Endpoint Indexing

Users

@router.get("/users/", tags=["users"])

```
-- Before indexes
"Limit  (cost=0.00..1072.30 rows=5 width=32) (actual time=0.141..1.068 rows=5 loops=1)"
"  ->  Seq Scan on ""user""  (cost=0.00..21446.00 rows=100 width=32) (actual time=0.140..1.067 rows=5 loops=1)"
"        Filter: (name ~~ 'oliver%'::text)"
"        Rows Removed by Filter: 11341"
"Planning Time: 0.073 ms"
"Execution Time: 1.081 ms"

-- Index done
CREATE INDEX idx_user_name ON public."user" (name text_pattern_ops);

-- After indexes
"Limit  (cost=0.42..0.83 rows=5 width=32) (actual time=0.066..0.103 rows=5 loops=1)"
"  ->  Index Scan using idx_user_name on ""user""  (cost=0.42..8.45 rows=100 width=32) (actual time=0.064..0.101 rows=5 loops=1)"
"        Index Cond: ((name ~>=~ 'oliver'::text) AND (name ~<~ 'olives'::text))"
"        Filter: (name ~~ 'oliver%'::text)"
"Planning Time: 0.842 ms"
"Execution Time: 0.114 ms"
```

The index was added on the `name` column of the users table because the endpoint could either list all users or filter by name. Since in the endpoint, the comparison for all names were done using the `LIKE` operator, we added `text_pattern_ops` for name comparisons.

