no lettuce', 'mint mayo', '2023-07-03 12:00:00'),
(218, 1018, 1, 'no tomato', 'double paneer', '2023-07-03 12:10:00'),
(219, 1019, 4, 'no cheese', 'extra garlic', '2023-07-03 12:20:00'),
(220, 1020, 2, 'no jalapenos', 'extra chicken', '2023-07-03 12:30:00');
```

```sql
insert into runner_orders (order_id, runner_id, pickup_time, distance, duration, cancellation) values
(201, 101, '2023-07-01 12:05:00', '2.5km', '15min', 'none'),
(202, 102, '2023-07-01 12:15:00', '3.0km', '18min', 'none'),
(203, 103, '2023-07-01 12:25:00', '1.8km', '12min', 'none'),
(204, 104, '2023-07-01 12:35:00', '2.2km', '14min', 'customer unavailable'),
(205, 105, '2023-07-02 13:05:00', '3.5km', '20min', 'none'),
(206, 106, '2023-07-02 13:20:00', '2.0km', '13min', 'none'),
(207, 107, '2023-07-02 13:35:00', '1.6km', '11min', 'none'),
(208, 108, '2023-07-02 14:05:00', '3.2km', '19min', 'runner cancelled'),
(209, 109, '2023-07-02 14:20:00', '2.8km', '16min', 'none'),
(210, 110, '2023-07-02 14:35:00', '2.4km', '15min', 'none'),
(211, 101, '2023-07-03 11:05:00', '2.0km', '13min', 'none'),
(212, 102, '2023-07-03 11:15:00', '2.3km', '14min', 'none'),
(213, 103, '2023-07-03 11:25:00', '2.1km', '15min', 'none'),
(214, 104, '2023-07-03 11:35:00', '1.9km', '12min', 'none'),
(215, 105, '2023-07-03 11:45:00', '3.0km', '17min', 'customer not home'),
(216, 106, '2023-07-03 11:55:00', '2.7km', '16min', 'none'),
(217, 107, '2023-07-03 12:05:00', '2.2km', '14min', 'none'),
(218, 108, '2023-07-03 12:15:00', '3.3km', '20min', 'none'),
