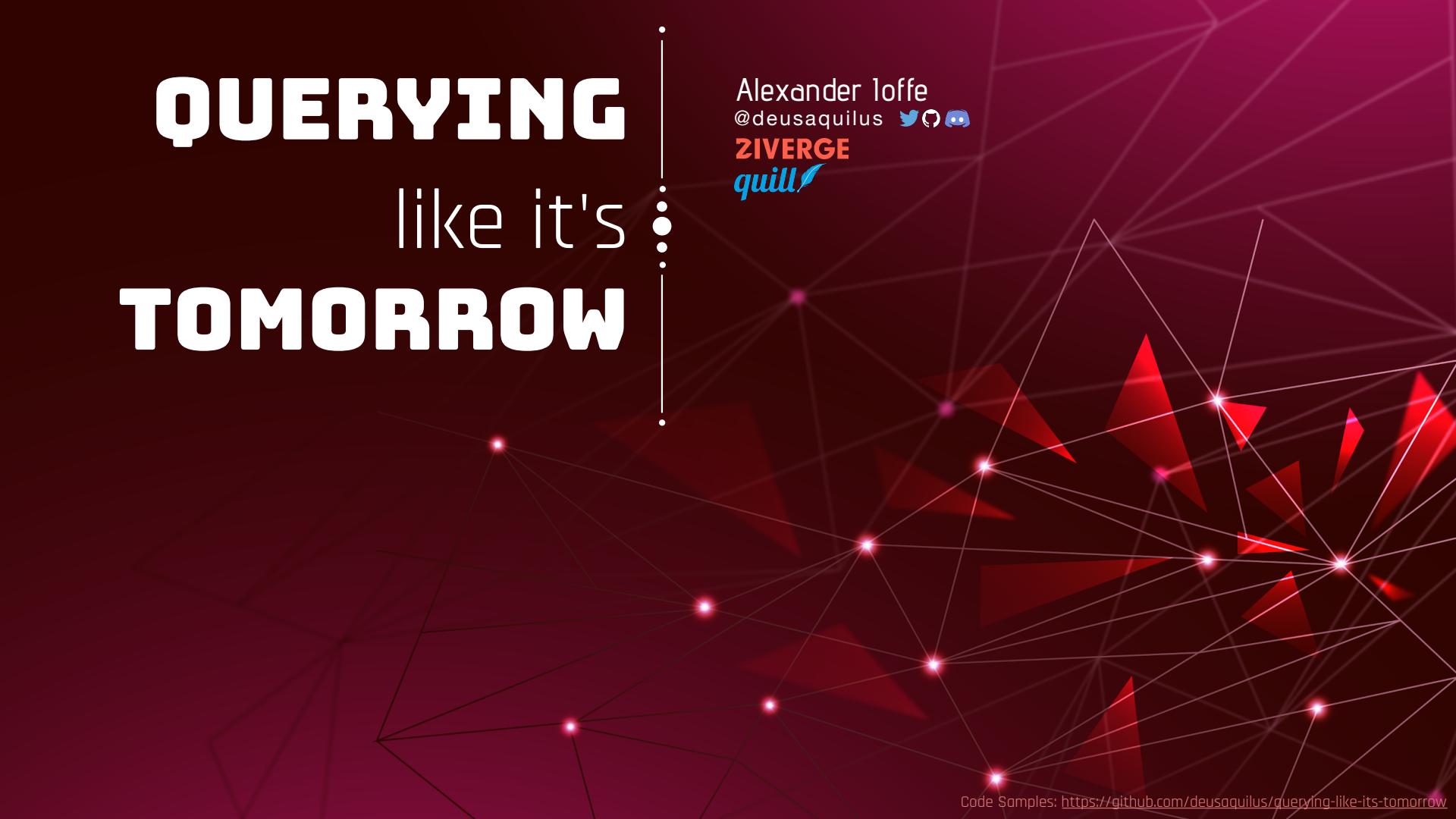
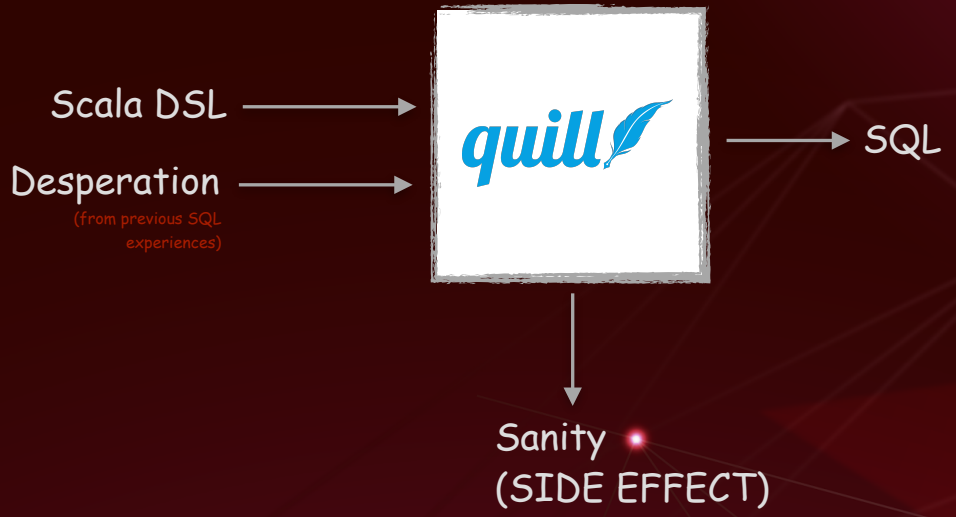
QUERYING like it's TOMORROW

Alexander Ioffe
@deusaquilus
ZIVERGE
quill



Code Samples: https://github.com/deusaquilus/querying-like-its-tomorrow



Alexander Ioffe
@deusaquilus
ZIVERGE
quill

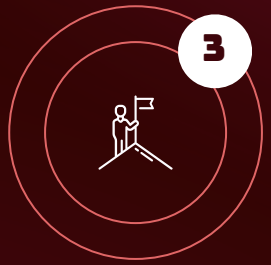
IN TODAYS INSTALLMENT...

1 SQL IN THE TRENCHES

SQL is hard
Quill makes it sane

2 BETTER, MUCH BETTER!

New Capabilities
of Quill in Scala 3



3 ZIO IS BEAUTIFUL

Why ZIO is the best way
to handle Quill outputs

4 THE ECOSYSTEM!

Caliban takes things
to a new level!

in case of http://api.human_customers

SERVE THIS TO THE USER

```
SELECT h.firstName || ' ' || h.lastName AS name, h.age, h.membership FROM Humans h
```

Code Sample: query_series.sql

in case of http://api.human_customers

SERVE THIS TO THE USER?

```
SELECT h.firstName || ' ' || h.lastName AS name, h.age, h.membership FROM Humans h
WHERE h.memberOf = 'h'
```

Code Sample: query_series.sql

in case of http://api.human_customers

SERVE THIS TO THE USER??

```
SELECT h.firstName || ' ' || h.lastName AS name, h.age, h.membership FROM Humans h
WHERE h.memberOf = 'h'
UNION
SELECT h.heroName AS name, h.age, 'PLAT' FROM SuperHumans h
WHERE h.side = 'g'
```

Code Sample: query_series.sql

in case of http://api.human_customers

SERVE THIS TO THE USER???

```
SELECT h.firstName || ' ' || h.lastName AS name, h.age, h.membership FROM Humans h
WHERE h.segment = 'h' AND h.age > 44
UNION
SELECT h.heroName AS name, h.age, 'PLAT' FROM SuperHumans h
WHERE h.side = 'g' AND h.age > 123
```

Code Sample: query_series.sql

SERVE THIS TO THE... WHO???

in case of http://api.human__customers

```
SELECT h.firstName || ' ' || h.lastName AS name, h.age, h.membership FROM Humans h
WHERE h.segment = 'h' AND h.age > 44
UNION
SELECT h.heroName AS name, h.age, 'PLAT' FROM SuperHumans h
WHERE h.side = 'g' AND h.age > 123
```

in case of http://api.robot__customers

```
SELECT r.model AS name, today() - r.assemblyYear, 'AUTO'
FROM Robots r
WHERE r.assemblyYear > 1990
UNION
SELECT r.series || "-" || r.model AS name, today() - r.assemblyYear, 'AUTO'
FROM KillerRobots r
WHERE r.series  = 'T' AND r.assemblyYear > 1992
```

Code Sample: query_series.sql

IF WE HAD POLYMORPHISM....

in case of http://api.human__customers

```
(SELECT Humans h WHERE h.segment = 'h').toCustomer(h.firstName || ' ' || h.lastName, h.membership, 1982)
UNION
(SELECT SuperHumans h WHERE r.side  = 'g').toCustomer(h.heroName, 'PLAT', 1982)
```

in case of http://api.robot__customers

```
(SELECT Robots r).toCustomer(r.label, 1990)
UNION
(SELECT KillerRobots r WHERE r.series  = 'T').toCustomer(r.series || "-" || r.model, 1992)
```

SO WE WERE SAYING...

in case of http://api.human__customers

```
SELECT h.firstName || ' ' || h.lastName AS name, h.age, h.membership FROM Humans h
WHERE h.segment = 'h' AND h.age > 44
UNION
SELECT h.heroName AS name, h.age, 'PLAT' FROM SuperHumans h
WHERE h.side = 'g' AND h.age > 123
```

in case of http://api.robot__customers

```
SELECT r.model AS name, today() - r.assemblyYear, 'AUTO'
FROM Robots r
WHERE r.assemblyYear > 1990
UNION
SELECT r.series || "-" || r.model AS name, today() - r.assemblyYear, 'AUTO' FROM KillerRobots r
WHERE r.series  = 'T' AND r.assemblyYear > 1992
```

Code Sample: query_series.sql

SO WE WERE SAYING... MORE PAIN!