@router.get("/users/{user_id}/", tags=["users"])

```
-- Before indexes
"Nested Loop  (cost=41608.32..78948.89 rows=1 width=64) (actual time=274.944..275.007 rows=1 loops=1)"
"  Join Filter: (""user"".user_id = user_1.user_id)"
"  ->  GroupAggregate  (cost=1000.42..13569.91 rows=1 width=64) (actual time=36.655..36.694 rows=1 loops=1)"
"        Group Key: ""user"".user_id"
"        ->  Nested Loop Left Join  (cost=1000.42..13569.89 rows=1 width=37) (actual time=1.731..36.686 rows=1 loops=1)"
"              Join Filter: (deposit.user_id = ""user"".user_id)"
"              ->  Index Scan using user_pkey on ""user""  (cost=0.42..8.44 rows=1 width=32) (actual time=0.010..0.013 rows=1 loops=1)"
"                    Index Cond: (user_id = 20000)"
"              ->  Gather  (cost=1000.00..13561.43 rows=1 width=13) (actual time=1.719..36.670 rows=1 loops=1)"
"                    Workers Planned: 2"
"                    Workers Launched: 2"
"                    ->  Parallel Seq Scan on deposit  (cost=0.00..12561.33 rows=1 width=13) (actual time=10.136..21.746 rows=0 loops=3)"
"                          Filter: (user_id = 20000)"
"                          Rows Removed by Filter: 333333"
"  ->  GroupAggregate  (cost=40607.89..65378.95 rows=1 width=64) (actual time=238.283..238.306 rows=1 loops=1)"
"        Group Key: user_1.user_id"
"        ->  Nested Loop Left Join  (cost=40607.89..65378.93 rows=1 width=37) (actual time=152.946..238.296 rows=1 loops=1)"
"              Join Filter: (category.user_id = user_1.user_id)"
"              ->  Index Scan using user_pkey on ""user"" user_1  (cost=0.42..8.44 rows=1 width=32) (actual time=0.019..0.022 rows=1 loops=
"                    Index Cond: (user_id = 20000)"
"              ->  Hash Right Join  (cost=40607.47..65370.48 rows=1 width=13) (actual time=152.924..238.271 rows=1 loops=1)"
"                    Hash Cond: (item.expense_id = expense.expense_id)"
"                    ->  Seq Scan on item  (cost=0.00..21013.00 rows=1000000 width=13) (actual time=0.023..40.469 rows=1000000 loops=1)"
"                    ->  Hash  (cost=40607.46..40607.46 rows=1 width=16) (actual time=150.911..150.933 rows=1 loops=1)"
"                          Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"                          ->  Hash Right Join  (cost=16595.45..40607.46 rows=1 width=16) (actual time=41.301..150.925 rows=1 loops=1)"
"                                Hash Cond: (expense.category_id = category.category_id)"
"                                ->  Seq Scan on expense  (cost=0.00..21387.00 rows=1000000 width=16) (actual time=0.022..66.525 rows=10000
"                                ->  Hash  (cost=16595.43..16595.43 rows=1 width=16) (actual time=38.924..38.945 rows=1 loops=1)"
"                                      Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"                                      ->  Gather  (cost=1000.00..16595.43 rows=1 width=16) (actual time=1.787..38.939 rows=1 loops=1)"
"                                            Workers Planned: 2"
"                                            Workers Launched: 2"
"                                            ->  Parallel Seq Scan on category  (cost=0.00..15595.33 rows=1 width=16) (actual time=12.775..
"                                                  Filter: (user_id = 20000)"
"                                                  Rows Removed by Filter: 333333"
"Planning Time: 0.371 ms"
"Execution Time: 275.059 ms"
```

```
-- Index done
CREATE INDEX idx_deposit_user_id ON deposit(user_id);
CREATE INDEX idx_category_user_id ON category(user_id);
CREATE INDEX idx_category_id ON category(category_id);
CREATE INDEX idx_item_expense_id ON item(expense_id);
CREATE INDEX idx_expense_category_id ON expense(category_id);

-- After indexes
"Nested Loop  (cost=2.55..42.83 rows=1 width=64) (actual time=0.300..0.301 rows=1 loops=1)"
"  Join Filter: (""user"".user_id = user_1.user_id)"
"  ->  GroupAggregate  (cost=0.85..16.92 rows=1 width=64) (actual time=0.108..0.108 rows=1 loops=1)"
"        Group Key: ""user"".user_id"
"        ->  Nested Loop Left Join  (cost=0.85..16.90 rows=1 width=37) (actual time=0.098..0.100 rows=1 loops=1)"
"              Join Filter: (deposit.user_id = ""user"".user_id)"
"              ->  Index Scan using user_pkey on ""user""  (cost=0.42..8.44 rows=1 width=32) (actual time=0.016..0.017 rows=1 loops=1)"
"                    Index Cond: (user_id = 20000)"
"              ->  Index Scan using idx_deposit_user_id on deposit  (cost=0.42..8.44 rows=1 width=13) (actual time=0.078..0.079 rows=1 loop
"                    Index Cond: (user_id = 20000)"
"  ->  GroupAggregate  (cost=1.70..25.88 rows=1 width=64) (actual time=0.188..0.188 rows=1 loops=1)"
"        Group Key: user_1.user_id"
"        ->  Nested Loop Left Join  (cost=1.70..25.86 rows=1 width=37) (actual time=0.183..0.185 rows=1 loops=1)"
"              ->  Nested Loop Left Join  (cost=1.27..25.35 rows=1 width=40) (actual time=0.137..0.138 rows=1 loops=1)"
"                    ->  Nested Loop Left Join  (cost=0.85..16.90 rows=1 width=40) (actual time=0.091..0.092 rows=1 loops=1)"
"                          Join Filter: (category.user_id = user_1.user_id)"
"                          ->  Index Scan using user_pkey on ""user"" user_1  (cost=0.42..8.44 rows=1 width=32) (actual time=0.012..0.012 r
"                                Index Cond: (user_id = 20000)"
"                          ->  Index Scan using idx_category_user_id on category  (cost=0.42..8.44 rows=1 width=16) (actual time=0.077..0.0
"                                Index Cond: (user_id = 20000)"
"                    ->  Index Scan using idx_expense_category_id on expense  (cost=0.42..8.44 rows=1 width=16) (actual time=0.044..0.044 r
"                          Index Cond: (category_id = category.category_id)"
"              ->  Index Scan using idx_item_expense_id on item  (cost=0.42..0.50 rows=1 width=13) (actual time=0.044..0.045 rows=1 loops=1
"                    Index Cond: (expense_id = expense.expense_id)"
"Planning Time: 4.556 ms"
"Execution Time: 0.369 ms"
```

The indexes added were for the `deposit(user_id)`, `category(user_id)`, `category(category_id)`, `item(expense_id)`, and `expense(category_id)`. I added these since these were the sequential scans with filters listed on the `EXPLAIN` query.

@router.post("/users/", tags=["users"])

```
-- Before indexes
-- Index done: N/A
-- No index added for this endpoint
```

Indexes optimize for queries that might do read operations. For this endpoint, we're just adding a row to the users table. No index added for this endpoint.

@router.post("/users/login", tags=["users"])

```
-- Before indexes
"Index Scan using idx_user_name on ""user""  (cost=0.42..8.45 rows=1 width=8) (actual time=0.049..0.050 rows=1 loops=1)"
"  Index Cond: (name = 'samuel96@example.org'::text)"
"  Filter: (hashed_pwd = 'password'::text)"
"Planning Time: 0.075 ms"
"Execution Time: 0.064 ms"

-- Index done
CREATE INDEX idx_user_name_and_hashed_pwd ON "user" (name, hashed_pwd);

-- After indexes
"Index Scan using idx_user_name_and_hashed_pwd on ""user""  (cost=0.42..8.45 rows=1 width=8) (actual time=0.048..0.049 rows=1 loops=1)"
"  Index Cond: ((name = 'samuel96@example.org'::text) AND (hashed_pwd = 'password'::text))"
"Planning Time: 0.736 ms"
"Execution Time: 0.060 ms"
```

The query is currently finding the user based on their name and password. In that case, we should index both `name` and `hashed_pwd` to find users based on those two things as an index.

