

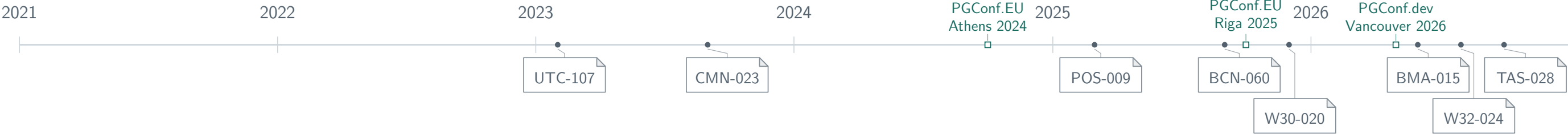
PGConf.EU · Valencia · 20 October 2026

Designing a new SQL language feature

Key joins

Five years from an idea to an SQL feature

Andreas Karlsson · Joel Jacobson



Application queries brought us here

Who are we?

Andreas Karlsson

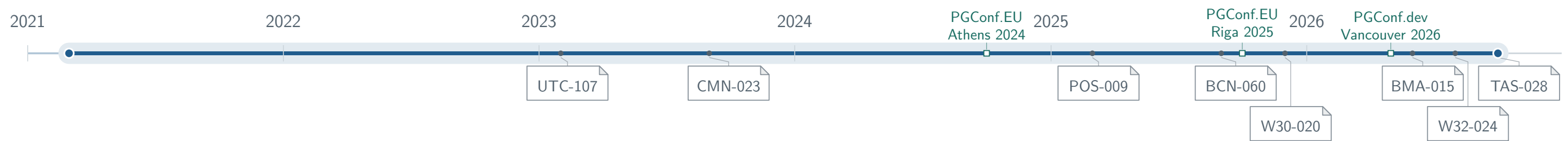
Percona

Joel Jacobson

SQL StdDev AB

Application developers. PostgreSQL contributors.

ISO/IEC JTC 1/SC 32/WG 3 committee members.



What will this talk be about?

 Key Joins paper submitted to WG3

12 OCTOBER 2026 · TASHKENT

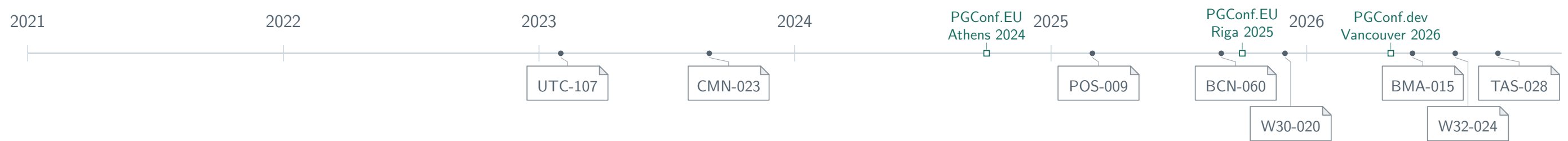
Key Joins accepted by ISO/IEC JTC 1/SC 32/WG 3

WHERE WE STARTED

A shorter way to write joins.

WHERE IT LED

New join syntax with
compile-time guarantees.



It runs. Is it right?

```
SELECT o.id,  
       SUM(oi.amount) AS item_total,  
       SUM(p.amount)  AS payment_total  
FROM orders AS o  
LEFT JOIN order_items AS oi ON oi.order_id = o.id  
LEFT JOIN payments    AS p  ON p.order_id  = o.id  
GROUP BY o.id;
```

Does this query return the correct item and payment totals for each order?

[Continue the story](#)

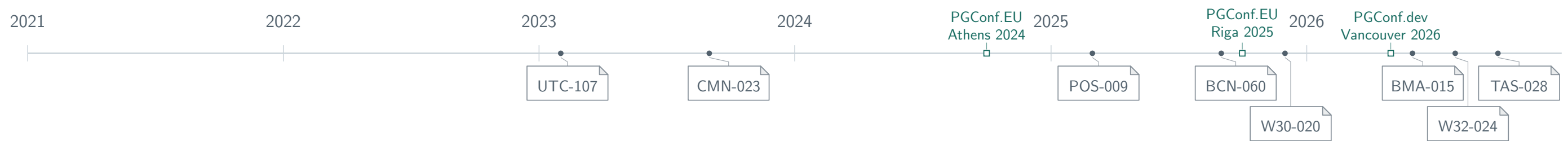
[Show how the rows multiply](#)

Catching the query's bug

```
% psql postgres
psql (20devel)
Type "help" for help.

postgres=# SELECT o.id,
        SUM(oi.amount) AS item_total,
        SUM(p.amount)  AS payment_total
FROM orders AS o
LEFT JOIN order_items AS oi FOR KEY o (id) <- oi (order_id)
LEFT JOIN payments    AS p  FOR KEY o (id) <- p (order_id)
GROUP BY o.id;
ERROR:  key join from referencing relation p to referenced relation o cannot be proven
LINE 6: LEFT JOIN payments    AS p  FOR KEY o (id) <- p (order_id)
          ^

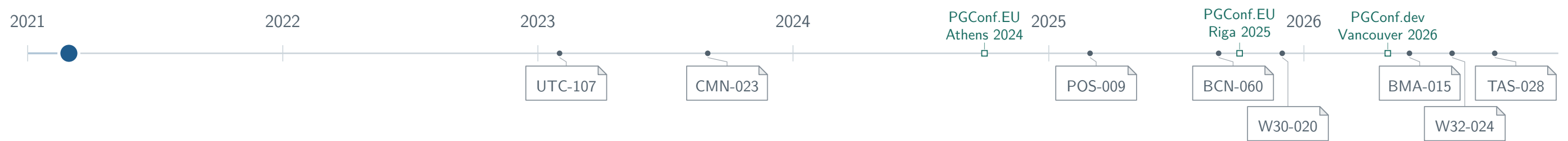
DETAIL:  Referenced columns o (id) are not proven unique. A preceding join may duplicate rows from referenced relation o.
postgres=# █
```



More joins. More to check.

```
SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM customers          AS c
JOIN countries          AS co ON co.id = c.country_id
JOIN customer_types     AS ct ON ct.id = c.customer_type_id
JOIN orders             AS o  ON c.id  = o.customer_id
JOIN order_statuses     AS os ON os.id  = o.order_status_id
LEFT JOIN discount_types AS dt ON dt.id  = o.discount_type_id
JOIN shipping_methods   AS sm ON sm.id  = o.shipping_method_id
JOIN payment_methods    AS pm ON pm.id  = o.payment_method_id
JOIN order_items        AS oi ON oi.id  = o.order_id
JOIN currencies         AS cur ON cur.id = o.currency_id
JOIN order_priorities   AS op ON op.id  = o.order_priority_id
JOIN tax_codes          AS tc ON tc.id  = o.tax_code_id
WHERE o.id = 100;
```

Now imagine these names refer to views built on other views.



27 March 2021 · Inspired by Cypher

Joel's first idea

THE CYPHER QUERY THAT INSPIRED IT

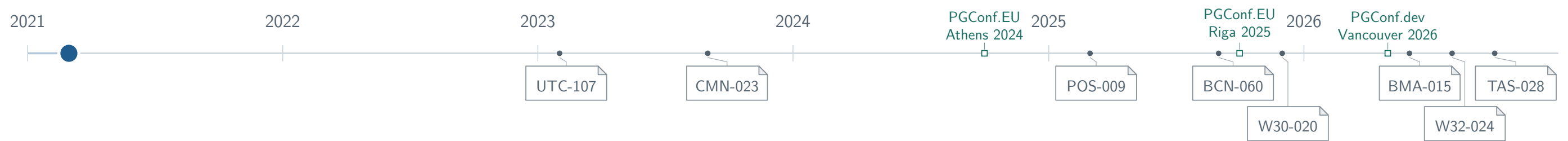
```
MATCH (p:Product {productName:"Chocolate"})<-[:PRODUCT]-(:Order)<-[:PURCHASED]-(c:Customer)
RETURN distinct c.companyName;
```

SQL WITH EXPLICIT JOINS

```
SELECT DISTINCT c.company_name
FROM customers AS c
JOIN orders AS o ON c.customer_id = o.customer_id
JOIN order_details AS od ON o.order_id = od.order_id
JOIN products AS p ON od.product_id = p.product_id
WHERE p.product_name = 'Chocolate';
```

COULD SQL FOLLOW FOREIGN KEYS WITH PATH EXPRESSIONS?

```
SELECT DISTINCT od.order_id.customer_id.company_name
FROM order_details AS od
WHERE od.product_id.product_name = 'Chocolate';
```



Late March 2021 · The problems with implicit joins

Feedback from mailinglist

“Thought experiment: how could the original syntax proposal make any use of a multi-column foreign key?”

Tom Lane

pgsql-hackers · 28 March 2021

“you just have to create one more FK, and suddenly a query that worked perfectly fine, now starts throwing errors because it’s now ambiguous.”

Álvaro Herrera

pgsql-hackers · 27 March 2021

“We’ve seen many frameworks that hide joins, and the results are ... less than universally good.”

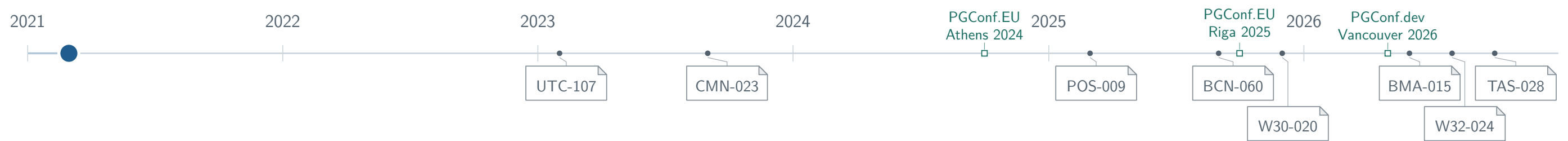
Andrew Dunstan

pgsql-hackers · 28 March 2021

“I like some mandatory redundancy in syntax, because it allows to detect some typo errors.”