(219, 109, '2023-07-03 12:25:00', '1.8km', '11min', 'none'),
(220, 110, '2023-07-03 12:35:00', '2.9km', '18min', 'none');
```

-- 1. Count how many burgers were ordered

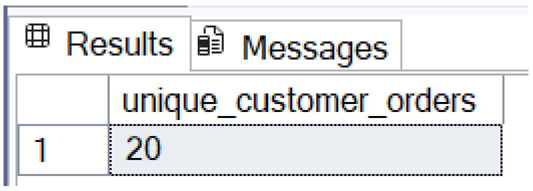
select count(*) as total_burgers_ordered from customer_orders;



	total_burgers_ordered
1	20

-- 2. Count how many unique customer orders were made

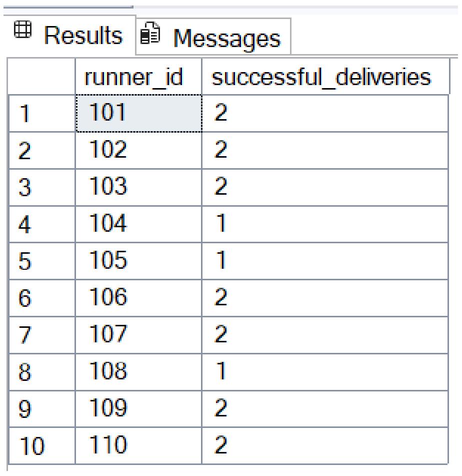
select count(distinct order_id) as unique_customer_orders from customer_orders;



	unique_customer_orders
1	20

-- 3. Count how many successful orders were delivered by each runner

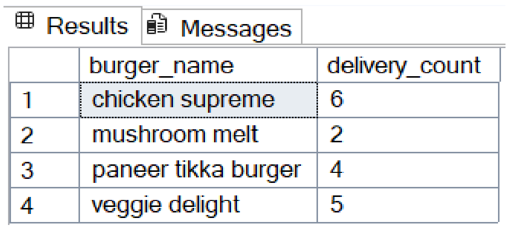
select runner_id, count(*) as successful_deliveries from runner_orders where cancellation is null or lower(cancellation) = 'none' group by runner_id;



	runner_id	successful_deliveries
1	101	2
2	102	2
3	103	2
4	104	1
5	105	1
6	106	2
7	107	2
8	108	1
9	109	2
10	110	2

-- 4. Count how many of each type of burger was delivered

select b.burger_name, count(*) as delivery_count from customer_orders c join runner_orders r on c.order_id = r.order_id join burger_names b on c.burger_id = b.burger_id where r.cancellation is null or lower(r.cancellation) = 'none' group by b.burger_name;

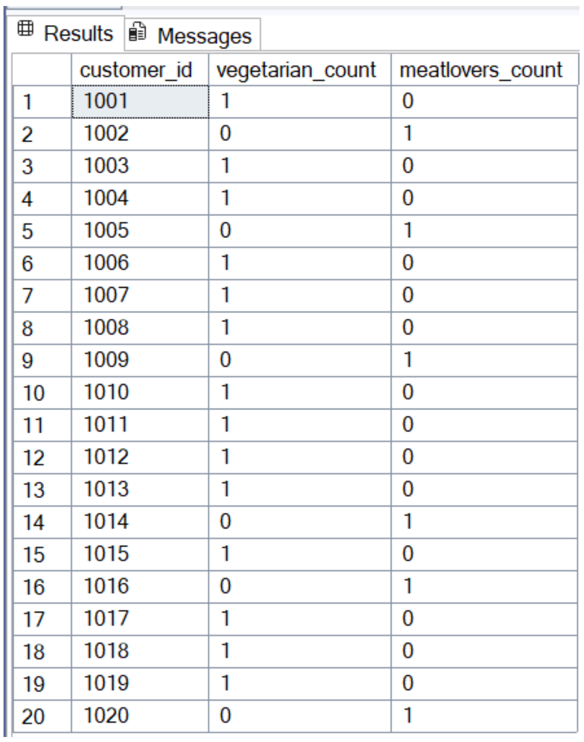


Results | Messages

	burger_name	delivery_count
1	chicken supreme	6
2	mushroom melt	2
3	paneer tikka burger	4
4	veggie delight	5

-- 5. Count how many Vegetarian and Meatlovers burgers were ordered by each customer