in case of http://api.human__customers

```
SELECT customer.name, customer.age, customer.membership, customer.id, h.id AS hid FROM (
  SELECT h.firstName || ' ' || h.lastName AS name, h.age, h.membership, h.id FROM Humans h
  WHERE h.segment = 'h' AND h.age > 1982
  UNION
  SELECT h.heroName AS name, h.age, 'PLAT', h.id FROM SuperHumans h
  WHERE h.side = 'g' AND h.age > 1856
) AS customer
JOIN Houses h ON h.OWNER = customer.ID
```

in case of http://api.robot__customers

```
SELECT customer.name, customer.age, customer.membership, customer.id, h.id AS hid FROM (
  SELECT r.model AS name, today() - r.assemblyYear, 'AUTO' FROM Robots r
  WHERE r.assemblyYear > 1990
  UNION
  SELECT r.series || "-" || r.model AS name, today() - r.assemblyYear, 'AUTO' FROM KillerRobots r
  WHERE r.series  = 'T' AND r.assemblyYear > 1992
) AS customer
JOIN Houses h ON h.OWNER = customer.ID
```

Code Sample: query_series.sql

SO WE WERE SAYING... MORE PAIN!... AND MORE...

in case of http://api.human__customers

```
SELECT customer.name, customer.age, customer.membership, customer.id, h.id AS hid FROM (
  SELECT h.firstName || ' ' || h.lastName AS name, h.age, h.membership, h.id FROM Humans h
  WHERE h.segment = 'h' AND h.age > 1982
  UNION
  SELECT h.heroName AS name, h.age, 'PLAT', h.id FROM SuperHumans h
  WHERE h.side = 'g' AND h.age > 1856
) AS customer
JOIN Houses h ON h.OWNER = customer.ID
JOIN PricingYears p ON customer.age > p.startYear && customer.age < p.endYear
```

in case of http://api.robot__customers

```
SELECT customer.name, customer.age, customer.membership, customer.id, h.id AS hid FROM (
  SELECT r.model AS name, today() - r.assemblyYear, 'AUTO' FROM Robots r
  WHERE r.assemblyYear > 1990
  UNION
  SELECT r.series || "-" || r.model AS name, today() - r.assemblyYear, 'AUTO' FROM KillerRobots r
  WHERE r.series  = 'T' AND r.assemblyYear > 1992
) AS customer
JOIN Pods h ON h.OWNER = customer.ID
JOIN PricingYears p ON customer.age > p.startYear && customer.age < p.endYear
```

Code Sample: query_series.sql

SO WE WERE SAYING... MORE PAIN!... AND MORE... AND MORE...

in case of http://api.human__customers

```
SELECT customer.name, customer.age,
  CASE WHEN p.pricing == 'sane' THEN p. customer.membership ELSE p.insaneMembership END as membership,
  customer.id, h.id AS hid FROM
(
  SELECT h.firstName || ' ' || h.lastName AS name, h.age, h.membership, h.id FROM Humans h
  WHERE h.segment = 'h' AND h.age > 1982
  UNION
  SELECT h.heroName AS name, h.age, 'PLAT', h.id FROM SuperHumans h
  WHERE h.side = 'g' AND h.age > 1856
) AS customer
JOIN Houses h ON h.OWNER = customer.ID
JOIN PricingYears p ON customer.age > p.startYear && customer.age < p.endYear
```

in case of http://api.robot__customers

```
SELECT customer.name, customer.age,
  CASE WHEN p.voltage = 120 THEN 'USA' ELSE 'EU' END as membership,
  customer.id, h.id AS hid FROM
(
  SELECT r.model AS name, today() - r.assemblyYear, 'AUTO' FROM Robots r
  WHERE r.assemblyYear > 1990
  UNION
  SELECT r.series || "-" || r.model AS name, today() - r.assemblyYear, 'AUTO' FROM KillerRobots r
  WHERE r.series  = 'T' AND r.assemblyYear > 1992
) AS customer
JOIN Houses h ON h.OWNER = customer.ID AND h.hasChargingPort=true
JOIN PricingYears p ON customer.age > p.startYear && customer.age < p.endYear
```

Code Sample: query_series.sql

SO WE WERE SAYING... MORE PAIN!... AND MORE... AND MORE... AND

in case of http://api.human__customers

```
SELECT customer.name, customer.age,
  CASE WHEN p.pricing == 'sane' THEN customer.membership ELSE p.insaneMembership END as membership,
  customer.id, h.id AS hid  FROM
(
  SELECT h.firstName || ' ' || h.lastName AS name, h.age, h.membership, h.id FROM Humans h
  WHERE h.segment = 'h' AND h.age > 1982
  UNION
  SELECT h.heroName AS name, h.age, 'PLAT', h.id FROM SuperHumans h
  WHERE h.side = 'g' AND h.age > 1856
) AS customer
JOIN Houses h ON h.OWNER = customer.ID
JOIN PricingYears p ON customer.age > p.startYear && customer.age < p.endYear
```

in case of http://api.robot__customers

```
SELECT customer.name, customer.age,
  CASE WHEN p.voltage = 120 THEN 'USA' ELSE 'EU' END as membership,
  customer.id, h.id AS hid FROM
(
  SELECT r.model AS name, today() - r.assemblyYear, 'AUTO' FROM Robots r
  WHERE r.assemblyYear > 1990
  UNION
  SELECT r.series || "-" || r.model AS name, today() - r.assemblyYear, 'AUTO' FROM KillerRobots r
  WHERE r.series  = 'T' AND r.assemblyYear > 1992
) AS customer
JOIN Houses h ON h.OWNER = customer.ID AND h.hasChargingPort=true
JOIN PricingYears p ON customer.age > p.startYear && customer.age < p.endYear
```

in case of http://api.yetti__customers

```
SELECT customer.name, customer.age, customer.membership, customer.id, h.id AS hid
FROM
(
  SELECT r.uniqueGruntingSound AS name, y.age, 'YETT'
  FROM Yetti y
) AS customer
JOIN Houses h ON h.OWNER = customer.ID AND (h.origin = 'Canada' OR h.origin = 'Russia')
JOIN PricingYears p ON customer.age > p.startYear && customer.age < p.endYear
```

Code Sample: query_series.sql

IF WE ONLY HAD POLYMORPHISM... BUT DATABASES CAN'T DO IT!

in case of http://api.human__customers

```
customerMembership(
  (SELECT Humans h WHERE h.segment = 'h').toCustomer(h.firstName || ' ' || h.lastName, h.membership, 1982)
  UNION
  (SELECT SuperHumans h WHERE r.side  = 'g').toCustomer(h.heroName, 'PLAT', 1982),
  true,
  CASE WHEN p.pricing == 'sane' THEN customer.membership ELSE p.insaneMembership
)
```

in case of http://api.robot__customers

```
customerMembership(
  (SELECT Robots r).toCustomer(r.label, 1990)
  UNION
  (SELECT KillerRobots r WHERE r.series  = 'T').toCustomer(r.series || "-" || r.model, 1992),
  h.hasChargingPort = true,
  CASE WHEN p.voltage = 120 THEN 'USA' ELSE 'EU'
)
```

in case of http://api.yetti__customers

```
customerMembership(
  (SELECT Yetti y).toCustomer
  h.origin = 'Canada' OR h.origin = 'Russia',
  true
)
```

QUILL CAN!

in case of http://api.human__customers

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true,
 (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership
)
```

in case of http://api.robot__customers

```
customerMembership {
  query[Robot].toCustomer(r => r.model, 1990)
  ++
  query[KillerRobot].toCustomer(r => r.series + "-" + r.model, 1992)
}(_.hasChargingPort == true
 (c, p) => if p.voltage == 120 then "US" else "EU"
)
```

in case of http://api.yetti__customers

```
customerMembership {
  query[Yetti].toCustomer
}(h => h.origin == "Canada" || h.origin == "Russia"
 (c, p) => c.membership
)
```

Code Sample: ExampleQueries.scala

QUILL CAN!

in case of http://api.human__customers

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

in case of http://api.robot__customers

```
customerMembership {
  query[Robot].toCustomer(r => r.model, 1990)
  ++
  query[KillerRobot].toCustomer(r => r.series + "-" + r.model, 1992)
}(_.hasChargingPort == true
 (c, p) => if p.voltage == 120 then "US" else "EU"
)
```

in case of http://api.yetti__customers

```
customerMembership {
  query[Yetti].toCustomer
}(h => h.origin == "Canada" || h.origin == "Russia"
 (c, p) => c.membership
)
```

Code Sample: ExampleQueries.scala

FROM THIS...

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

TO THIS!


```
SELECT customer.name, customer.age,
  CASE WHEN p.pricing == 'sane' THEN p. customer.membership ELSE p.insaneMembership END as membership,
  customer.id, h.id AS hid FROM
(
  SELECT h.firstName || ' ' || h.lastName AS name, h.age, h.membership, h.id FROM Humans h
  WHERE h.segment = 'h' AND h.age > 1982
  UNION
  SELECT h.heroName AS name, h.age, 'PLAT', h.id FROM SuperHumans h
  WHERE h.side = 'g' AND h.age > 1856
) AS customer
JOIN Houses h ON h.OWNER = customer.ID
JOIN PricingYears p ON customer.age > p.startYear && customer.age < p.endYear
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

HOW DO YOU KNOW THAT THIS... HAPPENS? MAYBE IT'S THIS!

```
select x2.x3,
       x2.x4,
       (case when (x5."pricing" = 'sane') then x2.x6 else x5."insaneMembership" end),
       x2.x7,
       x8."id"
from ((select "id" as x7, ("firstName" || ' ') || "lastName" as x3, "age" as x4, "membership" as x6
     from "Human" where ("segment" = 'h') and ("age" > ?))
     union all
     (select "id" as x7, "heroName" as x3, "age" as x4, ? as x6
     from "SuperHuman" where ("side" = 'g') and ("age" > ?))) x2,
    "Houses" x8, "PrincingYears" x5
where ((x8."owner" = x2.x7) and ?)
  and ((x2.x4 > x5."startYear") and (x2.x4 < x5."endYear"))
```

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

AGAIN... HOW DO WE KNOW?

```
SELECT customer.name, customer.age,
  CASE WHEN p.pricing == 'sane' THEN p. customer.membership ELSE p.insaneMembership END as membership,
  customer.id, h.id AS hid FROM
(
  SELECT h.firstName || ' ' || h.lastName AS name, h.age, h.membership, h.id FROM Humans h
  WHERE h.segment = 'h' AND h.age > 1982
  UNION
  SELECT h.heroName AS name, h.age, 'PLAT', h.id FROM SuperHumans h
  WHERE h.side = 'g' AND h.age > 1856
) AS customer
JOIN Houses h ON h.OWNER = customer.ID
JOIN PricingYears p ON customer.age > p.startYear && customer.age < p.endYear
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

HOW DO YOU KNOW THAT THIS... HAPPENS? THAT'S HOW!