Pavel Stehule

pgsql-hackers · 28 March 2021



28–31 March 2021 · Keep the join explicit

Vik’s response to Joel’s first idea

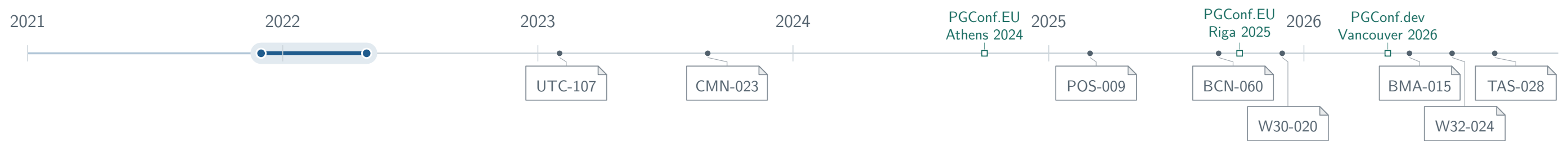
“This is a terrible idea, but let me explain why.”

“SQL is not an implicit language, though.”

Vik’s suggested alternative syntax

```
FROM a JOIN b WITH a_b_fk
```

*“I’m with you on this: let’s do it the Standard way,
or not do it at all.”*



Late 2021–May 2022 (approx.) · A declaration is not a column occurrence

Why does constraint names not work?

```
CREATE VIEW staff AS
SELECT e.dept_id AS employee_dept_id,
       m.dept_id AS manager_dept_id
FROM employees AS e
LEFT JOIN employees AS m ON e.manager_id = m.emp_id;
```

Both columns originate from the same foreign key declaration.

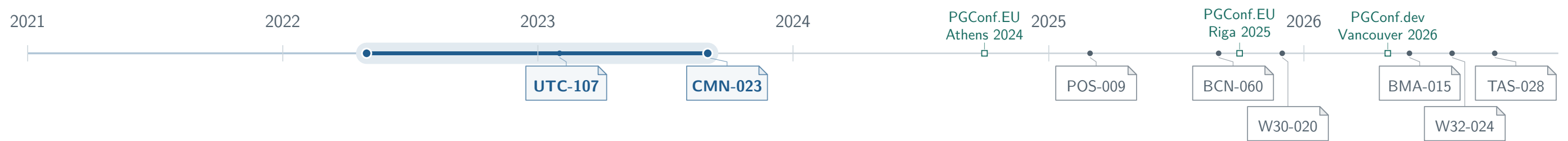
THE NAME LEAVES A CHOICE

```
FROM staff AS s
LEFT JOIN departments AS d WITH employees_dept_id_fkey
```

Employee or manager department?

COLUMNS MAKE THE CHOICE EXPLICIT

```
FROM staff AS s
LEFT JOIN departments AS d KEY (id) FROM s (employee_dept_id);
```



May 2022–September 2023 · From Sweden to WG3

Learning the standards process

2022 · SWEDEN

A proposal sent
to SIS



FEBRUARY 2023 · UTC

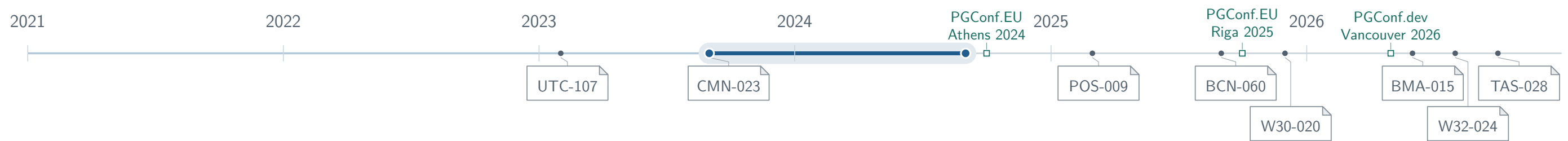
Keith presents
UTC-107



SEPTEMBER 2023 · CMN

Joel and Anders
present CMN-023

Vik offers to help. Karl and Jörn see the use case.



Late 2023–2024 (approx.) · A shortcut that did not simplify the design

Sidetrack: the MATCH predicate

The hope

Build on the **MATCH** predicate to make the proposal smaller.

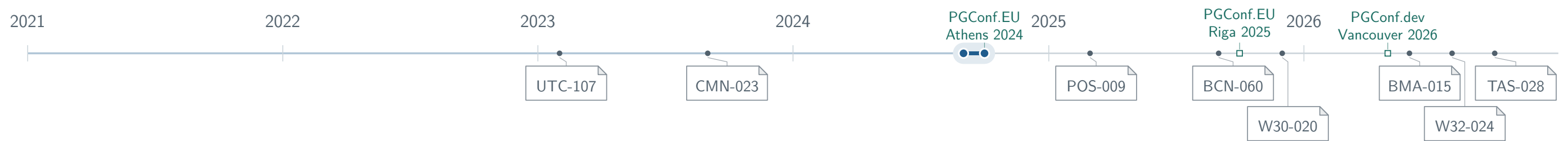
The complication

What should happen when some referencing values are **NULL**?

The insight

Transform to an ordinary equijoin.

A **NULL** in any compared column means no match.



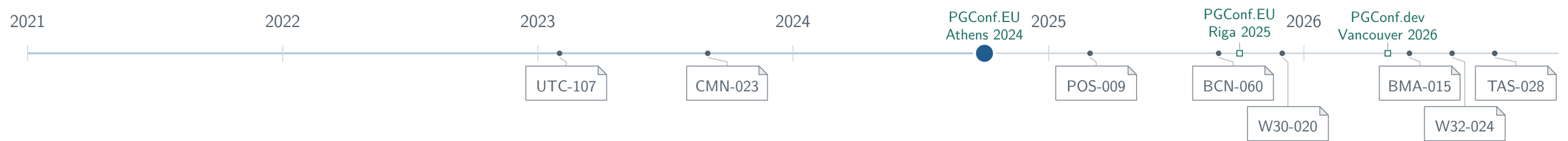
September–October 2024 · A prototype for Athens

Pitching to the PostgreSQL community

“Do you use joins?”

We wrote a small PostgreSQL prototype.
Then Joel took a laptop around PGConf.EU.

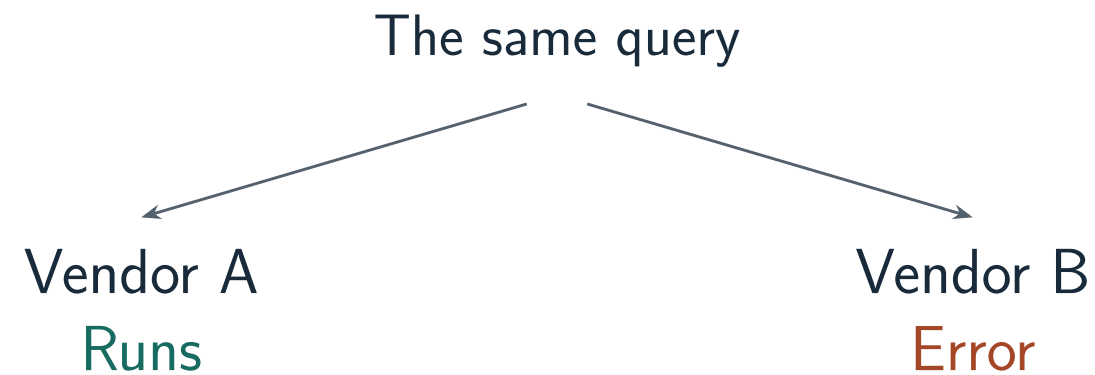
The first implementation only handled base tables.



23 October 2024 · Feedback from Peter Geoghegan

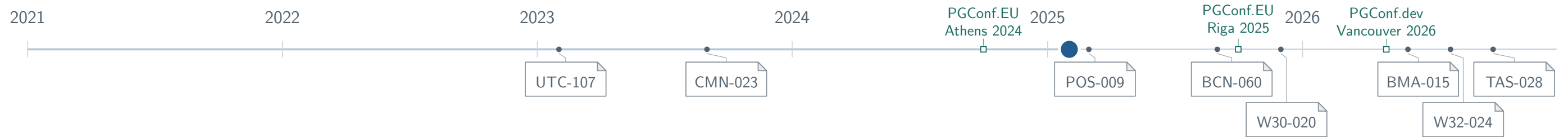
Derived tables could not wait

Peter Geoghegan pointed out the portability risk:



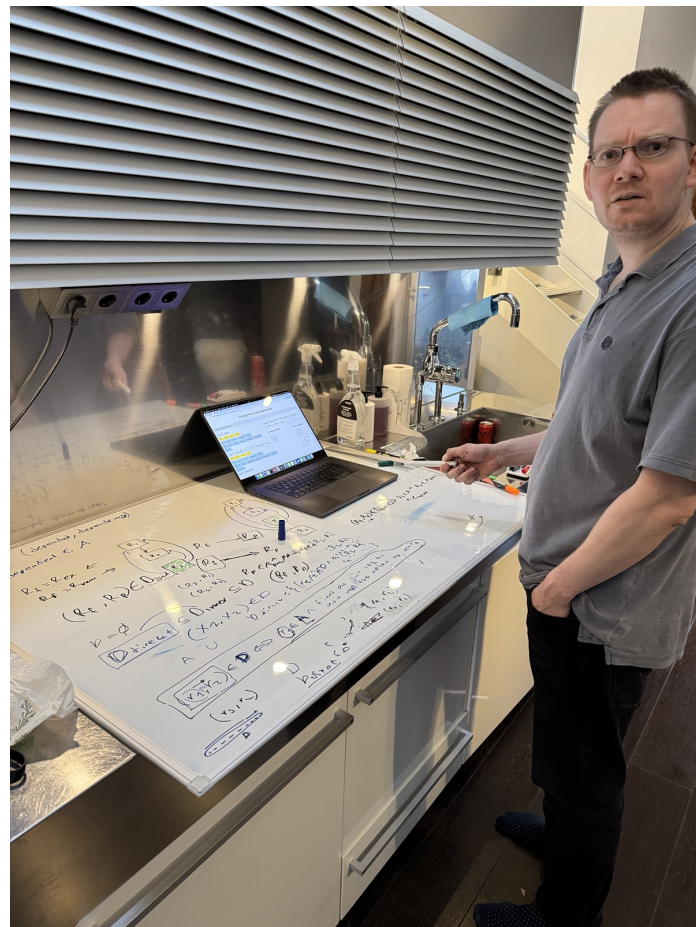
We changed direction:

Define derived-table behavior from the start.