select customer_id, sum(case when burger_id in (1, 3, 4) then 1 else 0 end) as vegetarian_count, sum(case when burger_id = 2 then 1 else 0 end) as meatlovers_count from customer_orders group by customer_id;

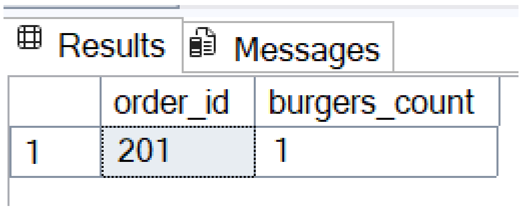


Results | Messages

	customer_id	vegetarian_count	meatlovers_count
1	1001	1	0
2	1002	0	1
3	1003	1	0
4	1004	1	0
5	1005	0	1
6	1006	1	0
7	1007	1	0
8	1008	1	0
9	1009	0	1
10	1010	1	0
11	1011	1	0
12	1012	1	0
13	1013	1	0
14	1014	0	1
15	1015	1	0
16	1016	0	1
17	1017	1	0
18	1018	1	0
19	1019	1	0
20	1020	0	1

-- 6. Get the maximum number of burgers delivered in a single order

select top 1 c.order_id, count(*) as burgers_count from customer_orders c join runner_orders r on c.order_id = r.order_id where r.cancellation is null or lower(r.cancellation) = 'none' group by c.order_id order by burgers_count desc;

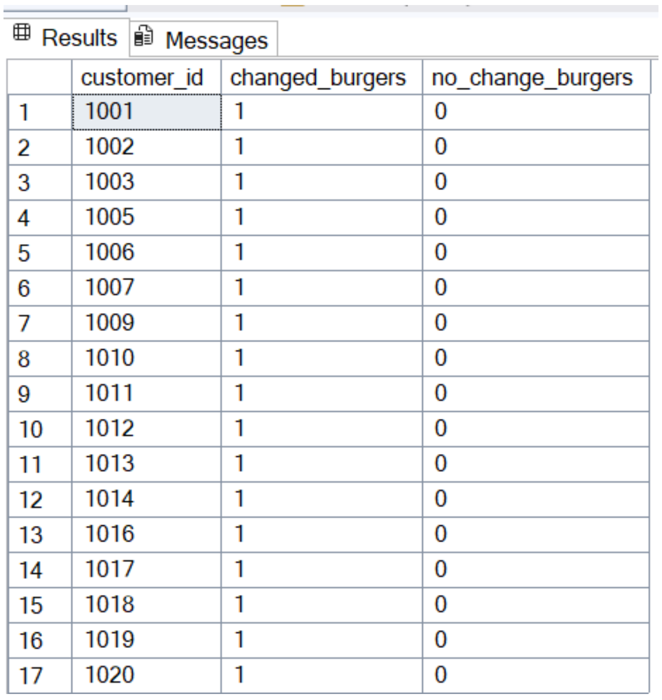


Results | Messages

	order_id	burgers_count
1	201	1

-- 7. For each customer, count how many delivered burgers had at least 1 change and how many had no changes

select customer_id, sum(case when (exclusions is not null and exclusions <> '') or (extras is not null and extras <> '') then 1 else 0 end) as changed_burgers, sum(case when (exclusions is null or exclusions = '') and (extras is null or extras = '') then 1 else 0 end) as no_change_burgers from customer_orders c join runner_orders r on c.order_id = r.order_id where r.cancellation is null or lower(r.cancellation) = 'none' group by customer_id;

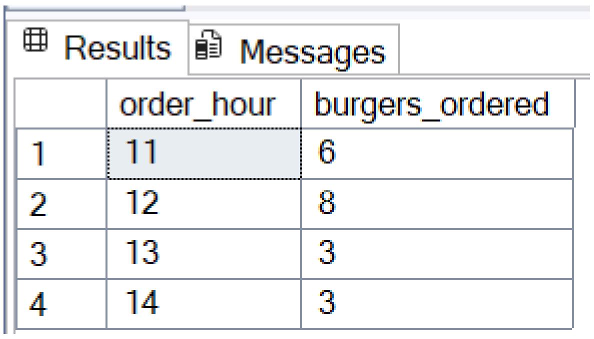