```
sbt:example-project> compile
[info] -- Info: /Users/path/to/my/code/Main.scala:180:29
[info] 180 |  def m4: Unit = println(run {
[info]     |                         ^
[info]     |             Quill Query: SELECT customer.name AS _1, customer.age AS _2, customer.membership AS _3, customer.id AS _4, h2.id AS _5
[info]     |             FROM
[info]     |               ((SELECT h.id, (h.firstName || ' ') || h.lastName AS name, h.age, h.membership
[info]     |                 FROM Human h WHERE h.segment = 'h' AND h.age > 1982)
[info]     |                 UNION ALL
[info]     |               (SELECT h1.id, h1.heroName AS name, h1.age, 'PLAT' AS membership
[info]     |                 FROM SuperHuman h1 WHERE h1.side = 'g' AND h1.age > 1856)
[info]     |               ) AS customer
[info]     |               INNER JOIN Houses h2 ON h2.owner = customer.id
[info]     |               AND true
[info]     |               INNER JOIN PricingYears p ON customer.age > p.startYear
[info]     |               AND customer.age < p.endYear
[info] 95 |    customerMembership {
[info] 96 |      query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
[info] 97 |      ++
[info] 98 |      query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
[info] 99 |    }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
[info] 100 |  })
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

The Flatter, the Better

Query Compilation Based on the Flattening Transformation

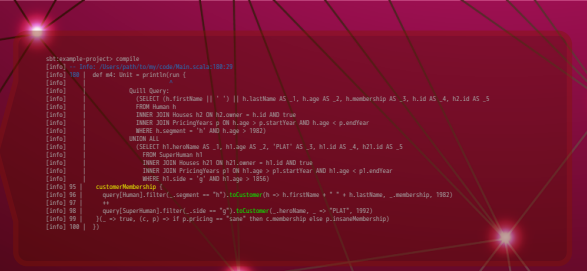
Alexander Ulrich Torsten Grust
Universität Tübingen
Tübingen, Germany
[alexander.ulrich, torsten.grust]@uni-tuebingen.de

CAN MAKE OPTIMIZATIONS! FLATTER => BETTER

```
sbt:example-project> compile
[info] -- Info: /Users/path/to/my/code/Main.scala:180:29
[info] 180 |  def m4: Unit = println(run {
[info]     |                         ^
[info]     |             Quill Query:
[info]     |               (SELECT (h.firstName || ' ') || h.lastName AS _1, h.age AS _2, h.membership AS _3, h.id AS _4, h2.id AS _5
[info]     |               FROM Human h
[info]     |               INNER JOIN Houses h2 ON h2.owner = h.id AND true
[info]     |               INNER JOIN PricingYears p ON h.age > p.startYear AND h.age < p.endYear
[info]     |               WHERE h.segment = 'h' AND h.age > 1982)
[info]     |             UNION ALL
[info]     |               (SELECT h1.heroName AS _1, h1.age AS _2, 'PLAT' AS _3, h1.id AS _4, h21.id AS _5
[info]     |                 FROM SuperHuman h1
[info]     |                 INNER JOIN Houses h21 ON h21.owner = h1.id AND true
[info]     |                 INNER JOIN PricingYears p1 ON h1.age > p1.startYear AND h1.age < p1.endYear
[info]     |                 WHERE h1.side = 'g' AND h1.age > 1856)
[info] 95 |    customerMembership {
[info] 96 |      query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
[info] 97 |      ++
[info] 98 |      query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
[info] 99 |    }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
[info] 100 |  })
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```



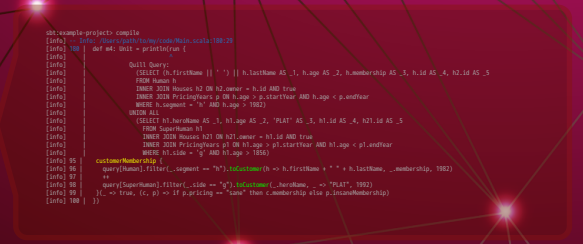
IN SCALA 2 EVERYTHING NEEDS TO BE IN `QUOTE`

```
def customerMembership = quote {
    (cs: Query[Customer],
     housesFilter: Houses => Boolean,
     membershipFunction: (Customer, PricingYears) => String
    ) =>
  for {
    customer <- cs
    h        <- query[Houses].join(h => h.owner == customer.id && housesFilter(h))
    p        <- query[PricingYears].join(p => customer.age > p.startYear && customer.age < p.endYear)
  } yield Record(customer.name, customer.age, customer.membership, customer.id, h.id)
}
```

```
implicit class Ops[H <: HumanLike](q: Query[H]) {
  def toCustomer = quote {
    (name: H => String, membership: H => String, age: Int) =>
        q.filter(h => h.age > age)
          .map(h => Customer(h.id, name(h), h.age, membership(h)))
  }
}
```

Code Sample: ExampleQueries.scala

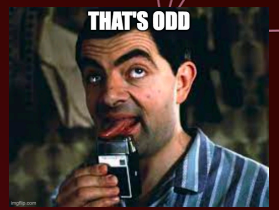
```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```



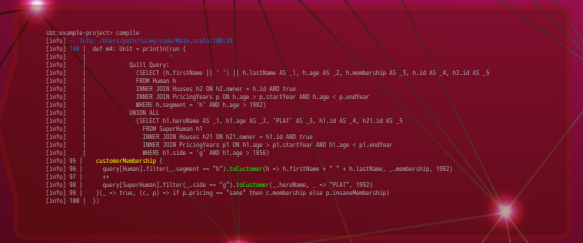
IN SCALA 2 EVERYTHING NEEDS TO BE IN `QUOTE`

```
def customerMembership = quote {
    (cs: Query[Customer],
     housesFilter: Houses => Boolean,
     membershipFunction: (Customer, PricingYears) => String
    ) =>
  for {
    customer <- cs
    h        <- query[Houses].join(h => h.owner == customer.id && housesFilter(h))
    p        <- query[PricingYears].join(p => customer.age > p.startYear && customer.age < p.endYear)
  } yield Record(customer.name, customer.age, customer.membership, customer.id, h.id)
}
```

```
implicit class Ops[H <: HumanLike](q: Query[H]) {
  def toCustomer = quote {
    (name: H => String, membership: H => String, age: Int) =>
        q.filter(h => h.age > age)
          .map(h => Customer(h.id, name(h), h.age, membership(h)))
  }
}
```



```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

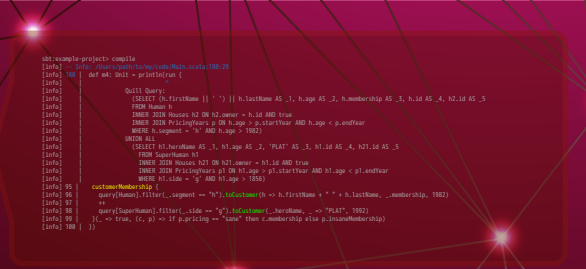


ALL YOU NEED IS A LITTLE BIT OF... IMPLICIT CLASSES?

```
implicit class Ops[H <: HumanLike](q: Query[H]) {
  def toCustomer = quote {
    (name: H => String, membership: H => String, age: Int) =>
        q.filter(h => h.age > age)
          .map(h => Customer(h.id, name(h), h.age, membership(h)))
  }
}
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

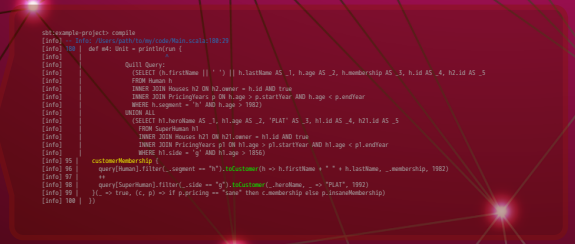


ALL YOU NEED IS A LITTLE BIT OF... ~~IMPLICIT CLASSES~~?

```
implicit class Ops[H <: HumanLike](q: Query[H]) {
  def toCustomer = quote {
    (name: H => String, membership: H => String, age: Int) =>
        q.filter(h => h.age > age)
          .map(h => Customer(h.id, name(h), h.age, membership(h)))
  }
}
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

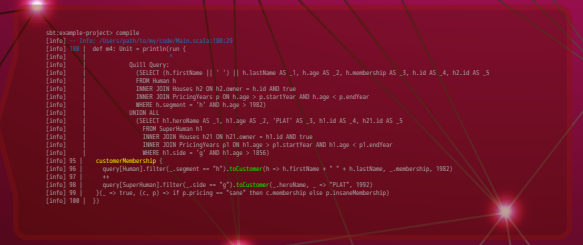


ALL YOU NEED IS A LITTLE BIT OF... ~~IMPLICIT CLASSES~~?