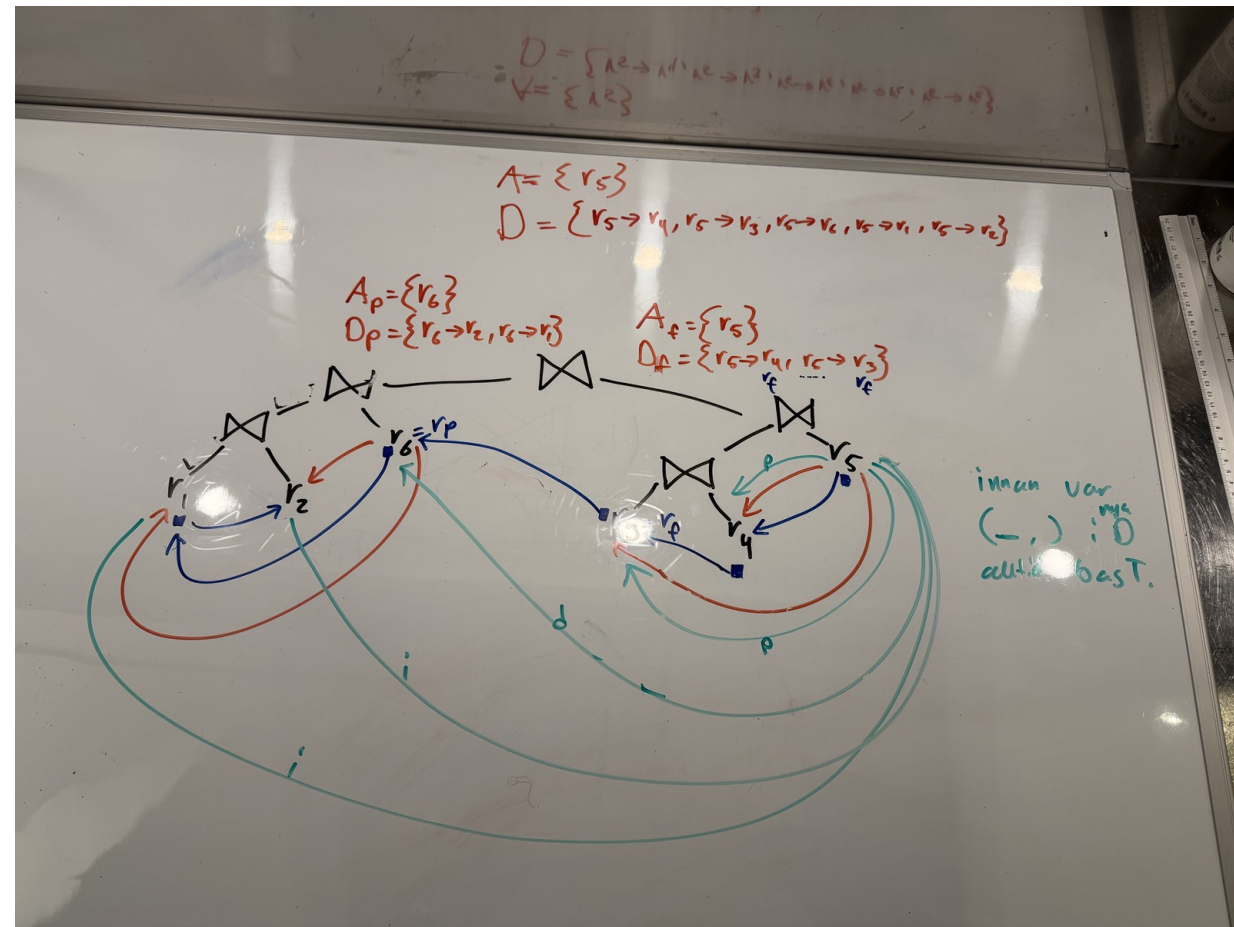


8 February 2025 · Stockholm, Sweden

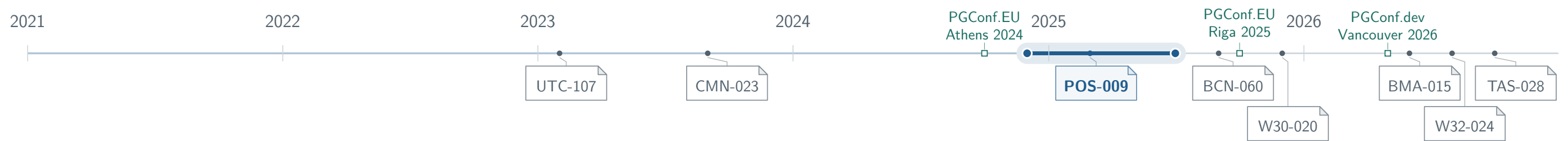
Trying to get derived tables right



Anders Granlund
Recursive proof algorithm



A join tree
Reasoning about derived tables



December 2024–July 2025 · Working with Vik and Jim

Restarting work with ISO

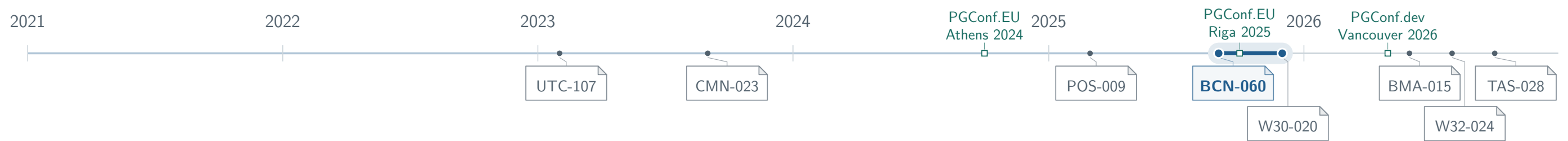
THE PLAN

A small change proposal for base tables only,
with a discussion section that explains the derived-table rules.

We thought a smaller first step would be easier to adopt.



Joel Jacobson, Vik Fearing,
Jim Melton, Fredrik Hjelm
Paris, France · 17 July 2025



September–December 2025 · PostgreSQL KEY is unreserved; FOR is reserved

KEY is not a keyword!

Peter Eisentraut spotted a grammar problem.

```
SELECT *  
FROM rooms room  
JOIN keycards key ON key.room_id = room.id;
```

Here, key is a table alias.

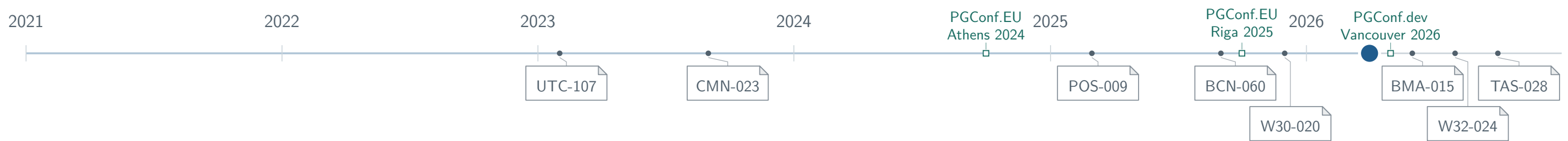
BEFORE

```
SELECT *  
FROM employees  
JOIN departments KEY (id) <- employees (dept_id);
```

AFTER

```
SELECT *  
FROM employees  
JOIN departments FOR KEY (id) <- employees (dept_id);
```

Reusing an existing reserved word, **FOR**, gives the parser an unambiguous start.



2–21 April 2026 · Anxiety and excitement at the same time

The breakthrough

Vik spots a CD ballot comment: feedback on a Committee Draft.
It hints that a similar feature may soon be proposed.

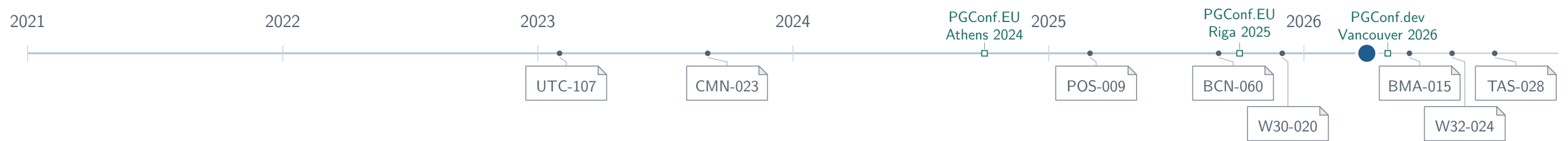
Joel wonders: *“What can we actually say about the result of each **FOR KEY** join?”*

Our rules did not yet guarantee:

- Every referencing row appears exactly once.
- Every row with no nulls in its referencing columns finds its unique match.

We realized we could change the rules to guarantee both.

On the flight to PGConf.DE, Joel realizes that taking the join type into account also removes the need for the **NULLS** keyword we had been considering.



4 April 2026 · Syntax used at the time

The Composability Bug

✓ View version: correctly rejected

```
CREATE VIEW v AS
  SELECT orders.id AS order_id
  FROM order_items
  JOIN orders FOR KEY (id) <- order_items (order_id);

SELECT * FROM v
JOIN payments FOR KEY (order_id) -> v (order_id);
```

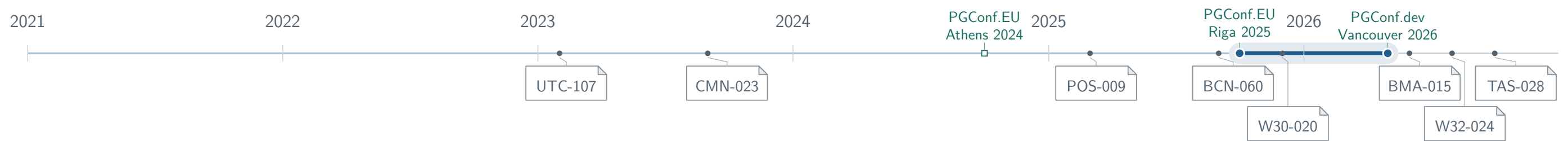
```
ERROR: foreign key join violation
DETAIL: referenced relation does not preserve
uniqueness of keys
```

✗ Inline version: incorrectly accepted

```
SELECT *
FROM order_items
JOIN orders FOR KEY (id) <- order_items (order_id)
JOIN payments FOR KEY (order_id) -> orders (id);
```

No error. Query compiles and runs.

Refactoring joins into a view should not change whether a query compiles.