Deposit

@router.get("/user/{user_id}/deposits/", tags=["deposits"])

```
-- Before indexes
"Limit  (cost=1000.00..17728.10 rows=1 width=21) (actual time=1.293..41.688 rows=1 loops=1)"
"  ->  Gather  (cost=1000.00..17728.10 rows=1 width=21) (actual time=1.292..41.686 rows=1 loops=1)"
"        Workers Planned: 2"
"        Workers Launched: 2"
"        ->  Parallel Seq Scan on deposit  (cost=0.00..16728.00 rows=1 width=21) (actual time=11.639..24.074 rows=0 loops=3)"
"              Filter: ((user_id = 10000) AND (date(""timestamp"") >= '2023-06-06'::date) AND (date(""timestamp"") <= '2023-06-13'::date))"
"              Rows Removed by Filter: 333333"
"Planning Time: 0.599 ms"
"Execution Time: 41.703 ms"

-- Index done
CREATE INDEX idx_deposit_user_id_timestamp ON deposit(user_id, timestamp);

-- After indexes
"Limit  (cost=0.42..8.45 rows=1 width=21) (actual time=0.037..0.038 rows=1 loops=1)"
"  ->  Index Scan using idx_deposit_user_id_timestamp on deposit  (cost=0.42..8.45 rows=1 width=21) (actual time=0.036..0.037 rows=1 loops=
"        Index Cond: (user_id = 10000)"
"        Filter: ((date(""timestamp"") >= '2023-06-06'::date) AND (date(""timestamp"") <= '2023-06-13'::date))"
"Planning Time: 0.801 ms"
"Execution Time: 0.048 ms"
```

The query is currently finding a deposit based on a specific `user_id` and a `timestamp`. In that case, adding an index for both attributes could help find a `deposit_id` quicker.

@router.post("/user/{user_id}/deposits/", tags=["deposits"])

```
-- Before indexes
-- Index done: N/A
-- No index added for this endpoint
```

Indexes optimize for queries that might do read operations. For this endpoint, we're just adding a row to the deposits table. No index added for this endpoint.

Category

@router.post("/users/{user_id}/categories/", tags=["category"])

```
-- Before indexes
-- Index done: N/A
-- No index added for this endpoint
```

Indexes optimize for queries that might do read operations. For this endpoint, we're just adding a row to the category table. No index added for this endpoint.

@router.get("/users/{user_id}/categories", tags=["category"])

```
-- Before indexes
"Limit  (cost=1000.42..16603.89 rows=1 width=45) (actual time=0.968..42.776 rows=1 loops=1)"
"  ->  Nested Loop  (cost=1000.42..16603.89 rows=1 width=45) (actual time=0.968..42.775 rows=1 loops=1)"
"        ->  Gather  (cost=1000.00..16595.43 rows=1 width=53) (actual time=0.954..42.759 rows=1 loops=1)"
"              Workers Planned: 2"
"              Workers Launched: 2"
"              ->  Parallel Seq Scan on category c  (cost=0.00..15595.33 rows=1 width=53) (actual time=11.888..24.863 rows=0 loops=3)"
"                    Filter: (user_id = 10050)"
"                    Rows Removed by Filter: 333333"
"        ->  Index Only Scan using user_pkey on ""user"" u  (cost=0.42..8.44 rows=1 width=8) (actual time=0.013..0.014 rows=1 loops=1)"
"              Index Cond: (user_id = 10050)"
"              Heap Fetches: 0"
"Planning Time: 0.092 ms"
```

```
"Execution Time: 42.798 ms"

-- Index done
CREATE INDEX idx_category_user_id ON category(user_id);

-- After indexes
"Limit  (cost=0.85..16.90 rows=1 width=45) (actual time=0.058..0.059 rows=1 loops=1)"
"  ->  Nested Loop  (cost=0.85..16.90 rows=1 width=45) (actual time=0.057..0.058 rows=1 loops=1)"
"        ->  Index Scan using idx_category_user_id on category c  (cost=0.42..8.44 rows=1 width=53) (actual time=0.047..0.047 rows=1 loops=
"              Index Cond: (user_id = 10050)"
"        ->  Index Only Scan using user_pkey on ""user"" u  (cost=0.42..8.44 rows=1 width=8) (actual time=0.009..0.010 rows=1 loops=1)"
"              Index Cond: (user_id = 10050)"
"              Heap Fetches: 0"
"Planning Time: 0.762 ms"
"Execution Time: 0.071 ms"
```