Results | Messages

	customer_id	changed_burgers	no_change_burgers
1	1001	1	0
2	1002	1	0
3	1003	1	0
4	1005	1	0
5	1006	1	0
6	1007	1	0
7	1009	1	0
8	1010	1	0
9	1011	1	0
10	1012	1	0
11	1013	1	0
12	1014	1	0
13	1016	1	0
14	1017	1	0
15	1018	1	0
16	1019	1	0
17	1020	1	0

-- 8. Show the total volume of burgers ordered for each hour of the day

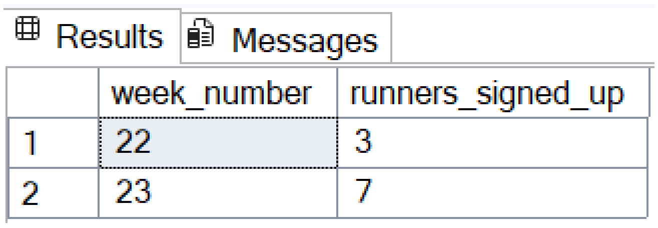
select datepart(hour, order_time) as order_hour, count(*) as burgers_ordered from customer_orders group by datepart(hour, order_time) order by order_hour;



	order_hour	burgers_ordered
1	11	6
2	12	8
3	13	3
4	14	3

-- 9. Show how many runners signed up for each 1 week period

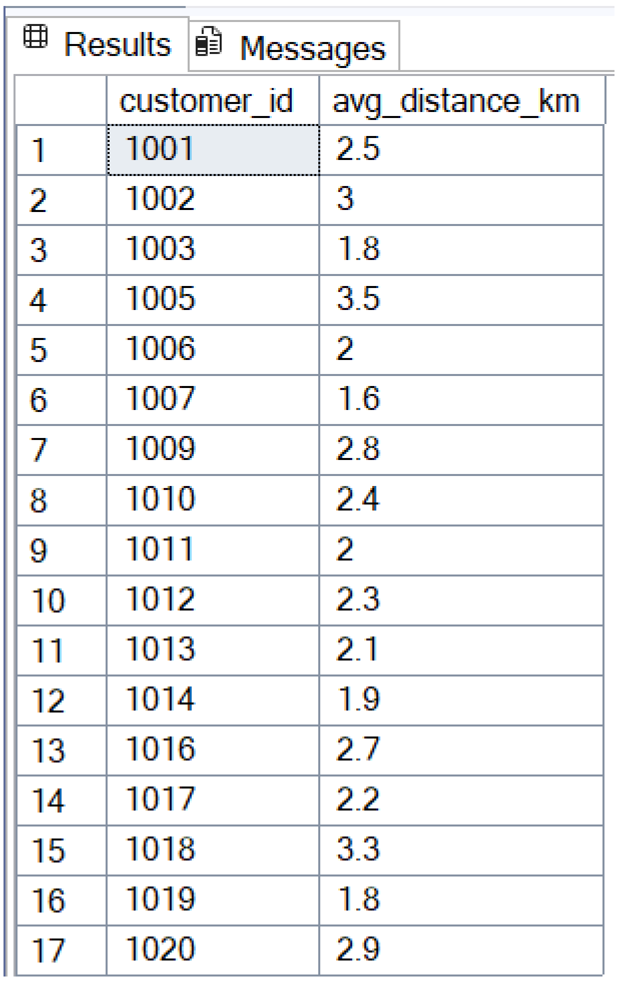
select datepart(week, registration_date) as week_number, count(*) as runners_signed_up from burger_runner group by datepart(week, registration_date) order by week_number;



	week_number	runners_signed_up
1	22	3
2	23	7

-- 10. Get the average distance travelled for each customer

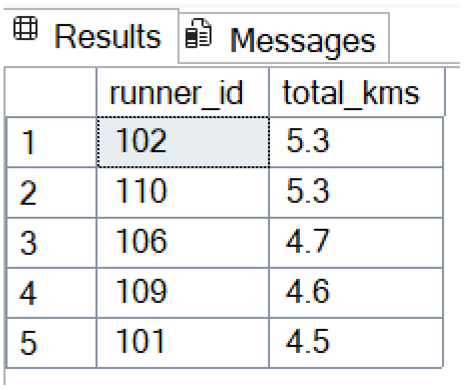
select c.customer_id, avg(cast(replace(r.distance, 'km', '') as float)) as avg_distance_km from customer_orders c join runner_orders r on c.order_id = r.order_id where r.cancellation is null or lower(r.cancellation) = 'none' group by c.customer_id;



	customer_id	avg_distance_km
1	1001	2.5
2	1002	3
3	1003	1.8
4	1005	3.5
5	1006	2
6	1007	1.6
7	1009	2.8
8	1010	2.4
9	1011	2
10	1012	2.3
11	1013	2.1
12	1014	1.9
13	1016	2.7
14	1017	2.2
15	1018	3.3
16	1019	1.8
17	1020	2.9