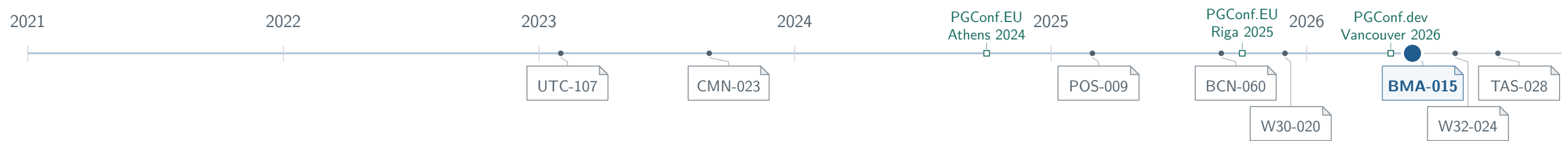
October 2025–May 2026

The road to the first patch to pgsql-hackers

We met Arne in Riga. In March, he joined the Key Joins effort.

March–April	● Arne	Strengthens query analysis and reworks stored-view revalidation.
10 April	● Joel	Fixes the composability bug: validates against the intermediate join result.
Early May	● Joel	Rewrites how the proof engine represents and propagates facts.
May	● Arne, Joel and Andreas	Arne tackles concurrency; together we test and fix view and dependency handling.

28 May: first public patch. Ready for the June meeting in Stockholm.



15–19 June 2026 · All SC 32 working groups · BMA

Stockholm: two proposals, more collaborators

JOIN TO ONE

Check at runtime that each left row has at most one match.

JOIN TO ONE is accepted.

KEY JOINS

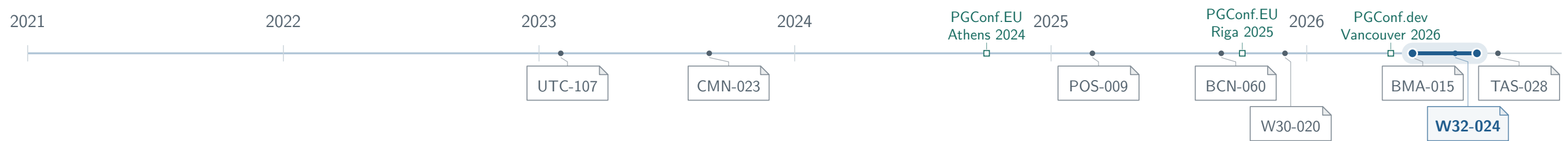
WG3 reaches consensus in support of the concept.

Our Change Proposal needs more work.

We meet Bengt Gunne from Mimer.



ISO/IEC JTC 1/SC 32 participants
Joel's house, Stockholm, Sweden
17 June 2026



June–September 2026 · A shared design effort, not just proofreading

Standardese experts

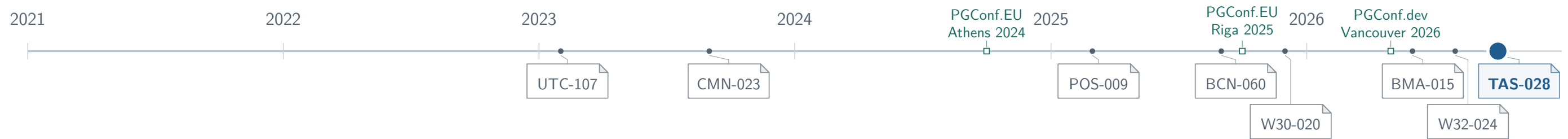
Vik Fearing · Jim Melton
Karl Schendel · Stephen Cannan
Jörn Bartels · Fred Zemke

“Let J be a <joined table> that does not simply contain a <partitioned join table>.”

Words such as “simply” have precisely defined meanings.



Stefan Plantikow, Jörn Bartels,
Jim Melton, Stephen Cannan, Vik Fearing
Editors' meeting, Åland · 23 June 2026



12 October 2026 · TAS

Acceptance in Tashkent

ISO/IEC JTC 1/SC 32/WG 3

Reached consensus to accept
TAS-028R1 — Key Joins: The Proposal

CO-AUTHORS OF TAS-028R1

Joel Jacobson
Vik Fearing
Andreas Karlsson
Arne Roland
Anders Granlund

Stephen Cannan
Karl Schendel
Jim Melton
Jörn Bartels
Fred Zemke

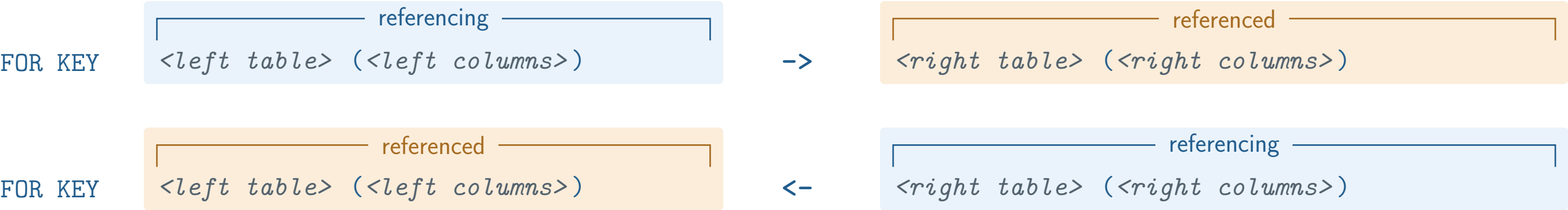


Keith Hare, Stephen Cannan, Jörn Bartels,
Jim Melton, Joel Jacobson
WG 3 meeting, Tashkent, Uzbekistan · 12 October 2026

Five years of work, many collaborators. Thank you to our hosts in Uzbekistan.

The syntax we arrived at

FOR KEY is syntax for a key join specification. The DBMS checks it before execution.



The arrow points to the column combination that must be unique.

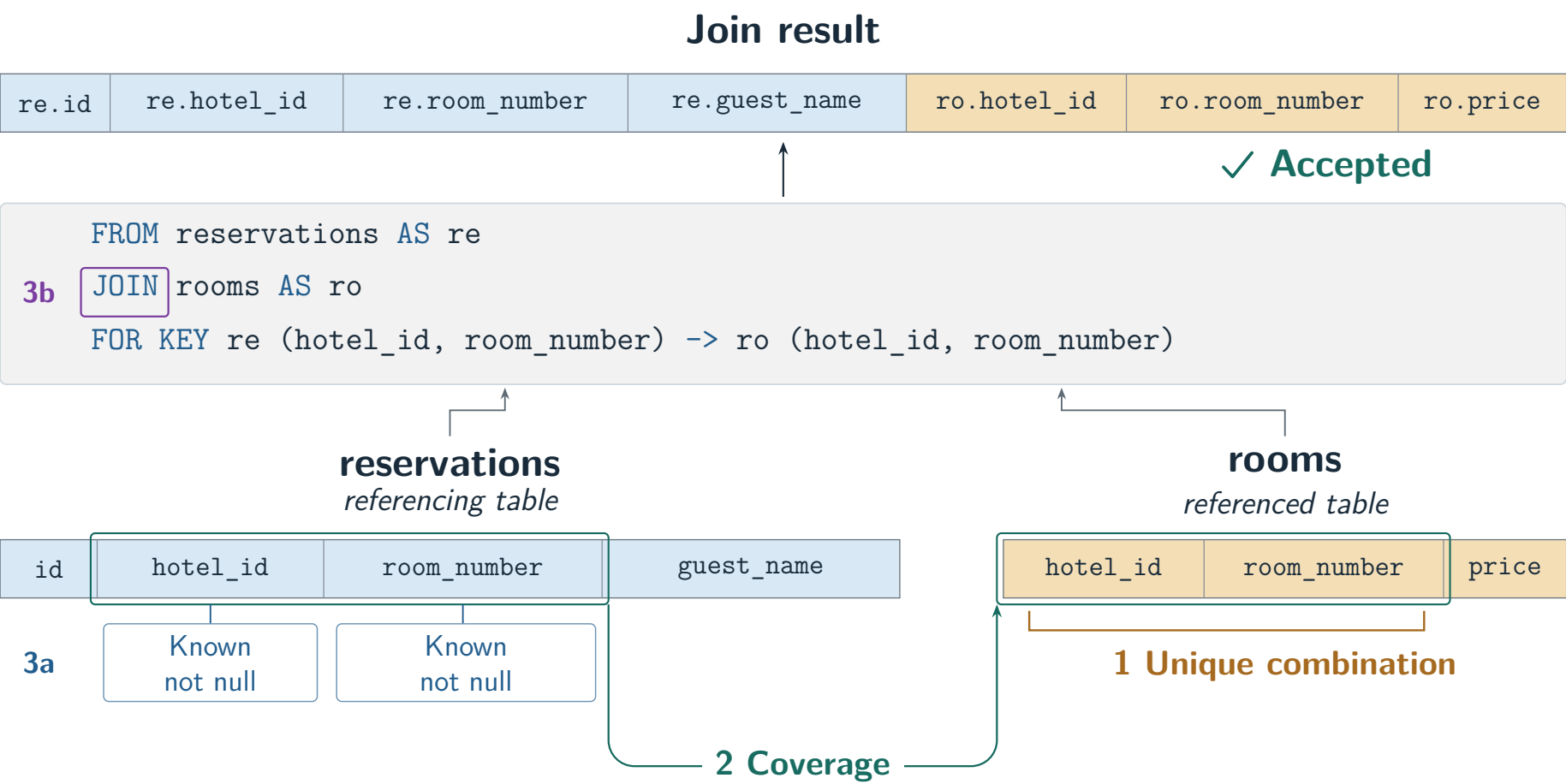
The table references and column lists follow left-to-right operand order.

In base-table declarations, the referencing table declares the **FOREIGN KEY** (a referential constraint). **REFERENCES** names the referenced table.

The DBMS proves the conditions or rejects the statement. An accepted specification becomes an **ON** condition with equality comparisons.

SUFFICIENT CONDITIONS $\Rightarrow$ **GUARANTEES**

Proving an inner key join



From **CREATE TABLE** reservations:

```
hotel_id INTEGER NOT NULL,
room_number INTEGER NOT NULL,
FOREIGN KEY (hotel_id, room_number)
REFERENCES rooms (hotel_id, room_number)
```

From **CREATE TABLE** rooms:

```
UNIQUE (hotel_id, room_number)
```

SUFFICIENT CONDITIONS

✓ **1. Uniqueness**

All-non-null combinations of referenced values are unique.

AND

✓ **2. Coverage**

Every all-non-null referencing combination has a matching referenced row.