```
extension [H <: HumanLike](inline q: Query[H])
  inline def toCustomer = quote {
    (name: H => String, membership: H => String, age: Int) =>
        q.filter(h => h.age > age)
          .map(h => Customer(h.id, name(h), h.age, membership(h)))
  }
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

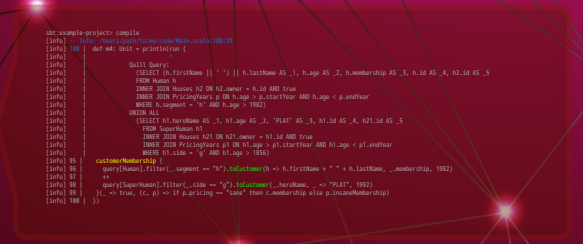


INLINE SWAPS IN THE VALUE AT COMPILE-TIME

```
extension [H <: HumanLike](inline q: Query[H])
  inline def toCustomer = quote {
    (name: H => String, membership: H => String, age: Int) =>
        q.filter(h => h.age > age)
          .map(h => Customer(h.id, name(h), h.age, membership(h)))
  }
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

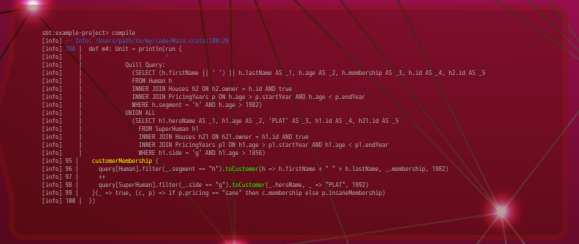


IN SCALA 3 CAN HAVE TYPE ANNOTATIONS!

```
extension [H <: HumanLike](inline q: Query[H])
  inline def toCustomer: Quoted[Query[Customer]] = quote {
    (name: H => String, membership: H => String, age: Int) =>
        q.filter(h => h.age > age)
          .map(h => Customer(h.id, name(h), h.age, membership(h)))
  }
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

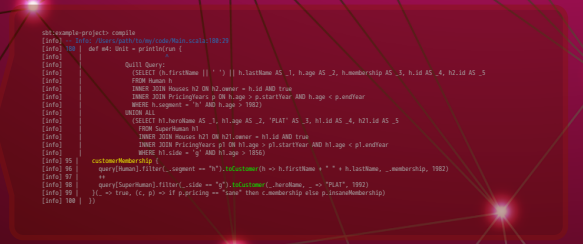


ALL YOU NEED IS A LITTLE BIT OF... QUOTES?

```
extension [H <: HumanLike](inline q: Query[H])
  inline def toCustomer = quote {
    (name: H => String, membership: H => String, age: Int) =>
        q.filter(h => h.age > age)
          .map(h => Customer(h.id, name(h), h.age, membership(h)))
  }
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

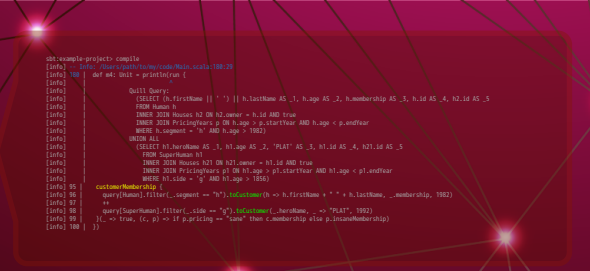


ALL YOU NEED IS A LITTLE BIT OF... QUOTES?

```
extension [H <: HumanLike](inline q: Query[H])
  inline def toCustomer =
    (name: H => String, membership: H => String, age: Int) =>
      q.filter(h => h.age > age)
        .map(h => Customer(h.id, name(h), h.age, membership(h)))
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

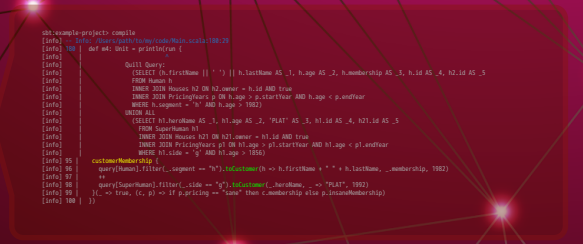


ALL YOU NEED IS A LITTLE BIT OF...

```
extension [H <: HumanLike](inline q: Query[H])
  inline def toCustomer =
    (name: H => String, membership: H => String, age: Int) =>
      q.filter(h => h.age > age)
        .map(h => Customer(h.id, name(h), h.age, membership(h)))
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  query[Human].filter(_.segment == "h").toCustomer(h => h.firstName + " " + h.lastName, _.membership, 1982)
  ++
  query[SuperHuman].filter(_.side == "g").toCustomer(_.heroName, _ => "PLAT", 1992)
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

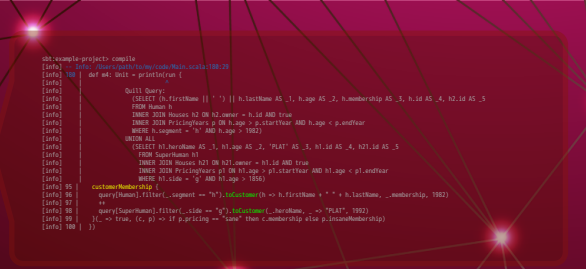


ALL YOU NEED IS A LITTLE BIT OF...

```
extension [H <: HumanLike](inline q: Query[H])
  inline def toCustomer(inline name: H => String, inline membership: H => String, inline minAge: Int) =
    q.filter(h => h.age > minAge)
      .map(h => Customer(h.id, name(h), h.age, membership(h)))
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  query[Human].toCustomer
  ++
  query[SuperHuman].toCustomer
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

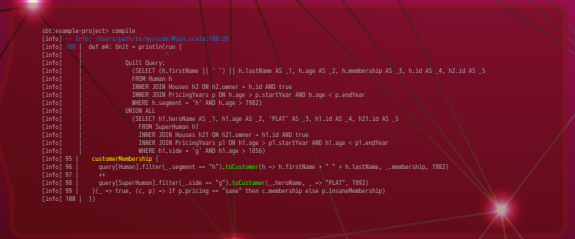


ALL YOU NEED IS A LITTLE BIT OF... TYPECLASSES?

```
trait HumanLikeFilter[H]:
  extension (h: Query[H]) def toCustomer: Query[Customer]

inline given HumanLikeFilter[Human] with
  extension (h: Query[Human])
    inline def toCustomer: Query[Customer] =
      query[Human]
        .filter(h => h.segment == "h" && h.age > 1982)
        .map(h => Customer(h.id, h.firstName + " " + h.lastName, h.age, h.membership))

inline given HumanLikeFilter[SuperHuman] with
  extension (h: Query[SuperHuman])
    inline def toCustomer: Query[Customer] =
      query[SuperHuman]
        .filter(h => h.side == "h" && h.age > 1992)
        .map(h => Customer(h.id, h.heroName, h.age, "PLAT"))
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  humanCustomer(HumanType.Regular("h", 1982))
  ++
  humanCustomer(HumanType.Super("g", 1856))
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```



ALL YOU NEED IS A LITTLE BIT OF...

```
enum HumanType:
  case Regular(seg: String, year: Int)
  case Super(side: String, year: Int)

inline def humanCustomer(inline tpe: HumanType) =
  inline tpe match
    case HumanType.Regular(seg, year) =>
      query[Human]
        .filter(h => h.segment == seg && h.age > year)
        .map(h => Customer(h.id, h.firstName + " " + h.lastName, h.age, h.membership))
    case HumanType.Super(side, year) =>
      query[SuperHuman]
        .filter(h => h.side == side && h.age > year)
        .map(h => Customer(h.id, h.heroName, h.age, "PLAT"))
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  humanCustomer(HumanType.Regular("h", 1982))
  ++
  humanCustomer(HumanType.Super("g", 1856))
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

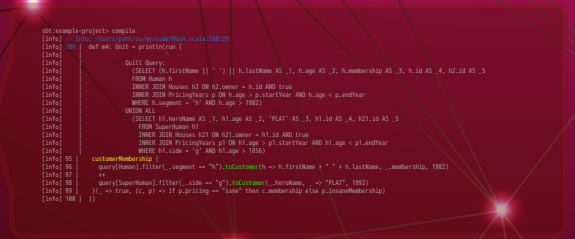
Warning. Dynamic Query

ALL YOU NEED IS A LITTLE BIT OF...

```
enum HumanType:
  case Regular(seg: String, year: Int)
  case Super(side: String, year: Int)

inline def humanCustomer(tpe: HumanType) =
  tpe match
    case HumanType.Regular(seg, year) =>
      query[Human]
        .filter(h => h.segment == seg && h.age > year)
        .map(h => Customer(h.id, h.firstName + " " + h.lastName, h.age, h.membership))
    case HumanType.Super(side, year) =>
      query[SuperHuman]
        .filter(h => h.side == side && h.age > year)
        .map(h => Customer(h.id, h.heroName, h.age, "PLAT"))
```

Code Sample: ExampleQueries.scala

```
customerMembership {
  humanCustomer(HumanType.Regular("h", 1982))
  ++
  humanCustomer(HumanType.Super("g", 1856))
}(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```



ALL YOU NEED IS A LITTLE BIT OF...

```
enum HumanType:
  case Regular(seg: String, year: Int)
  case Super(side: String, year: Int)

inline def humanCustomer(inline tpe: HumanType) =
  inline tpe match
    case HumanType.Regular(seg, year) =>
      query[Human]
        .filter(h => h.segment == seg && h.age > year)
        .map(h => Customer(h.id, h.firstName + " " + h.lastName, h.age, h.membership))
    case HumanType.Super(side, year) =>
      query[SuperHuman]
        .filter(h => h.side == side && h.age > year)
        .map(h => Customer(h.id, h.heroName, h.age, "PLAT"))
```

Code Sample: ExampleQueries.scala

SO WHAT NOW?

```
val result: Record =
  customerMembership {
    humanCustomer(HumanType.Regular("h", 1982))
    ++
    humanCustomer(HumanType.Super("g", 1856))
  }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
```

SO WHAT NOW?

```
val result: Record = customerMembership {...}(...)
```

SO WHAT NOW?

```
val result: Task[Record] = customerMembership {...}(...)
```

SO WHAT NOW?

```
val conn: Connection = ...
```

```
val result: Task[Record] = customerMembership {...}(...)
```

SO WHAT NOW?

```
val ctx = new SomeQuillContext[PostgresDialect, Literal](...) {
  def somewhereInside() =
    val conn: Connection = ds.openConnection; ...; conn.close()
}
```

```
val result: Task[Record] = customerMembership {...}(...)
```

SO WHAT NOW?