For this query, we're finding the categories associated with a `user_id`. Because of that, we indexed the by `category(user_id)` since the sequential search happens on category with a filter based on the `user_id`

@router.get("/users/{user_id}/categories/{category_id}", tags=["category"])

```
-- Before indexes
"Nested Loop  (cost=0.85..16.90 rows=1 width=45) (actual time=0.037..0.038 rows=1 loops=1)"
"  ->  Index Scan using idx_category_user_id on category c  (cost=0.42..8.45 rows=1 width=53) (actual time=0.023..0.024 rows=1 loops=1)"
"        Index Cond: (user_id = 48)"
"        Filter: (category_id = 48)"
"  ->  Index Only Scan using user_pkey on ""user"" u  (cost=0.42..8.44 rows=1 width=8) (actual time=0.013..0.013 rows=1 loops=1)"
"        Index Cond: (user_id = 48)"
"        Heap Fetches: 0"
"Planning Time: 0.147 ms"
"Execution Time: 0.065 ms"

-- Index done
CREATE INDEX idx_category_user_category_id ON category(user_id, category_id);

-- After indexes
"Nested Loop  (cost=0.85..16.90 rows=1 width=45) (actual time=0.044..0.045 rows=1 loops=1)"
"  ->  Index Scan using idx_category_user_category_id on category c  (cost=0.42..8.45 rows=1 width=53) (actual time=0.033..0.033 rows=1 loo
"        Index Cond: ((user_id = 48) AND (category_id = 48))"
"  ->  Index Only Scan using user_pkey on ""user"" u  (cost=0.42..8.44 rows=1 width=8) (actual time=0.009..0.010 rows=1 loops=1)"
"        Index Cond: (user_id = 48)"
"        Heap Fetches: 0"
"Planning Time: 0.835 ms"
"Execution Time: 0.057 ms"
```

For this query, we're finding the categories associated with a `user_id` and a `category_id`. Because of that, we indexed the by `category(user_id, category_id)`.

Budget

@router.post("/users/{user_id}/categories/{category_id}/budget/", tags=["budgets"])

```
-- Before indexes
-- Index done: N/A
-- No index added for this endpoint
```

Indexes optimize for queries that might do read operations. For this endpoint, we're just adding a row to the budget table. No index added for this endpoint.

@router.get("/users/{user_id}/categories/{category_id}/budget/{budget_id}/", tags=["budgets"])

```
-- Before indexes
"Nested Loop  (cost=1.27..25.35 rows=1 width=37) (actual time=0.037..0.039 rows=1 loops=1)"
"  ->  Nested Loop  (cost=0.85..16.90 rows=1 width=45) (actual time=0.025..0.026 rows=1 loops=1)"
"        ->  Index Scan using budget_pkey on budget  (cost=0.42..8.45 rows=1 width=37) (actual time=0.016..0.016 rows=1 loops=1)"
"              Index Cond: (budget_id = 150000)"
"              Filter: (category_id = 150000)"
```

```
"        ->  Index Scan using category_pkey on category  (cost=0.42..8.45 rows=1 width=16) (actual time=0.008..0.009 rows=1 loops=1)"
"              Index Cond: (category_id = 150000)"
"              Filter: (user_id = 150000)"
"  ->  Index Only Scan using user_pkey on ""user""  (cost=0.42..8.44 rows=1 width=8) (actual time=0.012..0.012 rows=1 loops=1)"
"        Index Cond: (user_id = 150000)"
"        Heap Fetches: 0"
"Planning Time: 0.168 ms"
"Execution Time: 0.061 ms"

-- Index done
CREATE INDEX idx_budget_budget_category_id ON budget(budget_id, category_id);
CREATE INDEX idx_category_category_user_id ON category(category_id, user_id);

-- After indexes
"Nested Loop  (cost=1.27..21.35 rows=1 width=37) (actual time=0.268..0.270 rows=1 loops=1)"
"  ->  Nested Loop  (cost=0.85..12.90 rows=1 width=45) (actual time=0.253..0.254 rows=1 loops=1)"
"        ->  Index Scan using idx_budget_budget_category_id on budget  (cost=0.42..8.45 rows=1 width=37) (actual time=0.035..0.036 rows=1 l
"              Index Cond: ((budget_id = 150000) AND (category_id = 150000))"
"        ->  Index Only Scan using idx_category_category_user_id on category  (cost=0.42..4.44 rows=1 width=16) (actual time=0.216..0.216 r
"              Index Cond: ((category_id = 150000) AND (user_id = 150000))"
"              Heap Fetches: 0"
"  ->  Index Only Scan using user_pkey on ""user""  (cost=0.42..8.44 rows=1 width=8) (actual time=0.015..0.015 rows=1 loops=1)"
"        Index Cond: (user_id = 150000)"
"        Heap Fetches: 0"
"Planning Time: 1.347 ms"
"Execution Time: 0.291 ms"
```