AND

AT LEAST ONE

✓ **3a.** Every referencing column is known not null.

OR

✗ **3b.** The join type preserves the referencing operand.

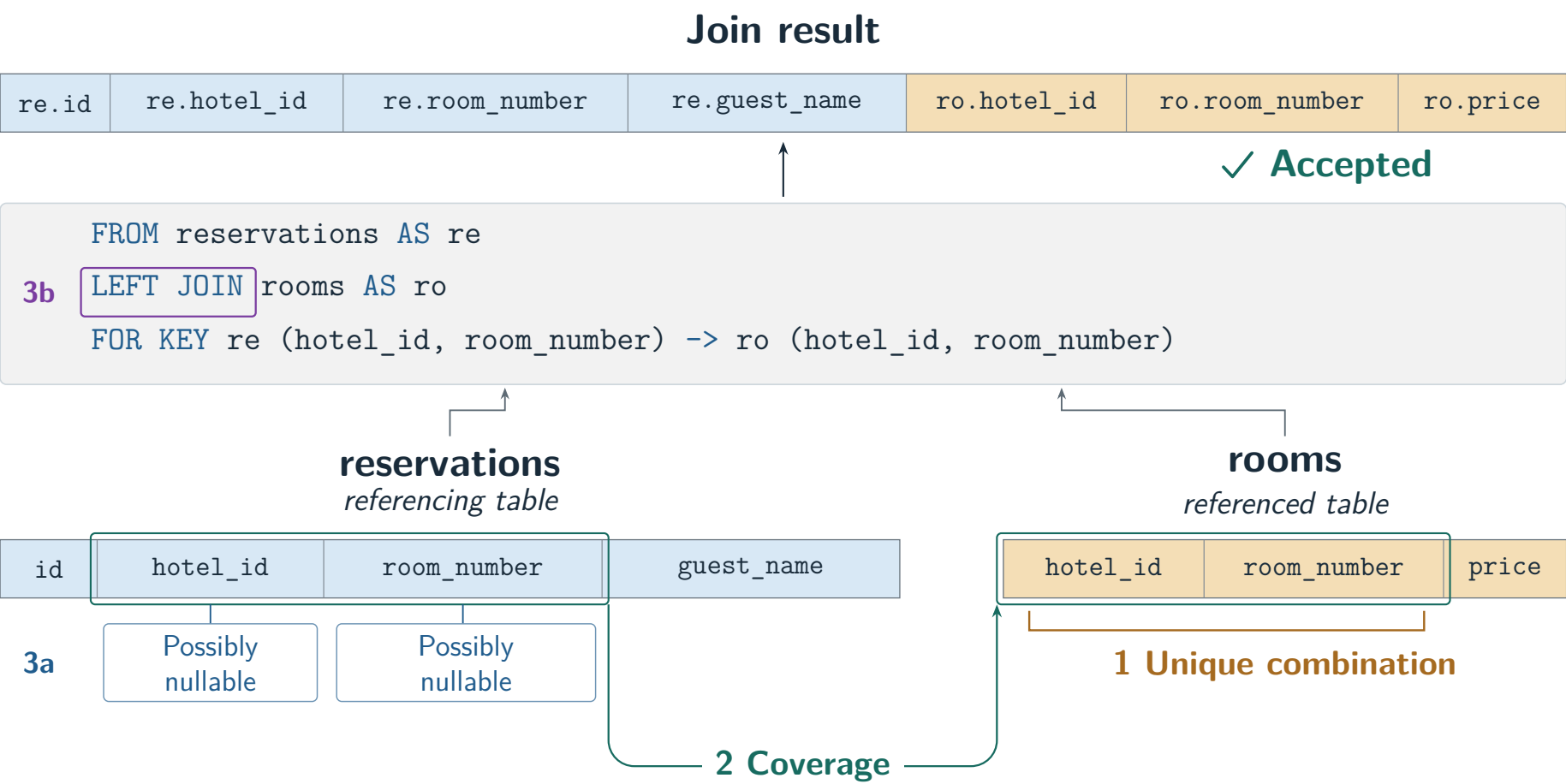


GUARANTEES

Every referencing row appears exactly once in the join result.

Rows with no nulls in their referencing columns find a match.

Preserving possibly nullable references



From **CREATE TABLE** reservations:

```
hotel_id INTEGER,
room_number INTEGER,
FOREIGN KEY (hotel_id, room_number)
  REFERENCES rooms (hotel_id, room_number)
```

From **CREATE TABLE** rooms:

```
UNIQUE (hotel_id, room_number)
```

SUFFICIENT CONDITIONS

✓ **1. Uniqueness**

All-non-null combinations of referenced values are unique.

AND

✓ **2. Coverage**

Every all-non-null referencing combination has a matching referenced row.

AND

AT LEAST ONE

✗ **3a.** Every referencing column is known not null.

OR

✓ **3b.** The join type preserves the referencing operand.



GUARANTEES

Every referencing row appears exactly once in the join result.

Rows with no nulls in their referencing columns find a match.

Is the first FROM item cardinality-preserved?

```
SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM  order_items      AS oi 3
2 JOIN  orders          AS o   FOR KEY oi (order_id)
JOIN  customers        AS c   FOR KEY o (customer_id)
JOIN  countries         AS co  FOR KEY c (country_id)
JOIN  customer_types    AS ct  FOR KEY c (customer_type_id)
JOIN  order_statuses     AS os  FOR KEY o (order_status_id)
LEFT JOIN discount_types AS dt  FOR KEY o (discount_type_id)
JOIN  shipping_methods   AS sm  FOR KEY o (shipping_method_id)
JOIN  payment_methods    AS pm  FOR KEY o (payment_method_id)
JOIN  currencies         AS cur FOR KEY o (currency_id)
JOIN  order_priorities   AS op  FOR KEY o (order_priority_id)
JOIN  tax_codes          AS tc  FOR KEY o (tax_code_id)
WHERE o.id = 100;
```

1. Do all key join arrows point right?

AND

2. Are all joins LEFT JOIN or [INNER] JOIN?



3. The first FROM item is cardinality-preserved in the join tree.

Key Joins Implementations



Swedish SQL database vendor.

Implemented the
Key Joins specification.

<https://www.mimer.com>



Patch under review.

We're looking for collaborators
to review and test the patch.

Browser demo via PGLite:

<https://keyjoin.org>

Questions?

Andreas Karlsson

`andreas@proxel.se`

Joel Jacobson

`joel@compiler.org`

`https://keyjoin.org`

Contents

DESIGN HISTORY

- A1 Syntax bikeshedding
- A2 New arrow syntax
- A3 FILTER (WHERE ...)
- A4 A puzzling objection
- A5 The questions predate our proposal

GUARANTEES

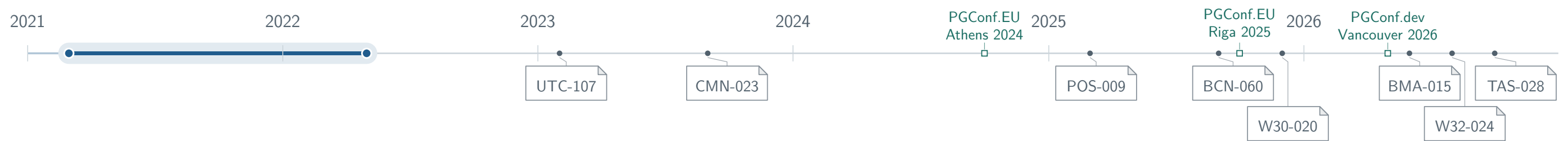
- A6–A7 [INNER] JOIN: FOR KEY and ON
- A8–A9 LEFT JOIN: FOR KEY and ON

FANOUT AND ITS CORRECTION

- A10 How the rows multiply
- A11 Aggregate before joining
- A12 The corrected query with FOR KEY

A CANONICAL FORM FOR KEY JOIN TREES

- A13 The larger query with ON
- A14 The larger query with FOR KEY
- A15–A25 Put the root table first
- A26 The eight FOR KEY forms



Appendix · [March 2021–May 2022](#) · [With Anders Granlund, the mailing list and Hacker News](#)

Syntax bikeshedding

MARCH 2021

```
JOIN departments AS d WITH employees_dept_id_fkey
```

DECEMBER 2021

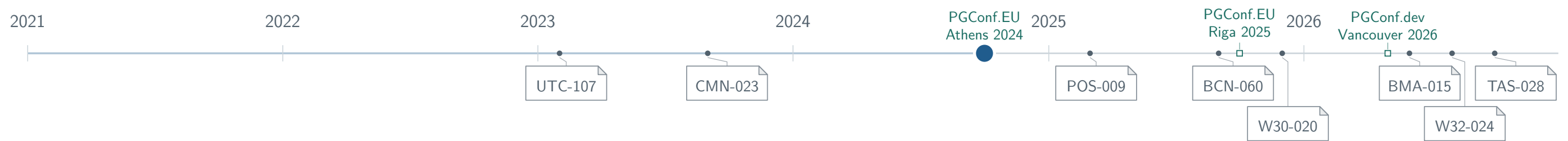
```
JOIN WITH e.employees_dept_id_fkey d
```

JANUARY 2022

```
JOIN departments AS d FOREIGN KEY (dept_id) FROM e
```

TWO COLUMN LISTS

```
JOIN departments AS d KEY (id) FROM e (dept_id)
```



Appendix · 6 October 2024 · Fewer competing meanings for FROM

New arrow syntax

BEFORE

```
KEY (id) FROM o (customer_id)
```

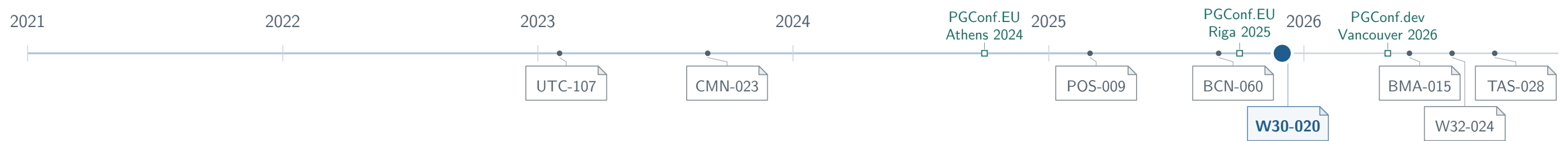
```
KEY (customer_id) TO c (id)
```

AFTER

```
KEY (id) <- o (customer_id)
```

```
KEY (customer_id) -> c (id)
```

The historical column-list order was still right operand first.