```
val ds: DataSource = ...
```

```
val ctx = new SomeQuillContext[PostgresDialect, Literal](ds) {
  def somewhereInside() =
    val conn: Connection = ds.openConnection; ...; conn.close()
}
```

```
val result: Task[Record] = customerMembership {...}(...)
```

SO WHAT NOW?

```scala
val ds: DataSource = ...

val ctx = new SomeQuillContext[PostgresDialect, Literal](ds) {
  def somewhereInside() =
    val conn: Connection = ds.openConnection; ...; conn.close()
}
import ctx._

val result: Task[Record] = customerMembership { lift(runtimeValue) }(...)
```

```
sbt:example-project> compile
[info] -- Info: /Users/path/to/my/code/Main.scala:180:29
[info]|  def m4: Unit = println(run {
[info]|                           ^
[info]|Quill Query:
[info]|(SELECT (h.firstName || ' ') || h.lastName, h.age, h.membership, h.id, h2.id AS
[info]|FROM Human h
[info]|INNER JOIN Houses h2 ON h2.owner = h.id AND true
[info]|INNER JOIN PricingYears p ON h.age > p.startYear AND h.age < p.endYear
[info]|WHERE h.segment = 'h' AND h.age > ?)
```

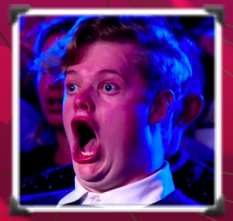
ENCODER MUST LIVE IN CONTEXT!

```
val ds: DataSource = ...

val ctx = new SomeQuillContext[PostgresDialect, Literal](ds)  {
  def somewhereInside() =
    val conn: Connection = ds.openConnection; ...; conn.close()
}
import ctx._
```

```
val result: Task[Record] = customerMembership { lift(statement.prepare(ctx.encode(runtimeValue))) }(...)
```

```
sbt:example-project> compile
[info] -- Info: /Users/path/to/my/code/Main.scala:180:29
[info]|  def m4: Unit = println(run {
[info]|                           ^
[info]|Quill Query:
[info]|(SELECT (h.firstName || ' ') || h.lastName, h.age, h.membership, h.id, h2.id AS
[info]|FROM Human h
[info]|INNER JOIN Houses h2 ON h2.owner = h.id AND true
[info]|INNER JOIN PricingYears p ON h.age > p.startYear AND h.age < p.endYear
[info]|WHERE h.segment = 'h' AND h.age > ?)
```

CAKE-PATTERN EVERYTHING???



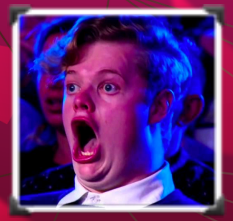
```
val ds: DataSource = ...
```

```
trait CustomerMembership:
  def customerMembership = ...
```

```
val ctx = new SomeQuillContext[PostgresDialect, Literal](ds) with CustomerMembership  {
  def somewhereInside() =
    val conn: Connection = ds.openConnection; ...; conn.close()
}
import ctx._
```

```
val result: Task[Record] = ctx.customerMembership { lift(statement.prepare(ctx.encode(runtimeValue))) }
```

CAKE-PATTERN EVERYTHING???



```
val ds: DataSource = ...

trait CustomerMembership:
  def customerMembership = ...

val ctx = new SomeQuillContext[PostgresDialect, Literal](ds) with CustomerMembership  {
  def somewhereInside() =
    val conn: Connection = ds.openConnection; ...; conn.close()
}
import ctx._

val result: Task[Record] = ctx.customerMembership { lift(statement.prepare(ctx.encode(runtimeValue))) }
```

SO WHAT NOW?

```
val ds: DataSource = ...
```

```
object Ctx extends SomeQuillContext[PostgresDialect, Literal](ds) {
  def somewhereInside() =
    val conn: Connection = ds.openConnection; ...; conn.close()
}
import Ctx._
```

```
(DataSource) => Task[Record] =
```

```
customerMembership { lift(runtimeValue) }(...)
```

STATIC!

SO WHAT NOW?

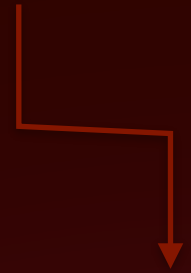
```
val ds: DataSource = ...

object Ctx extends SomeQuillContext[PostgresDialect, Literal](ds) {
  def somewhereInside() =
    val conn: Connection = ds.openConnection; ...; conn.close()
}
import Ctx._
```

Reader[DataSource, Task[Record]] = customerMembership { lift(runtimeValue) }(...)

STATIC!

ONE MORE THING...

```
val ds: DataSource = ...
```



```
Reader[DataSource, Task[Record]] = customerMembership { lift(runtimeValue) }(...)
```

STATIC!

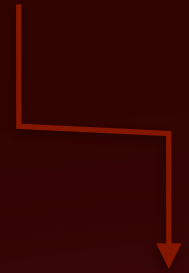
WE CAN ALSO THROW...

```
val ds: DataSource = ...
```

```
Either[SQLException, Reader[DataSource, Task[Record]]] = cM { ... }(...)
```

OR HERE?

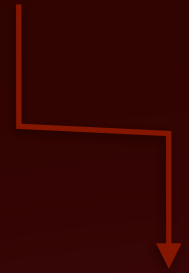
```
val ds: DataSource = ...
```



```
Reader[DataSource, Either[SQLException, Task[Record]]] = cM { ... }(...)
```

OR HERE?

```
val ds: DataSource = ...
```

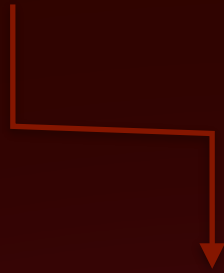


```
Reader[DataSource, Task[Either[SQLException, Record]]] = cM { ... }(...)
```

Before the task completes... or after?

IF WE JUST DO THIS...

```
val ds: DataSource = ...
```



```
OOMethod[DataSource, SQLException, Record] = customerMembership {...}(...)
```

THEN...

```
val ds: DataSource = ...
```

```
ZIO[DataSource, SQLException, Record] = customerMembership {...}(...)
```

THEN...

```
val ds: DataSource = ...
```

```
ZIO[DataSource, SQLException, Record] = customerMembership {...}(...)
```

THEN...

```
val result: ZIO[DataSource, SQLException, Record] = run(
  customerMembership {
    humanCustomer(HumanType.Regular("h", 1982))
    ++
    humanCustomer(HumanType.Super("g", 1856))
  }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
)
```

THEN...

```
val result: ZIO[Any, SQLException, Record] = run(
  customerMembership {
    humanCustomer(HumanType.Regular("h", 1982))
    ++
    humanCustomer(HumanType.Super("g", 1856))
  }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
).provideService(ds)
```

THEN...

```
val result: IO[SQLException, Record] = run(
  customerMembership {
    humanCustomer(HumanType.Regular("h", 1982))
    ++
    humanCustomer(HumanType.Super("g", 1856))
  }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
).provideService(ds)
```

USING THE MODULE PATTERN

```scala
trait DataService:
  def getCustomers: IO[SQLException, List[Record]]


final case class DataServiceLive(dataSource: DataSource):
  def getCustomers: IO[SQLException, List[Record]] =
      run(
       customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
       }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
      ).provideService(ds)
```

```scala
object DataService:
  val live = (DataServiceLive.apply _).toLayer[DataService]
```

Code Sample: simple/DataService.scala

USING THE MODULE PATTERN

```scala
trait DataService:
  def getCustomers: IO[SQLException, List[Record]]


final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers: IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
    }.provideService(dataSource)
```

```scala
object DataService:
  val live = (DataServiceLive.apply _).toLayer[DataService]
```

Code Sample: simple/DataService.scala

PLUG IN ZIO-HTTP!

```scala
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers: IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
    }.provideService(dataSource)


object DataService:
  val live = (DataServiceLive.apply _).toLayer[DataService]
```

```scala
object RestService extends ZIOAppDefault:
  given JsonEncoder[Record] = DeriveJsonEncoder.gen[Record]
  override def run =
    Server.start(
      8088,
      Http.collectZIO[Request] {
        case req @ Method.GET -> !! / "customers" =>
          ZIO.environment[DataService].flatMap(dsl =>
            dsl.get.getCustomers.map(cs => Response.json(cs.toJson))
          )
      }
    ).provide(QuillContext.dataSourceLayer, DataService.live).exitCode
```

Code Sample: simple/RestService.scala

WAIT... WE NEED FILTERS!

```scala
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers(id: Int): IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
        .filter(r => r.hid == lift(id))
    }.provideService(dataSource)
```

```scala
object RestService extends ZIOAppDefault:
  given JsonEncoder[Record] = DeriveJsonEncoder.gen[Record]
  override def run =
    Server.start(
      8088,
      Http.collectZIO[Request] {
        case req @ Method.GET -> !! / "customers" / id =>
          ZIO.environment[DataService].flatMap(dsl =>
            dsl.get.getCustomers(id.toInt).map(cs => Response.json(cs.toJson))
          )
      }
    ).provide(QuillContext.dataSourceLayer, DataService.live).exitCode
```

Code Sample: simplefilter/RestService.scala

WAIT... WE NEED FILTERS!

```scala
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers(id: Int): IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
        .filter(r => r.hid == lift(id))
    }.provideService(dataSource)
```

```
(SELECT (h.firstName || ' ') || h.lastName AS name, h.age, h.membership, h.id, h2.id AS hid
FROM Human h
        INNER JOIN Houses h2 ON h2.owner = h.id AND true
        INNER JOIN PricingYears p ON h.age > p.startYear AND h.age < p.endYear
WHERE h.segment = 'h'
 AND h.age > 1982)
UNION ALL
(SELECT h1.heroName AS name, h1.age, 'PLAT' AS membership, h1.id, h2.id AS hid
FROM SuperHuman h1
        INNER JOIN Houses h21 ON h21.owner = h1.id AND true
        INNER JOIN PricingYears p1 ON h1.age > p1.startYear AND h1.age < p1.endYear
WHERE h1.side = 'g'
 AND h1.age > 1856)
```

Code Sample: simplefilter/DataService.scala

... FILTERS MODIFY QUERIES!