For this query, we're finding the budget associated with a `budget_id`. The query filters based on the `category_id` while it uses the `budget_id` when searching within the budget table. So for that, we added `idx_budget_budget_category_id`. Since category also searches based on it's primary key but filters based on the `user_id`, we added `idx_category_category_user_id`

@router.get("/users/{user_id}/categories/{category_id}/budget/", tags=["budgets"])

```
-- Before indexes
"Limit  (cost=16595.35..30189.86 rows=1 width=66) (actual time=80.442..83.546 rows=1 loops=1)"
"  ->  Gather  (cost=16595.35..30189.86 rows=1 width=66) (actual time=80.441..83.544 rows=1 loops=1)"
"        Workers Planned: 2"
"        Workers Launched: 2"
"        ->  Parallel Hash Join  (cost=15595.35..29189.76 rows=1 width=66) (actual time=52.369..65.853 rows=0 loops=3)"
"              Hash Cond: (b.category_id = c.category_id)"
"              ->  Parallel Seq Scan on budget b  (cost=0.00..12500.67 rows=416667 width=37) (actual time=0.376..22.070 rows=333333 loops=3
"              ->  Parallel Hash  (cost=15595.33..15595.33 rows=1 width=45) (actual time=23.988..23.988 rows=0 loops=3)"
"                    Buckets: 1024  Batches: 1  Memory Usage: 40kB"
"                    ->  Parallel Seq Scan on category c  (cost=0.00..15595.33 rows=1 width=45) (actual time=11.755..23.968 rows=0 loops=3)
"                          Filter: (user_id = 20200)"
"                          Rows Removed by Filter: 333333"
"Planning Time: 0.158 ms"
"Execution Time: 83.567 ms"

-- Index done
CREATE INDEX idx_category_user_id ON category(user_id);

-- After indexes
"Limit  (cost=1008.46..14602.97 rows=1 width=66) (actual time=2.472..57.228 rows=1 loops=1)"
"  ->  Gather  (cost=1008.46..14602.97 rows=1 width=66) (actual time=2.471..57.226 rows=1 loops=1)"
"        Workers Planned: 2"
"        Workers Launched: 2"
"        ->  Hash Join  (cost=8.46..13602.87 rows=1 width=66) (actual time=22.050..39.266 rows=0 loops=3)"
"              Hash Cond: (b.category_id = c.category_id)"
"              ->  Parallel Seq Scan on budget b  (cost=0.00..12500.67 rows=416667 width=37) (actual time=0.169..22.963 rows=333333 loops=3
"              ->  Hash  (cost=8.44..8.44 rows=1 width=45) (actual time=0.036..0.037 rows=1 loops=3)"
"                    Buckets: 1024  Batches: 1  Memory Usage: 9kB"
"                    ->  Index Scan using idx_category_user_id on category c  (cost=0.42..8.44 rows=1 width=45) (actual time=0.032..0.033 r
"                          Index Cond: (user_id = 20200)"
"Planning Time: 0.607 ms"
"Execution Time: 57.248 ms"
```

For this query, we're finding all the budgets associated with a `user_id` and `category_id`. For this, we indexed the `category(user_id)` since that's where we do a sequential search on based on the `user_id` as a filter.