Appendix · December 2025 · A requested feature that we later removed

FILTER (WHERE ...)

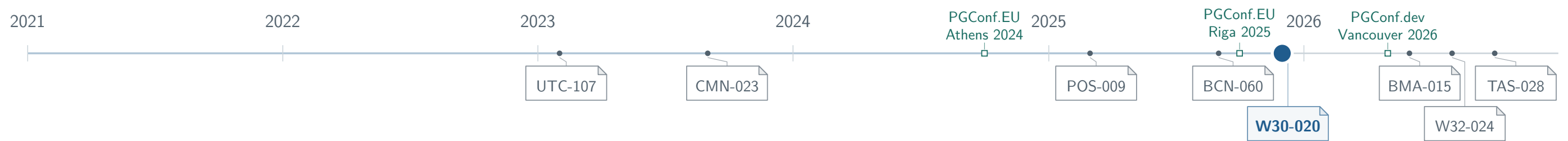
The idea: key joins with additional filtering conditions.

```
SELECT *  
FROM orders AS o  
LEFT JOIN customers AS c  
  FOR KEY (id) <- o (customer_id)  
  FILTER (WHERE c.active)
```

- Should filters apply before or after key join validation?
- Co-authors disagreed. We found no simple resolution.
- One reviewer strongly disliked all our syntax variants.

Omitted for simplicity.

We may revisit it if enough production queries demonstrate a need.



Appendix · December 2025 · Testing and compile-time proof

A puzzling objection

OBJECTION (PARAPHRASED)

Application testing should catch these mistakes; they do not justify new SQL syntax.

OUR RESPONSE

Buggy queries can pass tests on the chosen data.

FOR KEY requires a proof from the schema and query structure.

The questions predate our proposal

SQL STANDARD WORK

1986: schema links

1989: join shorthand objections

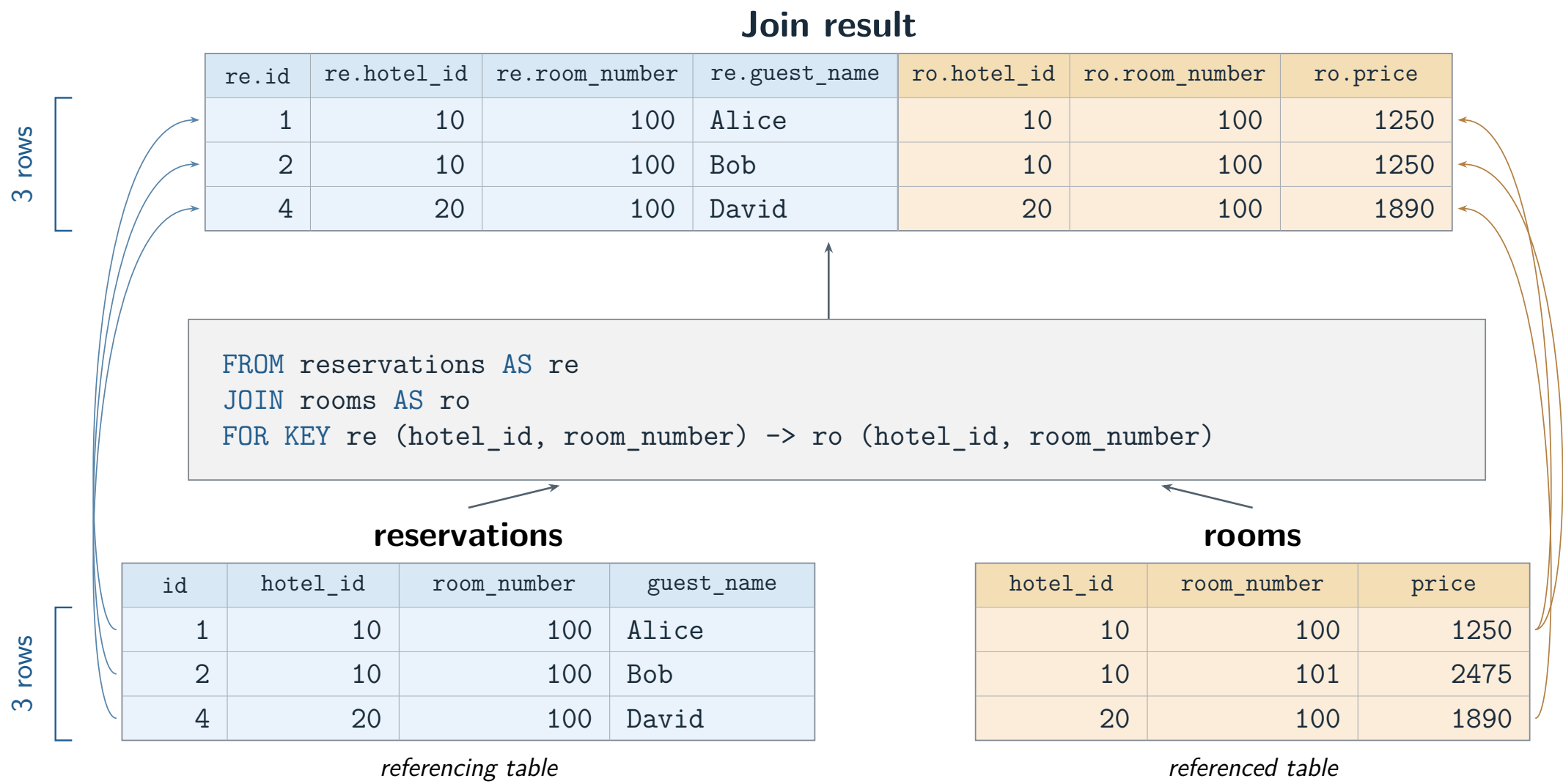
1990: added-FK ambiguity

SQL ANYWHERE

KEY JOIN infers
a relationship from
the available metadata.

Views, filters, ambiguity and dependencies
were hard questions then, too.

What FOR KEY guarantees: [INNER] JOIN

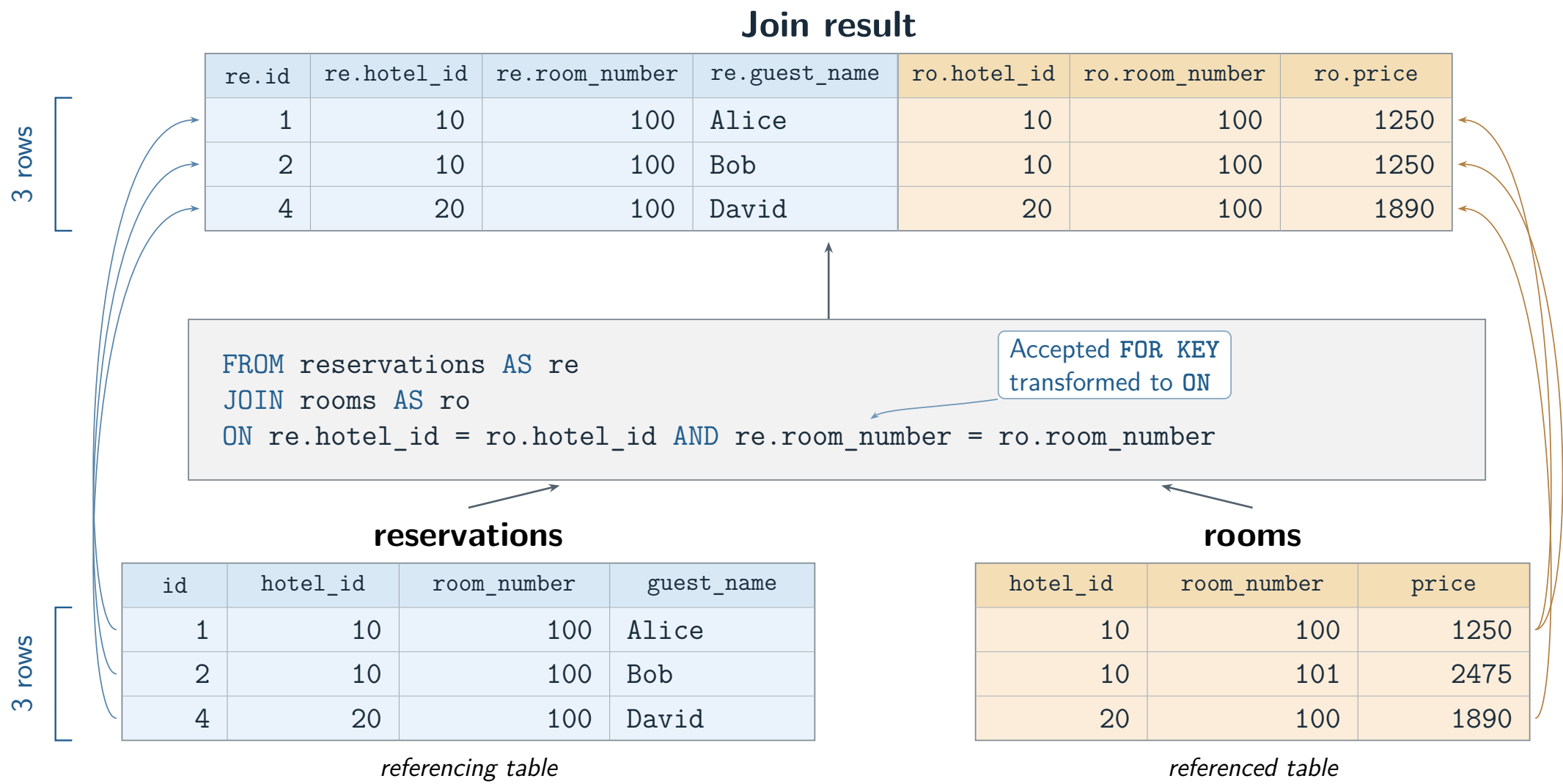


GUARANTEES

- 1 Every referencing row appears exactly once.
- 2 Rows with non-null values in all referencing columns find their unique matching referenced row.

FOR KEY requires the DBMS to prove these guarantees before execution.