```scala
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers(id: Int): IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
        .filter(r => r.hid == lift(id))
    }.provideService(dataSource)
```

```sql
SELECT r.name, r.age, r.membership, r.id, r.hid FROM (
  (SELECT (h.firstName || ' ') || h.lastName AS name, h.age, h.membership, h.id, h2.id AS hid
  FROM Human h
          INNER JOIN Houses h2 ON h2.owner = h.id AND true
          INNER JOIN PricingYears p ON h.age > p.startYear AND h.age < p.endYear
  WHERE h.segment = 'h'
   AND h.age > 1982)
  UNION ALL
  (SELECT h1.heroName AS name, h1.age, 'PLAT' AS membership, h1.id, h2.id AS hid
  FROM SuperHuman h1
          INNER JOIN Houses h21 ON h21.owner = h1.id AND true
          INNER JOIN PricingYears p1 ON h1.age > p1.startYear AND h1.age < p1.endYear
  WHERE h1.side = 'g'
   AND h1.age > 1856)
) AS r WHERE r.hid = ?
```

Code Sample: simplefilter/DataService.scala

... WE NEED DYNAMIC ONES!

```
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers(id: Int): IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
        .filter(r => r.hid == lift(id))
    }.provideService(dataSource)
```

```
SELECT r.name, r.age, r.membership, r.id, r.hid FROM (
...
) AS r WHERE r.<any-column> = ?
```

http://api.human_customers?column=value

... WE NEED DYNAMIC ONES!

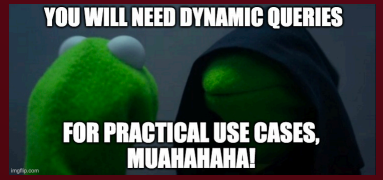
```
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers(<any-column>): IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
        .filter(r => r.<any-column> == lift(<any-column>))
    }.provideService(dataSource)
```

```
SELECT r.name, r.age, r.membership, r.id, r.hid FROM (
...
) AS r WHERE r.<any-column> = ?
```

http://api.human_customers?column=value

```
object RestService extends ZIOAppDefault:
  given JsonEncoder[Record] = DeriveJsonEncoder.gen[Record]
  override def run =
    Server.start(
      8088,
      Http.collectZIO[Request] {
        case req @ Method.GET -> !! / "customers"=>
          ZIO.environment[DataService].flatMap(dsl =>
            dsl.get.getCustomers(req.url.queryParams).map(cs => Response.json(cs.toJson))
          )
      }
    ).provide(QuillContext.dataSourceLayer, DataService.live).exitCode
```

... AND WE NEED LOTS OF THEM!




```
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers(<any-column>): IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
        .filter(r => r.<any-column> == lift(<any-column>) && r.<any-column> == lift(<any-column>)... )
    }.provideService(dataSource)
```

```
SELECT r.name, r.age, r.membership, r.id, r.hid FROM (
...
) AS r WHERE r.<any-column> = ? AND r.<any-column> = ?...
```

http://api.human_customers?columnA=valueA&columnB=valueB....

```
object RestService extends ZIOAppDefault:
  given JsonEncoder[Record] = DeriveJsonEncoder.gen[Record]
  override def run =
    Server.start(
      8088,
      Http.collectZIO[Request] {
        case req @ Method.GET -> !! / "customers"=>
          ZIO.environment[DataService].flatMap(dsl =>
            dsl.get.getCustomers(req.url.queryParams).map(cs => Response.json(cs.toJson))
          )
      }
    ).provide(QuillContext.dataSourceLayer, DataService.live).exitCode
```

... FILTER ALL THE THINGS!

```scala
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers(params: Map[String, String]): IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
        .filterByKeys(params)
    }.provideService(dataSource)
```


```
SELECT r.name, r.age, r.membership, r.id, r.hid FROM (
...
) AS r
  WHERE (r.columnA = ${map.contains("columnA")} OR ${map.contains("columnA")} IS NULL) AND
    AND (r.columnB = ${map.contains("columnB")} OR ${map.contains("columnB")} IS NULL) AND
    AND (r.columnC = ${map.contains("columnC")} OR ${map.contains("columnC")} IS NULL) AND
    ...
)
```

http://api.human_customers?columnA=valueA&columnB=valueB....

Code Sample: intermediate/DataService.scala

... FILTER ALL THE THINGS!

```scala
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers(params: Map[String, String]): IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
        .filterByKeys(params)
    }.provideService(dataSource)
```

http://api.human_customers?name=???&age=???&membership=???....

```
SELECT r.name, r.age, r.membership, r.id, r.hid FROM (
...
) AS r
  WHERE (r.name       = ${map.contains("name")}       OR ${map.contains("name")}       IS NULL) AND
    AND (r.age        = ${map.contains("age")}        OR ${map.contains("age")}        IS NULL) AND
    AND (r.membership = ${map.contains("membership")} OR ${map.contains("membership")} IS NULL) AND
    ...
)
```

Code Sample: intermediate/DataService.scala

... FILTER ALL THE THINGS!

```scala
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers(params: Map("name"->"VALUE", age->VALUE)): IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
        .filterByKeys(params)
    }.provideService(dataSource)
```

http://api.human_customers?name=???&age=???&membership=???....

```
SELECT r.name, r.age, r.membership, r.id, r.hid FROM (
...
) AS r
  WHERE (r.name       = ${map.contains("name")}       OR ${map.contains("name")}       IS NULL) AND
    AND (r.age        = ${map.contains("age")}        OR ${map.contains("age")}        IS NULL) AND
    AND (r.membership = ${map.contains("membership")} OR ${map.contains("membership")} IS NULL) AND
    ...
)
```

Code Sample: intermediate/DataService.scala

AN EXAMPLE...

```scala
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers(params: Map("name"->"Rich Shurman", age->69)): IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
        .filterByKeys(params)
    }.provideService(dataSource)
```


```
SELECT r.name, r.age, r.membership, r.id, r.hid FROM (                    http://api.human_customers?name=Rich%20Shurman&age=69
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 69             OR 69             IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND // QUERY PLAN KNOWS TO IGNORE membership COLUMN!
    ...
)
```

```
Subquery Scan on p  (cost=3695.46..7753.80 rows=5 width=76)
  Output: p.name, p.age, p.membership, p.id, p.hid
  Filter: (p.name = 'Rich Shurman'::text)
  ->  Append  (cost=3695.46..7742.23 rows=926 width=76)
                            ...
                           Filter: ((h.age > 40) AND ((h.segment)::text = 'h'::text) AND (((h.age)::character varying)::text = '59'::text))
        ->  Subquery Scan on "*SELECT* 2"  (cost=11.59..1941.27 rows=16 width=76)
              Output: "*SELECT* 2".name, "*SELECT* 2".age, 'PLAT'::character varying, "*SELECT* 2".id, "*SELECT* 2".hid
                                        ...
                                       Filter: ((h1.age > 166) AND ((h1.side)::text = 'g'::text) AND (((h1.age)::character varying)::text = '59'::text))
```

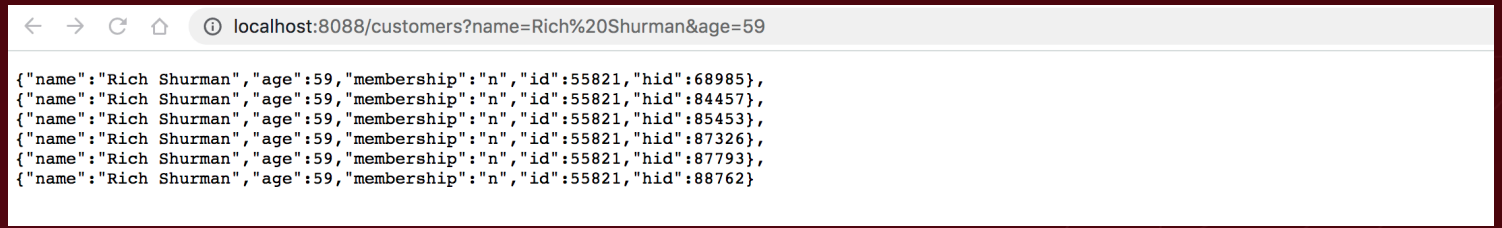
Code Sample: intermediate/DataService.scala

AN EXAMPLE...

```scala
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers(params: Map("name"->"Rich Shurman", age->69)): IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
        .filterByKeys(params)
    }.provideService(dataSource)
```

```
SELECT r.name, r.age, r.membership, r.id, r.hid FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 69             OR 69             IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND // QUERY PLAN KNOWS TO IGNORE membership COLUMN!
    ...
)
```

http://api.human_customers?name=Rich%20Shurman&age=69



Code Sample: intermediate/DataService.scala

DO SAME THING TO COLUMNS

```scala
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers(params: Map("name"->"Joe", age->123), columns: List("name", "age", "membership")): IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
        .filterByKeys(params)
        .filterColumns(columns)
    }.provideService(dataSource)
```

http://api.human_customers?show=name,age,membership...

```
SELECT
  CASE WHEN ${map.contains("name")}       THEN r.name       ELSE null,
  CASE WHEN ${map.contains("age")}        THEN r.age        ELSE null,
  CASE WHEN ${map.contains("membership")} THEN r.membership ELSE null,
  CASE WHEN ${map.contains("id")}         THEN r.id         ELSE null,
  CASE WHEN ${map.contains("hid")}        THEN r.hid        ELSE null FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 59             OR 59             IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND // QUERY PLAN KNOWS TO IGNORE membership COLUMN!
    ...
)
```

JSON:

```
{ r.name (on/off), r.age (on/off), r.membership (on/off), r.id (on/off), r.hid (on/off) }
```

Code Sample: intermediate/DataService.scala

DO SAME THING TO COLUMNS

```scala
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers(params: Map("name"->"Joe", age->123), columns: List("name", "age", "membership")): IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
        .filterByKeys(params)
        .filterColumns(columns)
    }.provideService(dataSource)
```

http://api.human_customers?show=name,age,membership...