Expense

@router.get("/users/{user_id}/expenses/{expense_id}", *tags*=["expenses"])

```
-- Before indexes
"GroupAggregate  (cost=17246.36..17246.39 rows=1 width=129) (actual time=38.447..42.066 rows=1 loops=1)"
"  Group Key: expense.expense_id, category.category_id"
"  ->  Sort  (cost=17246.36..17246.37 rows=1 width=102) (actual time=38.440..42.057 rows=1 loops=1)"
"        Sort Key: category.category_id"
"        Sort Method: quicksort  Memory: 25kB"
"        ->  Nested Loop Left Join  (cost=1000.85..17246.35 rows=1 width=102) (actual time=22.251..42.049 rows=1 loops=1)"
"              Join Filter: (expense.expense_id = item.expense_id)"
"              ->  Nested Loop  (cost=0.85..24.91 rows=1 width=97) (actual time=0.016..0.020 rows=1 loops=1)"
"                    ->  Index Scan using expense_pkey on expense  (cost=0.42..8.44 rows=1 width=60) (actual time=0.010..0.013 rows=1 loops
"                          Index Cond: (expense_id = 333222)"
"                    ->  Index Scan using category_pkey on category  (cost=0.42..8.45 rows=1 width=45) (actual time=0.005..0.005 rows=1 loo
"                          Index Cond: (category_id = expense.category_id)"
"                          Filter: (user_id = 333222)"
"              ->  Gather  (cost=1000.00..17221.43 rows=1 width=13) (actual time=22.233..42.025 rows=1 loops=1)"
"                    Workers Planned: 2"
"                    Workers Launched: 2"
"                    ->  Parallel Seq Scan on item  (cost=0.00..16221.33 rows=1 width=13) (actual time=16.652..22.019 rows=0 loops=3)"
"                          Filter: (expense_id = 333222)"
"                          Rows Removed by Filter: 333333"
"Planning Time: 0.211 ms"
"Execution Time: 42.107 ms"

-- Index done:
CREATE INDEX idx_item_expense_id ON item(expense_id);

-- After indexes
"GroupAggregate  (cost=33.37..33.40 rows=1 width=129) (actual time=0.077..0.078 rows=1 loops=1)"
"  Group Key: expense.expense_id, category.category_id"
"  ->  Sort  (cost=33.37..33.38 rows=1 width=102) (actual time=0.072..0.073 rows=1 loops=1)"
"        Sort Key: category.category_id"
"        Sort Method: quicksort  Memory: 25kB"
"        ->  Nested Loop Left Join  (cost=1.27..33.36 rows=1 width=102) (actual time=0.063..0.064 rows=1 loops=1)"
"              Join Filter: (expense.expense_id = item.expense_id)"
"              ->  Nested Loop  (cost=0.85..24.91 rows=1 width=97) (actual time=0.018..0.018 rows=1 loops=1)"
"                    ->  Index Scan using expense_pkey on expense  (cost=0.42..8.44 rows=1 width=60) (actual time=0.008..0.009 rows=1 loops
"                          Index Cond: (expense_id = 333222)"
"                    ->  Index Scan using category_pkey on category  (cost=0.42..8.45 rows=1 width=45) (actual time=0.008..0.008 rows=1 loo
"                          Index Cond: (category_id = expense.category_id)"
"                          Filter: (user_id = 333222)"
"              ->  Index Scan using idx_item_expense_id on item  (cost=0.42..8.44 rows=1 width=13) (actual time=0.044..0.044 rows=1 loops=1
"                    Index Cond: (expense_id = 333222)"
"Planning Time: 1.107 ms"
"Execution Time: 0.101 ms"
```

For this query, we're finding the information related to a specific `expense_id` for a given user. Since the only sequential scan we're doing is based on the `expense_id`, we created an index for the item table's `expense_id`.