What FOR KEY guarantees: [INNER] JOIN

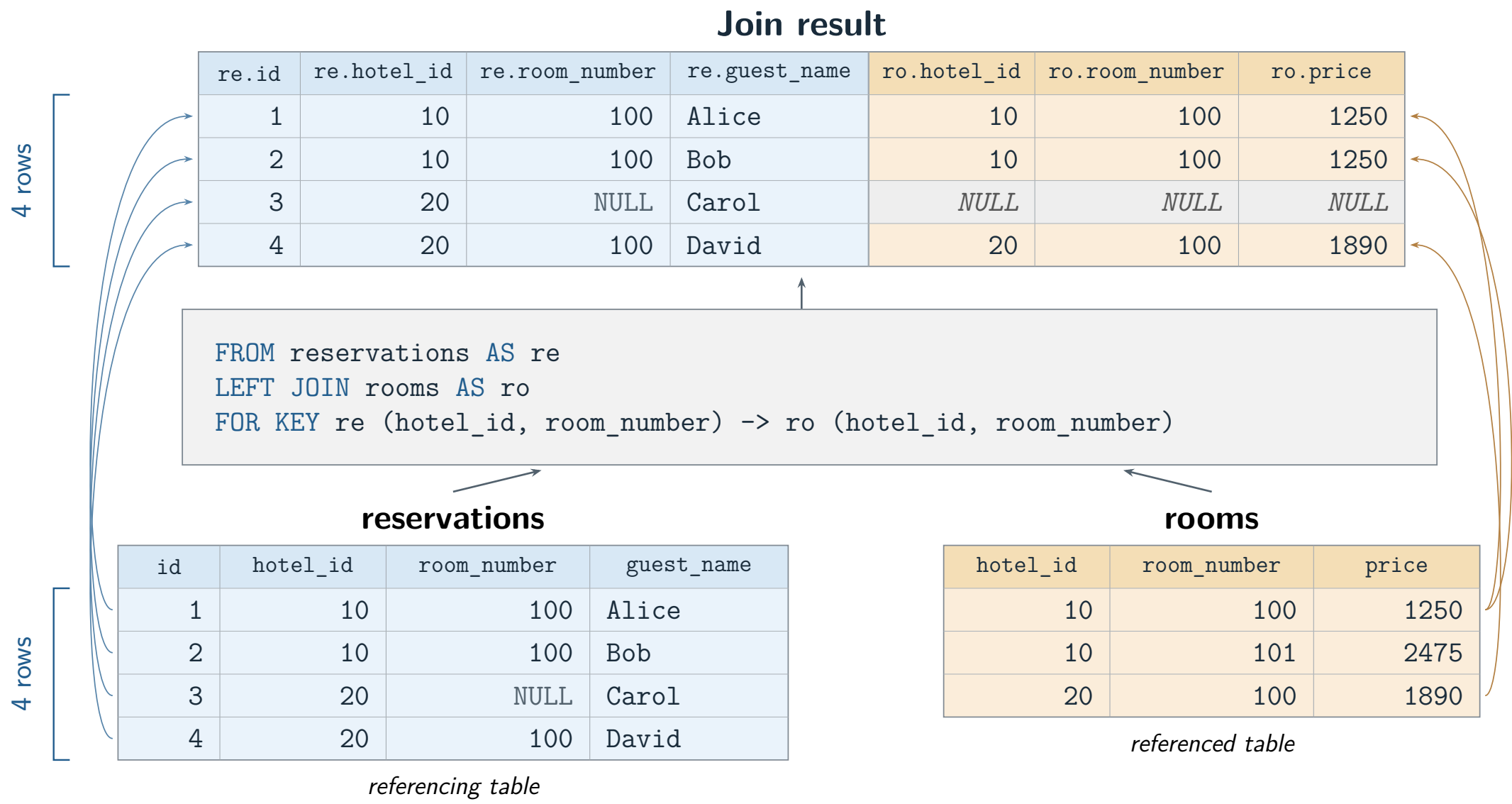


GUARANTEES

- 1 Every referencing row appears exactly once.
- 2 Rows with non-null values in all referencing columns find their unique matching referenced row.

FOR KEY requires the DBMS to prove these guarantees before execution.

What FOR KEY guarantees: LEFT JOIN

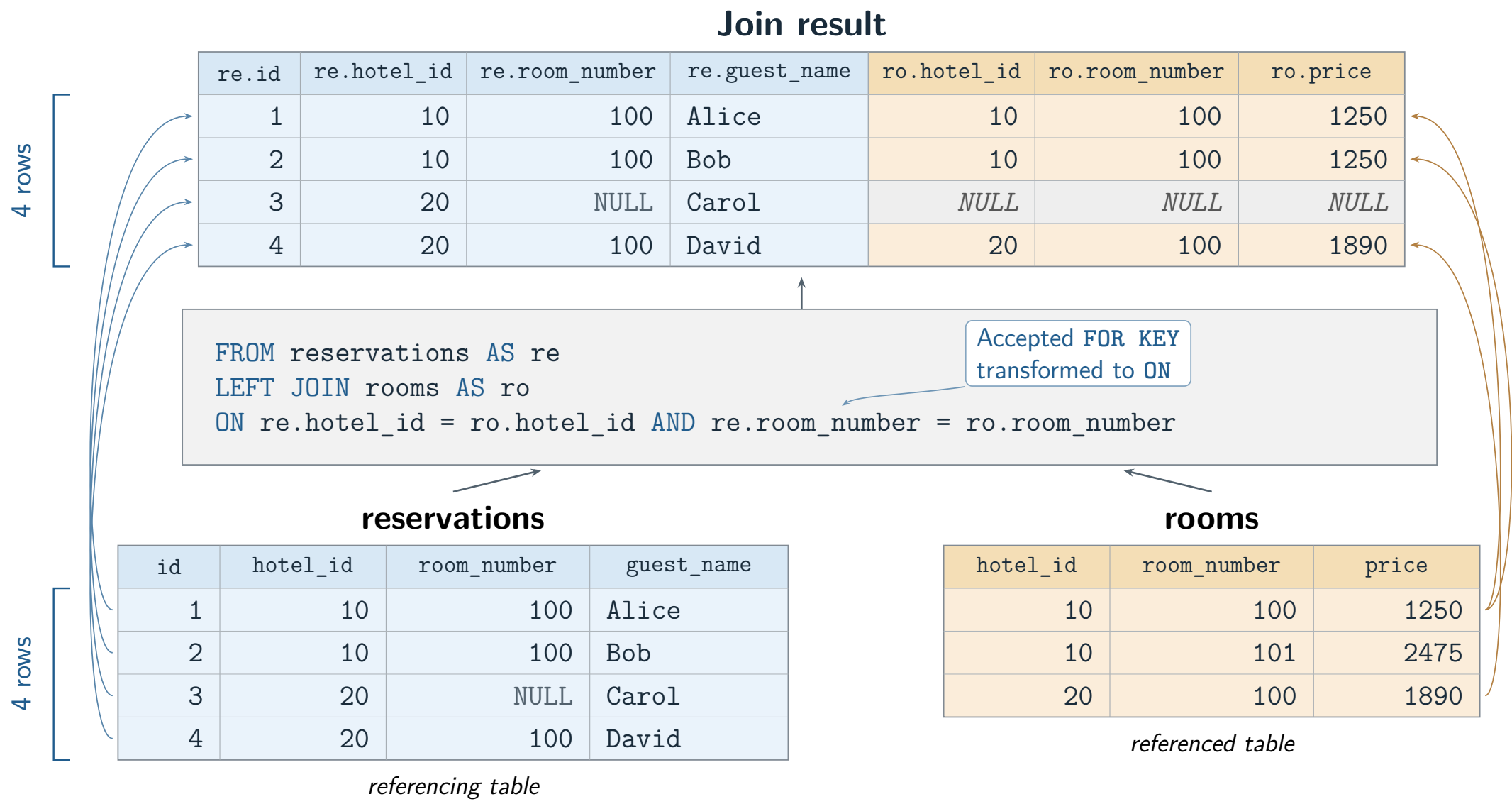


GUARANTEES

- 1 Every referencing row appears exactly once.
- 2 Rows with non-null values in all referencing columns find their unique matching referenced row.

FOR KEY requires the DBMS to prove these guarantees before execution.

What FOR KEY guarantees: LEFT JOIN



GUARANTEES

- 1 Every referencing row appears exactly once.
- 2 Rows with non-null values in all referencing columns find their unique matching referenced row.

FOR KEY requires the DBMS to prove these guarantees before execution.

How the rows multiply

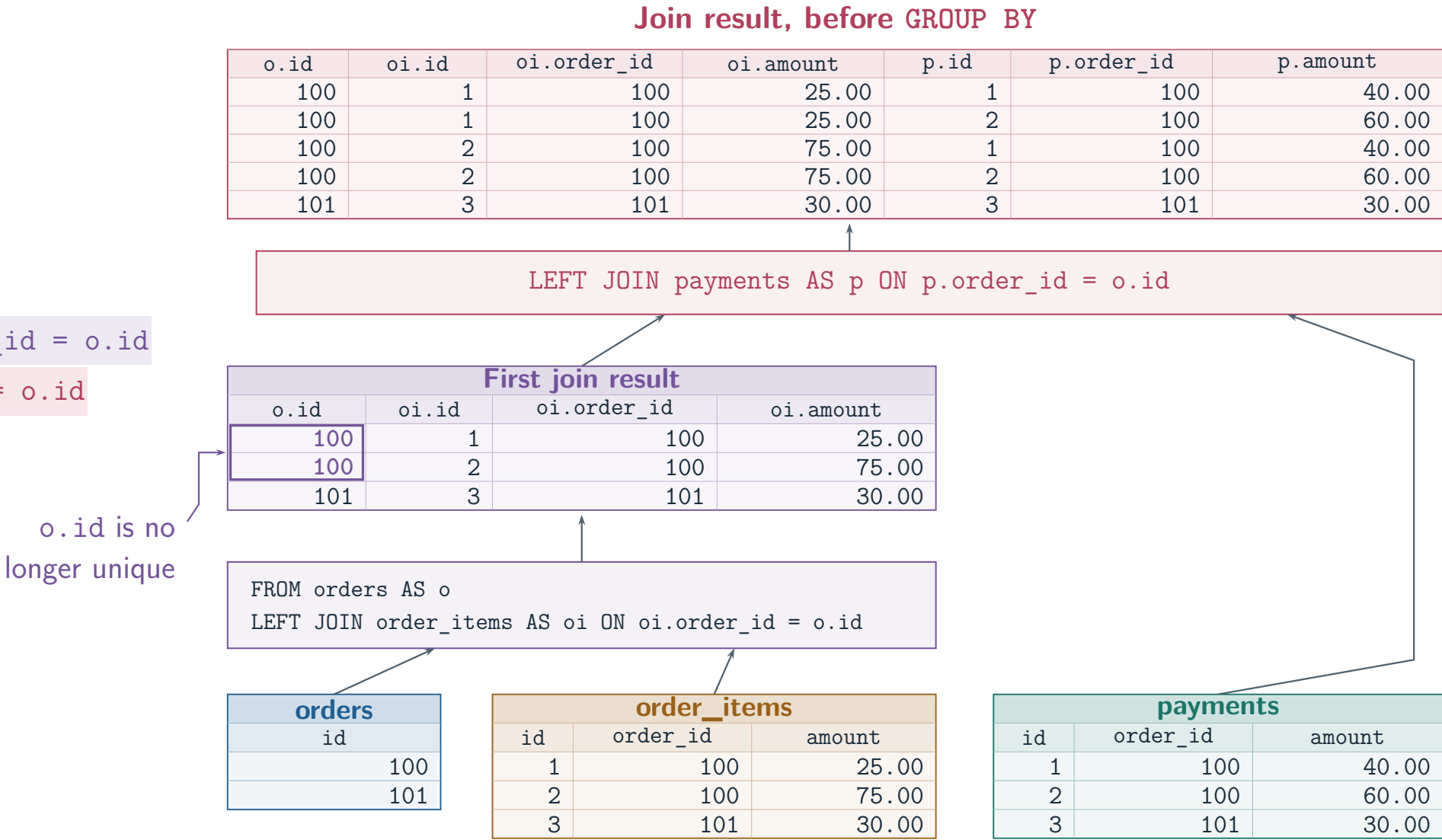
```
SELECT o.id,  
       SUM(oi.amount) AS item_total,  
       SUM(p.amount)  AS payment_total  
FROM orders AS o  
LEFT JOIN order_items AS oi ON oi.order_id = o.id  
LEFT JOIN payments AS p   ON p.order_id = o.id  
GROUP BY o.id;
```