-- Complex Query 1: Most active runners with max total distance

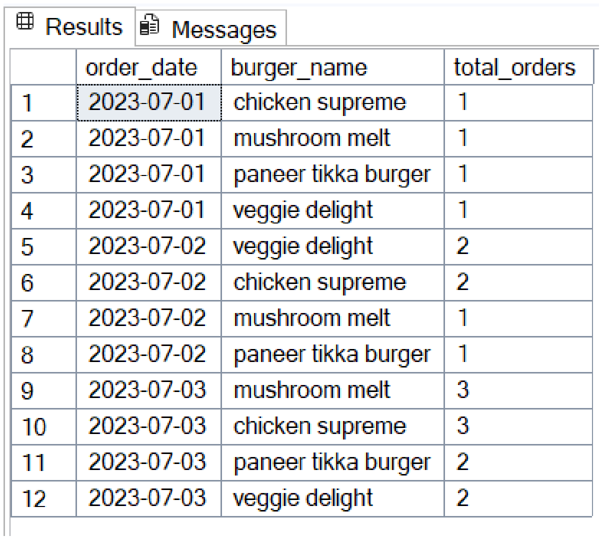
SELECT TOP 5 r.runner_id, SUM(CAST(REPLACE(r.distance, 'km', '') AS FLOAT)) AS total_kms FROM runner_orders r WHERE r.cancellation IS NULL OR LOWER(r.cancellation) = 'none' GROUP BY r.runner_id ORDER BY total_kms DESC;


Results | Messages

	runner_id	total_kms
1	102	5.3
2	110	5.3
3	106	4.7
4	109	4.6
5	101	4.5

-- Complex Query 2: Burger type popularity per day

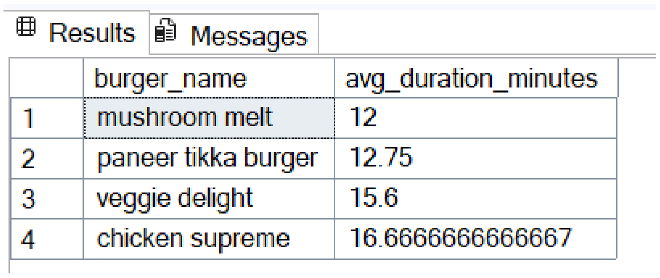
SELECT CAST(order_time AS DATE) AS order_date, b.burger_name, COUNT(*) AS total_orders FROM customer_orders c JOIN burger_names b ON c.burger_id = b.burger_id GROUP BY CAST(order_time AS DATE), b.burger_name ORDER BY order_date, total_orders DESC;


Results | Messages

	order_date	burger_name	total_orders
1	2023-07-01	chicken supreme	1
2	2023-07-01	mushroom melt	1
3	2023-07-01	paneer tikka burger	1
4	2023-07-01	veggie delight	1
5	2023-07-02	veggie delight	2
6	2023-07-02	chicken supreme	2
7	2023-07-02	mushroom melt	1
8	2023-07-02	paneer tikka burger	1
9	2023-07-03	mushroom melt	3
10	2023-07-03	chicken supreme	3
11	2023-07-03	paneer tikka burger	2
12	2023-07-03	veggie delight	2

-- Complex Query 3: Average duration of delivery by burger type (only successful)

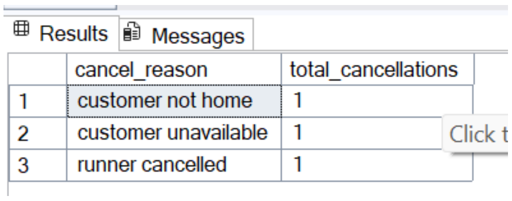
SELECT b.burger_name, AVG(CAST(REPLACE(r.duration, 'min', '') AS FLOAT)) AS avg_duration_minutes FROM customer_orders c JOIN runner_orders r ON c.order_id = r.order_id JOIN burger_names b ON c.burger_id = b.burger_id WHERE r.cancellation IS NULL OR LOWER(r.cancellation) = 'none' GROUP BY b.burger_name ORDER BY avg_duration_minutes;