@router.get("/users/{user_id}/expenses", tags=["expenses"])

```
-- Before indexes
"Limit  (cost=57190.96..57190.99 rows=1 width=129) (actual time=539.676..551.634 rows=1 loops=1)"
"  ->  GroupAggregate  (cost=57190.96..57190.99 rows=1 width=129) (actual time=539.675..551.632 rows=1 loops=1)"
"        Group Key: expense.expense_id, category.category_id"
"        ->  Sort  (cost=57190.96..57190.97 rows=1 width=102) (actual time=539.664..551.622 rows=1 loops=1)"
"              Sort Key: expense.expense_id, category.category_id"
"              Sort Method: quicksort  Memory: 25kB"
"              ->  Gather  (cost=23423.43..57190.95 rows=1 width=102) (actual time=505.568..551.610 rows=1 loops=1)"
"                    Workers Planned: 2"
"                    Workers Launched: 2"
"                    ->  Parallel Hash Left Join  (cost=22423.43..56190.85 rows=1 width=102) (actual time=512.593..523.544 rows=0 loops=3)"
"                          Hash Cond: (expense.expense_id = item.expense_id)"
"                          ->  Nested Loop  (cost=0.42..31730.83 rows=1 width=97) (actual time=358.376..406.206 rows=0 loops=3)"
"                                ->  Parallel Seq Scan on expense  (cost=0.00..19720.33 rows=2083 width=60) (actual time=0.239..53.166 rows
"                                      Filter: ((date(""timestamp"") >= '2023-06-06'::date) AND (date(""timestamp"") <= '2023-06-13'::date)
"                                ->  Index Scan using category_pkey on category  (cost=0.42..5.76 rows=1 width=45) (actual time=0.001..0.00
"                                      Index Cond: (category_id = expense.category_id)"
"                                      Filter: (user_id = 643576)"
"                                      Rows Removed by Filter: 1"
"                          ->  Parallel Hash  (cost=15179.67..15179.67 rows=416667 width=13) (actual time=83.282..83.283 rows=333333 loops=
"                                Buckets: 262144  Batches: 8  Memory Usage: 7968kB"
"                                ->  Parallel Seq Scan on item  (cost=0.00..15179.67 rows=416667 width=13) (actual time=0.168..35.318 rows=
"Planning Time: 0.304 ms"
```

```
"Execution Time: 551.690 ms"

-- Index done:
CREATE INDEX idx_expense_timestamp ON expense((date(timestamp)));
CREATE INDEX idx_item_expense_id ON item(expense_id);
CREATE INDEX idx_category_user_id ON category(user_id);

-- After indexes
"Limit  (cost=9267.82..9267.85 rows=1 width=129) (actual time=147.672..147.674 rows=1 loops=1)"
"  ->  GroupAggregate  (cost=9267.82..9267.85 rows=1 width=129) (actual time=147.671..147.672 rows=1 loops=1)"
"        Group Key: expense.expense_id, category.category_id"
"        ->  Sort  (cost=9267.82..9267.83 rows=1 width=102) (actual time=147.662..147.663 rows=1 loops=1)"
"              Sort Key: expense.expense_id, category.category_id"
"              Sort Method: quicksort  Memory: 25kB"
"              ->  Nested Loop Left Join  (cost=72.52..9267.81 rows=1 width=102) (actual time=100.329..147.656 rows=1 loops=1)"
"                    ->  Nested Loop  (cost=72.10..9262.01 rows=1 width=97) (actual time=100.272..147.598 rows=1 loops=1)"
"                          Join Filter: (expense.category_id = category.category_id)"
"                          Rows Removed by Join Filter: 999999"
"                          ->  Index Scan using idx_category_user_id on category  (cost=0.42..8.44 rows=1 width=45) (actual time=0.026..0.0
"                                Index Cond: (user_id = 643576)"
"                          ->  Bitmap Heap Scan on expense  (cost=71.67..9191.06 rows=5000 width=60) (actual time=17.368..108.770 rows=1000
"                                Recheck Cond: ((date(""timestamp"") >= '2023-06-06'::date) AND (date(""timestamp"") <= '2023-06-13'::date)
"                                Heap Blocks: exact=11387"
"                                ->  Bitmap Index Scan on idx_expense_timestamp  (cost=0.00..70.42 rows=5000 width=0) (actual time=16.333..
"                                      Index Cond: ((date(""timestamp"") >= '2023-06-06'::date) AND (date(""timestamp"") <= '2023-06-13'::d
"                    ->  Index Scan using idx_item_expense_id on item  (cost=0.42..5.80 rows=1 width=13) (actual time=0.052..0.053 rows=1 l
"                          Index Cond: (expense_id = expense.expense_id)"
"Planning Time: 2.705 ms"
"Execution Time: 147.768 ms"
```

For this query, we're listing all the expenses for a user within a certain time frame. Since we use the timestamp for searching, we added an index for the expense table's `timestamp` column. For all sequential searches, we added an index to optimize searches. That resulted in us adding indexes for item's `expense_id` and category's `user_id`.

@router.post("/users/{user_id}/expenses/", *tags*=["expenses"])

```
-- Before indexes
-- Index done: N/A
-- No index added for this endpoint
```

Indexes optimize for queries that might do read operations. For this endpoint, we're just adding a row to the expenses table. No index added for this endpoint.

Items

@router.post("/users/{user_id}/expenses/{expense_id}/items", tags=["items"])

```
-- Before indexes
-- Index done: N/A
-- No index added for this endpoint
```

Indexes optimize for queries that might do read operations. For this endpoint, we're just adding a row to the items table. No index added for this endpoint.