```
SELECT
  CASE WHEN true  THEN r.name       ELSE null,
  CASE WHEN true  THEN r.age        ELSE null,
  CASE WHEN true  THEN r.membership ELSE null,
  CASE WHEN false THEN r.id         ELSE null,
  CASE WHEN false THEN r.hid        ELSE null FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 59             OR 59             IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND // QUERY PLAN KNOWS TO IGNORE membership COLUMN!
    ...
)
```

JSON:

{ r.name (on/off), r.age (on/off), r.membership (on/off), r.id (on/off), r.hid (on/off) }

Code Sample: intermediate/DataService.scala

DO SAME THING TO COLUMNS

```scala
final case class DataServiceLive(dataSource: DataSource) extends DataService:
  def getCustomers(params: Map("name"->"Joe", age->123), columns: List("name", "age", "membership")): IO[SQLException, List[Record]] =
    run {
      customerMembership {
        humanCustomer(HumanType.Regular("h", 1982))
        ++
        humanCustomer(HumanType.Super("g", 1856))
      }(_ => true, (c, p) => if p.pricing == "sane" then c.membership else p.insaneMembership)
        .filterByKeys(params)
        .filterColumns(columns)
    }.provideService(dataSource)
```

http://api.human_customers?show=name,age,membership...

QUERY PLANNER

```
SELECT
  CASE WHEN true  THEN r.name       ELSE null,
  CASE WHEN true  THEN r.age        ELSE null,
  CASE WHEN true  THEN r.membership ELSE null,
  CASE WHEN false THEN r.id         ELSE null,
  CASE WHEN false THEN r.hid        ELSE null FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 59             OR 59             IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND
    ...
)
```

```
SELECT
  r.name,
  r.age,
  r.membership,
  null,
  null FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 59             OR 59             IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND
    ...
)
```

JSON:

```
{ r.name (on), r.age (on), r.membership (on), r.id (off), r.hid (off) }
```

Code Sample: intermediate/DataService.scala

DO SAME THING TO COLUMNS

http://api.human_customers?show=name,age,membership...

```
SELECT
  CASE WHEN true  THEN r.name       ELSE null,
  CASE WHEN true  THEN r.age        ELSE null,
  CASE WHEN true  THEN r.membership ELSE null,
  CASE WHEN false THEN r.id         ELSE null,
  CASE WHEN false THEN r.hid        ELSE null FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 59             OR 59             IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND
    ...
)
```

QUERY PLANNER

```
SELECT
  r.name,
  r.age,
  r.membership,
  null,
  null FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 59             OR 59             IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND
    ...
)
```

```
Subquery Scan on p  (cost=3695.46..7753.80 rows=5 width=76)
  Output: p.name, p.age, p.membership, NULL::integer, NULL::integer
  Filter: (p.name = 'Rich Shurman'::text)
  ->  Append  (cost=3695.46..7742.23 rows=926 width=76)
        ->  Nested Loop  (cost=3695.46..5791.86 rows=910 width=46)
              Output: (((h.firstname)::text || ' '::text) || (h.lastname)::text), h.age, h.membership, h.id, h2.id
              Join Filter: ((h.age < p_1.endyear) AND ((2022 - h.age) > p_1.startyear))
              ->  Hash Join  (cost=3695.46..5612.63 rows=117 width=27)
                                ...
                                Filter: ((h.age > 40) AND ((h.segment)::text = 'h'::text) AND (((h.age)::character varying)::text = '59'::text))
        ->  Subquery Scan on "*SELECT* 2"  (cost=11.59..1941.27 rows=16 width=76)
              Output: "*SELECT* 2".name, "*SELECT* 2".age, 'PLAT'::character varying, "*SELECT* 2".id, "*SELECT* 2".hid
              ->  Nested Loop  (cost=11.59..1941.11 rows=16 width=560)
                    Output: h1.heroname, h1.age, 'PLAT'::character varying, h1.id, h21.id
                    Join Filter: ((h1.age < p1.endyear) AND ((2022 - h1.age) > p1.startyear))
                    ->  Seq Scan on public.pricingyears p1  (cost=0.00..10.70 rows=70 width=8)
                          Output: p1.startyear, p1.endyear, p1.pricing, p1.insanemembership, p1.voltage
                                        ...
                                        Filter: ((h1.age > 166) AND ((h1.side)::text = 'g'::text) AND (((h1.age)::character varying)::text = '59'::text))
```

Code Sample: intermediate/DataService.scala

DO SAME THING TO COLUMNS

http://api.human_customers?show=name,age,membership...

```
SELECT
  CASE WHEN true  THEN r.name       ELSE null,
  CASE WHEN true  THEN r.age        ELSE null,
  CASE WHEN true  THEN r.membership ELSE null,
  CASE WHEN false THEN r.id         ELSE null,
  CASE WHEN false THEN r.hid        ELSE null FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 59             OR 59             IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND
    ...
)
```

QUERY PLANNER

```
SELECT
  r.name,
  r.age,
  r.membership,
  null,
  null FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 59             OR 59             IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND
    ...
)
```

```
Subquery Scan on p  (cost=3695.46..7753.80 rows=5 width=76)
  Output: p.name, p.age, p.membership, NULL::integer, NULL::integer
  Filter: (p.name = 'Rich Shurman'::text)
  ->  Append  (cost=3695.46..7742.23 rows=926 width=76)
        ->  Nested Loop  (cost=3695.46..5791.86 rows=910 width=46)
              Output: (((h.firstname)::text || ' '::text) || (h.lastname)::text), h.age, h.membership, h.id, h2.id
              Join Filter: ((h.age < p_1.endyear) AND ((2022 - h.age) > p_1.startyear))
              ->  Hash Join  (cost=3695.46..5612.63 rows=117 width=27)
                                ...
                                Filter: ((h.age > 40) AND ((h.segment)::text = 'h'::text) AND (((h.age)::character varying)::text = '59'::text))
        ->  Subquery Scan on "*SELECT* 2"  (cost=11.59..1941.27 rows=16 width=76)
              Output: "*SELECT* 2".name, "*SELECT* 2".age, 'PLAT'::character varying, "*SELECT* 2".id, "*SELECT* 2".hid
              ->  Nested Loop  (cost=11.59..1941.11 rows=16 width=560)
                    Output: h1.heroname, h1.age, 'PLAT'::character varying, h1.id, h21.id
                    Join Filter: ((h1.age < p1.endyear) AND ((2022 - h1.age) > p1.startyear))
                    ->  Seq Scan on public.pricingyears p1  (cost=0.00..10.70 rows=70 width=8)
                          Output: p1.startyear, p1.endyear, p1.pricing, p1.insanemembership, p1.voltage
                                        ...
                                        Filter: ((h1.age > 166) AND ((h1.side)::text = 'g'::text) AND (((h1.age)::character varying)::text = '59'::text))
```

JSON:

```
{ r.name (on), r.age (on), r.membership (on), r.id (off), r.hid (off) }
```

Code Sample: intermediate/DataService.scala

ENTER THE GRAPHQL

http://api.graphql

```
SELECT
  CASE WHEN true  THEN r.name       ELSE null,
  CASE WHEN true  THEN r.age        ELSE null,
  CASE WHEN true  THEN r.membership ELSE null,
  CASE WHEN false THEN r.id         ELSE null,
  CASE WHEN false THEN r.hid        ELSE null FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 59             OR 59             IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND
    ...
)
```

QUERY PLANNER

```
SELECT
  r.name,
  r.age,
  r.membership,
  null,
  null FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 59             OR 123            IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND
    ...
)
```

▶ (Run query)

```
query{
  customers(name: "Rich Shurman", age:59) {
    name
    age
    membership
  }
}
```

200 OK 1036ms CLEAR

```
{
  "data": {
    "customers": [
      {
        "name": "Rich Shurman",
        "age": 59,
        "membership": "n"
      },
      {
        "name": "Rich Shurman",
        "age": 59,
        "membership": "n"
      },
      {
        "name": "Rich Shurman",
        "age": 59,
        "membership": "n"
```

Code Sample: GraphqlService.scala

ENTER THE GHOST-DOG!

QUERY PLANNER

```
SELECT
  r.name,
  r.age,
  r.membership,
  null,
  null FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 59             OR 123            IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND
    ...
)
```

```
case class Queries(customers: Field => (ProductArgs[Record] => Task[List[Record]]))

def graphqlService(dsa: DataServiceAdvanced) =
  graphQL(
    RootResolver(
      Queries(customers =>
        (productArgs => dsa.getCustomers(productArgs.keyValues, quillColumns(customers)))
      )
    )
  ).interpreter
```

Code Sample: GraphqlService.scala

That is to say...

CALIBAN!

QUERY PLANNER

```
SELECT
  r.name,
  r.age,
  r.membership,
  null,
  null FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 59             OR 59             IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND
    ...
)
```

```
case class Queries(customers: Field => (ProductArgs[Record] => Task[List[Record]]))

def graphqlService(dsa: DataServiceAdvanced) =
  graphQL(
    RootResolver(
      Queries(customers =>
        (productArgs => dsa.getCustomers(productArgs.keyValues, quillColumns(customers)))
      )
    )
  ).interpreter
```

```
query{
  customers(name: "Rich Shurman", age:59) {
    name
    age
    membership
  }
}
```

Code Sample: GraphqlService.scala

CALIBAN -> QUERY OPTIMIZER

QUERY PLANNER