	burger_name	avg_duration_minutes
1	mushroom melt	12
2	paneer tikka burger	12.75
3	veggie delight	15.6
4	chicken supreme	16.6666666666667

-- Complex Query 4: Number of cancelled orders by reason

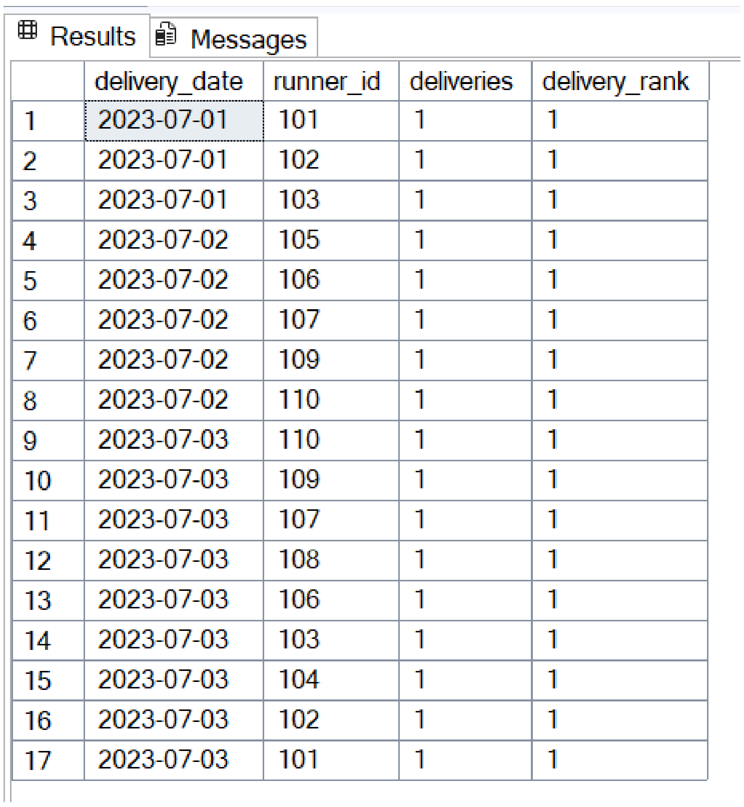
SELECT LOWER(cancellation) AS cancel_reason, COUNT(*) AS total_cancellations FROM runner_orders WHERE cancellation IS NOT NULL AND LOWER(cancellation) <> 'none' GROUP BY LOWER(cancellation);



	cancel_reason	total_cancellations
1	customer not home	1
2	customer unavailable	1
3	runner cancelled	1

-- Complex Query 5: Rank runners by number of deliveries per day

SELECT CAST(pickup_time AS DATE) AS delivery_date, runner_id, COUNT(*) AS deliveries, RANK() OVER (PARTITION BY CAST(pickup_time AS DATE) ORDER BY COUNT(*) DESC) AS delivery_rank FROM runner_orders WHERE cancellation IS NULL OR LOWER(cancellation) = 'none' GROUP BY CAST(pickup_time AS DATE), runner_id ORDER BY delivery_date, delivery_rank;



Results | Messages

	delivery_date	runner_id	deliveries	delivery_rank
1	2023-07-01	101	1	1
2	2023-07-01	102	1	1
3	2023-07-01	103	1	1
4	2023-07-02	105	1	1
5	2023-07-02	106	1	1
6	2023-07-02	107	1	1
7	2023-07-02	109	1	1
8	2023-07-02	110	1	1
9	2023-07-03	110	1	1
10	2023-07-03	109	1	1
11	2023-07-03	107	1	1
12	2023-07-03	108	1	1
13	2023-07-03	106	1	1
14	2023-07-03	103	1	1
15	2023-07-03	104	1	1
16	2023-07-03	102	1	1
17	2023-07-03	101	1	1