@router.get("/users/{user_id}/expenses/{expense_id}/items/{item_id}", tags=["items"])

```
-- Before indexes
"Nested Loop  (cost=1.27..33.36 rows=1 width=50) (actual time=0.069..0.070 rows=1 loops=1)"
"  ->  Nested Loop  (cost=0.85..16.90 rows=1 width=58) (actual time=0.049..0.050 rows=1 loops=1)"
"        ->  Index Scan using item_pkey on item  (cost=0.42..8.45 rows=1 width=58) (actual time=0.042..0.042 rows=1 loops=1)"
"              Index Cond: (item_id = 306648)"
"              Filter: (expense_id = 306648)"
"        ->  Index Scan using expense_pkey on expense  (cost=0.42..8.44 rows=1 width=16) (actual time=0.006..0.006 rows=1 loops=1)"
"              Index Cond: (expense_id = 306648)"
"  ->  Index Scan using category_pkey on category  (cost=0.42..8.45 rows=1 width=8) (actual time=0.019..0.019 rows=1 loops=1)"
"        Index Cond: (category_id = expense.category_id)"
```

```
"        Filter: (user_id = 306648)"
"Planning Time: 1.349 ms"
"Execution Time: 0.088 ms"

-- Index done:
CREATE INDEX idx_category_user_id ON category(user_id);

-- After indexes
"Nested Loop  (cost=1.27..25.35 rows=1 width=50) (actual time=0.056..0.057 rows=1 loops=1)"
"  Join Filter: (expense.category_id = category.category_id)"
"  ->  Nested Loop  (cost=0.85..16.90 rows=1 width=58) (actual time=0.023..0.024 rows=1 loops=1)"
"        ->  Index Scan using item_pkey on item  (cost=0.42..8.45 rows=1 width=58) (actual time=0.012..0.012 rows=1 loops=1)"
"              Index Cond: (item_id = 306648)"
"              Filter: (expense_id = 306648)"
"        ->  Index Scan using expense_pkey on expense  (cost=0.42..8.44 rows=1 width=16) (actual time=0.010..0.011 rows=1 loops=1)"
"              Index Cond: (expense_id = 306648)"
"  ->  Index Scan using idx_category_user_id on category  (cost=0.42..8.44 rows=1 width=8) (actual time=0.032..0.032 rows=1 loops=1)"
"        Index Cond: (user_id = 306648)"
"Planning Time: 1.131 ms"
"Execution Time: 0.076 ms"
```

For this query, it was more simple than most other queries. It was just searching for an item based on an `item_id`. We join it to the expense table, but we use `expense_id` which is the primary key of the expense table. We can add an index for category's `user_id` so that searches based on `user_id` will be marginally quicker.

@router.get("/users/{user_id}/expenses/{expense_id}/items", *tags*=["items"])

```
-- Before indexes
"Limit  (cost=1000.85..17246.35 rows=1 width=50) (actual time=22.228..44.673 rows=1 loops=1)"
"  ->  Nested Loop  (cost=1000.85..17246.35 rows=1 width=50) (actual time=22.227..44.671 rows=1 loops=1)"
"        ->  Nested Loop  (cost=1000.42..17229.89 rows=1 width=58) (actual time=22.217..44.660 rows=1 loops=1)"
"              ->  Gather  (cost=1000.00..17221.43 rows=1 width=58) (actual time=22.201..44.643 rows=1 loops=1)"
"                    Workers Planned: 2"
"                    Workers Launched: 2"
"                    ->  Parallel Seq Scan on item  (cost=0.00..16221.33 rows=1 width=58) (actual time=20.473..26.904 rows=0 loops=3)"
"                          Filter: (expense_id = 306648)"
"                          Rows Removed by Filter: 333333"
"              ->  Index Scan using expense_pkey on expense  (cost=0.42..8.44 rows=1 width=16) (actual time=0.014..0.014 rows=1 loops=1)"
"                    Index Cond: (expense_id = 306648)"
"        ->  Index Scan using category_pkey on category  (cost=0.42..8.45 rows=1 width=8) (actual time=0.008..0.008 rows=1 loops=1)"
"              Index Cond: (category_id = expense.category_id)"
"              Filter: (user_id = 306648)"
"Planning Time: 0.857 ms"
"Execution Time: 44.700 ms"

-- Index done:
CREATE INDEX idx_item_expense_id ON item(expense_id);

-- After indexes
"Limit  (cost=1.27..33.36 rows=1 width=50) (actual time=0.058..0.060 rows=1 loops=1)"
"  ->  Nested Loop  (cost=1.27..33.36 rows=1 width=50) (actual time=0.058..0.059 rows=1 loops=1)"
"        ->  Nested Loop  (cost=0.85..16.90 rows=1 width=58) (actual time=0.048..0.049 rows=1 loops=1)"
"              ->  Index Scan using idx_item_expense_id on item  (cost=0.42..8.44 rows=1 width=58) (actual time=0.035..0.036 rows=1 loops=1
"                    Index Cond: (expense_id = 306648)"
"              ->  Index Scan using expense_pkey on expense  (cost=0.42..8.44 rows=1 width=16) (actual time=0.011..0.011 rows=1 loops=1)"
"                    Index Cond: (expense_id = 306648)"
"        ->  Index Scan using category_pkey on category  (cost=0.42..8.45 rows=1 width=8) (actual time=0.009..0.009 rows=1 loops=1)"
"              Index Cond: (category_id = expense.category_id)"
"              Filter: (user_id = 306648)"
"Planning Time: 0.936 ms"
"Execution Time: 0.081 ms"
```

For this query, we're listing all the expenses that a category has. We see that the query does a sequential scan on item's `expense_id`. We can add an index for this to optimize against sequential searches.