```
Subquery Scan on p  (cost=3695.46..7753.80 rows=5 width=76)
  Output: p.name, p.age, p.membership, NULL::integer, NULL::integer
  Filter: (p.name = 'Rich Shurman'::text)
  ->  Append  (cost=3695.46..7742.23 rows=926 width=76)
        ->  Nested Loop  (cost=3695.46..5791.86 rows=910 width=46)
              Output: (((h.firstname)::text || ' '::text) || (h.lastname)::text), h.age, h.membership, h.id, h2.id
              Join Filter: ((h.age < p_1.endyear) AND ((2022 - h.age) > p_1.startyear))
              ->  Hash Join  (cost=3695.46..5612.63 rows=117 width=27)
                    ...
                    Filter: ((h.age > 40) AND ((h.segment)::text = 'h'::text) AND (((h.age)::character varying)::text = '59'::text))
        ->  Subquery Scan on "*SELECT* 2"  (cost=11.59..1941.27 rows=16 width=76)
              Output: "*SELECT* 2".name, "*SELECT* 2".age, 'PLAT'::character varying, "*SELECT* 2".id, "*SELECT* 2".hid
              ->  Nested Loop  (cost=11.59..1941.11 rows=16 width=560)
                    Output: h1.heroname, h1.age, 'PLAT'::character varying, h1.id, h21.id
                    Join Filter: ((h1.age < p1.endyear) AND ((2022 - h1.age) > p1.startyear))
                    ->  Seq Scan on public.pricingyears p1  (cost=0.00..10.70 rows=70 width=8)
                          Output: p1.startyear, p1.endyear, p1.pricing, p1.insanemembership, p1.voltage
                          ...
                          Filter: ((h1.age > 166) AND ((h1.side)::text = 'g'::text) AND (((h1.age)::character varying)::text = '59'::text))
```

```
SELECT
  r.name,
  r.age,
  r.membership,
  null,
  null FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 59             OR 59             IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND
    ...
)
```

```scala
case class Queries(customers: Field => (ProductArgs[Record] => Task[List[Record]]))

def graphqlService(dsa: DataServiceAdvanced) =
  graphQL(
    RootResolver(
      Queries(customers =>
        (productArgs => dsa.getCustomers(productArgs.keyValues, quillColumns(customers)))
      )
    )
  ).interpreter
```

```
query{
  customers(name: "Rich Shurman", age:59) {
    name
    age
    membership
  }
}
```

Code Sample: GraphqlService.scala

ALL TOGETHER NOW!

QUERY PLANNER

```
Subquery Scan on p  (cost=3695.46..7753.80 rows=5 width=76)
  Output: p.name, p.age, p.membership, NULL::integer, NULL::integer
  Filter: (p.name = 'Rich Shurman'::text)
  ->  Append  (cost=3695.46..7742.23 rows=926 width=76)
        ->  Nested Loop  (cost=3695.46..5791.86 rows=910 width=46)
              Output: (((h.firstname)::text || ' '::text) || (h.lastname)::text), h.age, h.membership, h.id, h2.id
              Join Filter: ((h.age < p_1.endyear) AND ((2022 - h.age) > p_1.startyear))
              ->  Hash Join  (cost=3695.46..5612.63 rows=117 width=27)
                                ...
                          Filter: ((h.age > 40) AND ((h.segment)::text = 'h'::text) AND (((h.age)::character varying)::text = '59'::text))
        ->  Subquery Scan on "*SELECT* 2"  (cost=11.59..1941.27 rows=16 width=76)
              Output: "*SELECT* 2".name, "*SELECT* 2".age, 'PLAT'::character varying, "*SELECT* 2".id, "*SELECT* 2".hid
              ->  Nested Loop  (cost=11.59..1941.11 rows=16 width=560)
                    Output: h1.heroname, h1.age, 'PLAT'::character varying, h1.id, h21.id
                    Join Filter: ((h1.age < p1.endyear) AND ((2022 - h1.age) > p1.startyear))
                    ->  Seq Scan on public.pricingyears p1  (cost=0.00..10.70 rows=70 width=8)
                          Output: p1.startyear, p1.endyear, p1.pricing, p1.insanemembership, p1.voltage
                                      ...
                                Filter: ((h1.age > 166) AND ((h1.side)::text = 'g'::text) AND (((h1.age)::character varying)::text = '59'::text))
```

```
SELECT
  r.name,
  r.age,
  r.membership,
  null,
  null FROM (
...
) AS r
  WHERE (r.name       = 'Rich Shurman' OR 'Rich Shurman' IS NULL) AND
    AND (r.age        = 59             OR 59             IS NULL) AND
    AND (r.membership = NULL           OR NULL           IS NULL) AND
    ...
)
```

```
object GraphqlService extends ZIOAppDefault:
  case class Queries(customers: Field => (ProductArgs[Record] => Task[List[Record]]))

  def graphqlService(dsa: DataServiceAdvanced) =
    graphQL(
      RootResolver(
        Queries(customers =>
          (productArgs => dsa.getCustomers(productArgs.keyValues, quillColumns(customers)))
        )
      )
    ).interpreter

  val myApp = (for {
    dsa         <- ZIO.environment[DataServiceAdvanced]
    interpreter <- graphqlService(dsa.get)
    _ <- Server.start(
           port = 8088,
           http = Http.route[Request] { case _ -> !! / "api" / "graphql" =>
             ZHttpAdapter.makeHttpService(interpreter)
           }
         ).forever
  } yield ()).provide(QuillContext.dataSourceLayer, DataServiceAdvanced.live)

  def run = myApp.exitCode

end GraphqlService
```

```
query{
  customers(name: "Rich Shurman", age:59) {
    name
    age
    membership
  }
}
```

```
{
  "data": {
    "customers": [
      { "name": "Rich Shurman", "age": 59, "membership": "n" },
      { "name": "Rich Shurman", "age": 59, "membership": "n" },
      { "name": "Rich Shurman", "age": 59, "membership": "n" }
      ...
    ]
  }
}
```

Code Sample: GraphqlService.scala

ALL TOGETHER NOW!

DYNAMIC OPTIMIZATION

```
Subquery Scan on p  (cost=3695.46..7753.80 rows=5 width=76)
  Output: p.name, p.age, p.membership, NULL::integer, NULL::integer
  Filter: (p.name = 'Rich Shurman'::text)
  ->  Append  (cost=3695.46..7742.23 rows=926 width=76)
        ->  Nested Loop  (cost=3695.46..5791.86 rows=910 width=46)
              Output: (((h.firstname)::text || ' '::text) || (h.lastname)::text), h.age, h.membership, h.id, h2.id
              Join Filter: ((h.age < p_1.endyear) AND ((2022 - h.age) > p_1.startyear))
              ->  Hash Join  (cost=3695.46..5612.63 rows=117 width=27)
                                ...
                            Filter: ((h.age > 40) AND ((h.segment)::text = 'h'::text) AND (((h.age)::character varying)::text = '59'::text))
        ->  Subquery Scan on "*SELECT* 2"  (cost=11.59..1941.27 rows=16 width=76)
              Output: "*SELECT* 2".name, "*SELECT* 2".age, 'PLAT'::character varying, "*SELECT* 2".id, "*SELECT* 2".hid
              ->  Nested Loop  (cost=11.59..1941.11 rows=16 width=560)
                    Output: h1.heroname, h1.age, 'PLAT'::character varying, h1.id, h21.id
                    Join Filter: ((h1.age < p1.endyear) AND ((2022 - h1.age) > p1.startyear))
                    ->  Seq Scan on public.pricingyears p1  (cost=0.00..10.70 rows=70 width=8)
                          Output: p1.startyear, p1.endyear, p1.pricing, p1.insanemembership, p1.voltage
                                      ...
                                  Filter: ((h1.age > 166) AND ((h1.side)::text = 'g'::text) AND (((h1.age)::character varying)::text = '59'::text))
```

STATIC QUERY

```
sbt:quill-2> compile
[info] compiling 1 Scala source to /Users/...
[info] 41 |    run(customersWithFiltersAndColumns(params, columns)).provideService(dataSource)
[info]    |    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
[info]    | Quill Query: SELECT
[info]    |   CASE WHEN ? THEN p.name ELSE null END AS name,
[info]    |   CASE WHEN ? THEN p.age ELSE null END AS age,
[info]    |   CASE WHEN ? THEN p.membership ELSE null END AS membership,
[info]    |   CASE WHEN ? THEN p.id ELSE null END AS id,
[info]    |   CASE WHEN ? THEN p.hid ELSE null END AS hid
[info]    | FROM
[info]    |   ...
[info]    | WHERE (p.name = ? OR ? IS NULL)
[info]    |   AND (p.age as VARCHAR = ? OR ? IS NULL)
[info]    |   AND (p.membership = ? OR ? IS NULL)
[info]    |   AND (p.id as VARCHAR = ? OR ? IS NULL)
[info]    |   AND (p.hid as VARCHAR) = ? OR ? IS NULL)
```

SIMPLE REST-TO-DB API

```
object GraphqlService extends ZIOAppDefault:
  case class Queries(customers: Field => (ProductArgs[Record] => Task[List[Record]]))

  def graphqlService(dsa: DataServiceAdvanced) =
    graphQL(
      RootResolver(
        Queries(customers =>
          (productArgs => dsa.getCustomers(productArgs.keyValues, quillColumns(customers)))
        )
      )
    ).interpreter

  val myApp = (for {
    dsa         <- ZIO.environment[DataServiceAdvanced]
    interpreter <- graphqlService(dsa.get)
    _ <- Server.start(
           port = 8088,
           http = Http.route[Request] { case _ -> !! / "api" / "graphql" =>
             ZHttpAdapter.makeHttpService(interpreter)
           }
         ).forever
  } yield ()).provide(QuillContext.dataSourceLayer, DataServiceAdvanced.live)

  def run = myApp.exitCode

end GraphqlService
```

FULLY-FEATURED GRAPHQL

```
query{
  customers(name: "Rich Shurman", age:59) {
    name
    age
    membership
  }
}
```

```
{
  "data": {
    "customers": [
      { "name": "Rich Shurman", "age": 59, "membership": "n" },
      { "name": "Rich Shurman", "age": 59, "membership": "n" },
      { "name": "Rich Shurman", "age": 59, "membership": "n" }
      ...
    ]
  }
}
```

Code Sample: GraphqlService.scala

CONCLUSIONS

SCALA 2

Quoted Methods
Refined Types

SCALA 3

Inline Methods
Inline match
Inline Type Classes
Type Annotations
Extension Methods

ZIO

The most natural way
to handle Quill effects!

CALIBAN + ZIO-HTTP

Out of the Box GraphQL
handling with DB-level
optimizations!

THANKS!!

- Pierre Ricadat a.k.a. @ghostdogpr
- Adam Fraser
- Ziverge
- ZIO Community