After GROUP BY

id	item_total	payment_total
100	200.00	200.00
101	30.00	30.00

(2 rows)

Back to the story



Aggregate before joining

```
SELECT o.id,  
       oi.item_total,  
       p.payment_total  
FROM orders AS o  
LEFT JOIN (  
    SELECT order_id, SUM(amount) AS item_total  
    FROM order_items  
    GROUP BY order_id  
) AS oi ON o.id = oi.order_id  
LEFT JOIN (  
    SELECT order_id, SUM(amount) AS payment_total  
    FROM payments  
    GROUP BY order_id  
) AS p ON o.id = p.order_id;
```

Each subquery returns at most one row per order.

The corrected query with FOR KEY

```
SELECT o.id,  
       oi.item_total,  
       p.payment_total  
FROM orders AS o  
LEFT JOIN (  
    SELECT order_id, SUM(amount) AS item_total  
    FROM order_items  
    GROUP BY order_id  
) AS oi FOR KEY o (id) <- oi (order_id)  
LEFT JOIN (  
    SELECT order_id, SUM(amount) AS payment_total  
    FROM payments  
    GROUP BY order_id  
) AS p FOR KEY o (id) <- p (order_id);
```

The DBMS can now prove both key joins.

The larger query with ON

```

SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM customers          AS c
JOIN countries          AS co  ON co.id   = c.country_id
JOIN customer_types     AS ct  ON ct.id   = c.customer_type_id
JOIN orders             AS o    ON c.id    = o.customer_id
JOIN order_statuses     AS os  ON os.id   = o.order_status_id
LEFT JOIN discount_types AS dt  ON dt.id   = o.discount_type_id
JOIN shipping_methods   AS sm  ON sm.id   = o.shipping_method_id
JOIN payment_methods    AS pm  ON pm.id   = o.payment_method_id
JOIN order_items        AS oi  ON o.id    = oi.order_id
JOIN currencies         AS cur  ON cur.id  = o.currency_id
JOIN order_priorities   AS op  ON op.id   = o.order_priority_id
JOIN tax_codes          AS tc  ON tc.id   = o.tax_code_id
WHERE o.id = 100;

```

Each join needs a check against the schema and the preceding joins.

The larger query with FOR KEY

```
SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM customers AS c
JOIN countries AS co FOR KEY c (country_id) -> co (id)
JOIN customer_types AS ct FOR KEY c (customer_type_id) -> ct (id)
JOIN orders AS o FOR KEY c (id) <- o (customer_id)
JOIN order_statuses AS os FOR KEY o (order_status_id) -> os (id)
LEFT JOIN discount_types AS dt FOR KEY o (discount_type_id) -> dt (id)
JOIN shipping_methods AS sm FOR KEY o (shipping_method_id) -> sm (id)
JOIN payment_methods AS pm FOR KEY o (payment_method_id) -> pm (id)
JOIN order_items AS oi FOR KEY o (id) <- oi (order_id)
JOIN currencies AS cur FOR KEY o (currency_id) -> cur (id)
JOIN order_priorities AS op FOR KEY o (order_priority_id) -> op (id)
JOIN tax_codes AS tc FOR KEY o (tax_code_id) -> tc (id)
WHERE o.id = 100;
```

Put the root table first

```
SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM customers AS c
JOIN countries AS co FOR KEY c (country_id) -> co (id)
JOIN orders AS o FOR KEY c (id) <- o (customer_id)
JOIN customer_types AS ct FOR KEY c (customer_type_id) -> ct (id)
JOIN order_statuses AS os FOR KEY o (order_status_id) -> os (id)
LEFT JOIN discount_types AS dt FOR KEY o (discount_type_id) -> dt (id)
JOIN shipping_methods AS sm FOR KEY o (shipping_method_id) -> sm (id)
JOIN payment_methods AS pm FOR KEY o (payment_method_id) -> pm (id)
JOIN order_items AS oi FOR KEY o (id) <- oi (order_id)
JOIN currencies AS cur FOR KEY o (currency_id) -> cur (id)
JOIN order_priorities AS op FOR KEY o (order_priority_id) -> op (id)
JOIN tax_codes AS tc FOR KEY o (tax_code_id) -> tc (id)
WHERE o.id = 100;
```

Put the root table first

```
SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM customers AS c
JOIN orders AS o FOR KEY c (id) <- o (customer_id)
JOIN countries AS co FOR KEY c (country_id) -> co (id)
JOIN customer_types AS ct FOR KEY c (customer_type_id) -> ct (id)
JOIN order_statuses AS os FOR KEY o (order_status_id) -> os (id)
LEFT JOIN discount_types AS dt FOR KEY o (discount_type_id) -> dt (id)
JOIN shipping_methods AS sm FOR KEY o (shipping_method_id) -> sm (id)
JOIN payment_methods AS pm FOR KEY o (payment_method_id) -> pm (id)
JOIN order_items AS oi FOR KEY o (id) <- oi (order_id)
JOIN currencies AS cur FOR KEY o (currency_id) -> cur (id)
JOIN order_priorities AS op FOR KEY o (order_priority_id) -> op (id)
JOIN tax_codes AS tc FOR KEY o (tax_code_id) -> tc (id)
WHERE o.id = 100;
```

Put the root table first

```
SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM orders AS o
JOIN customers AS c FOR KEY o (customer_id) -> c (id)
JOIN countries AS co FOR KEY c (country_id) -> co (id)
JOIN customer_types AS ct FOR KEY c (customer_type_id) -> ct (id)
JOIN order_statuses AS os FOR KEY o (order_status_id) -> os (id)
LEFT JOIN discount_types AS dt FOR KEY o (discount_type_id) -> dt (id)
JOIN shipping_methods AS sm FOR KEY o (shipping_method_id) -> sm (id)
JOIN payment_methods AS pm FOR KEY o (payment_method_id) -> pm (id)
JOIN order_items AS oi FOR KEY o (id) <- oi (order_id)
JOIN currencies AS cur FOR KEY o (currency_id) -> cur (id)
JOIN order_priorities AS op FOR KEY o (order_priority_id) -> op (id)
JOIN tax_codes AS tc FOR KEY o (tax_code_id) -> tc (id)
WHERE o.id = 100;
```

Put the root table first

```
SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM orders AS o
JOIN customers AS c FOR KEY o (customer_id) -> c (id)
JOIN countries AS co FOR KEY c (country_id) -> co (id)
JOIN customer_types AS ct FOR KEY c (customer_type_id) -> ct (id)
JOIN order_statuses AS os FOR KEY o (order_status_id) -> os (id)
LEFT JOIN discount_types AS dt FOR KEY o (discount_type_id) -> dt (id)
JOIN shipping_methods AS sm FOR KEY o (shipping_method_id) -> sm (id)
JOIN order_items AS oi FOR KEY o (id) <- oi (order_id)
JOIN payment_methods AS pm FOR KEY o (payment_method_id) -> pm (id)
JOIN currencies AS cur FOR KEY o (currency_id) -> cur (id)
JOIN order_priorities AS op FOR KEY o (order_priority_id) -> op (id)
JOIN tax_codes AS tc FOR KEY o (tax_code_id) -> tc (id)
WHERE o.id = 100;
```

Put the root table first

```
SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM orders AS o
JOIN customers AS c FOR KEY o (customer_id) -> c (id)
JOIN countries AS co FOR KEY c (country_id) -> co (id)
JOIN customer_types AS ct FOR KEY c (customer_type_id) -> ct (id)
JOIN order_statuses AS os FOR KEY o (order_status_id) -> os (id)
LEFT JOIN discount_types AS dt FOR KEY o (discount_type_id) -> dt (id)
JOIN order_items AS oi FOR KEY o (id) <- oi (order_id)
JOIN shipping_methods AS sm FOR KEY o (shipping_method_id) -> sm (id)
JOIN payment_methods AS pm FOR KEY o (payment_method_id) -> pm (id)
JOIN currencies AS cur FOR KEY o (currency_id) -> cur (id)
JOIN order_priorities AS op FOR KEY o (order_priority_id) -> op (id)
JOIN tax_codes AS tc FOR KEY o (tax_code_id) -> tc (id)
WHERE o.id = 100;
```

Put the root table first

```
SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM orders AS o
JOIN customers AS c FOR KEY o (customer_id) -> c (id)
JOIN countries AS co FOR KEY c (country_id) -> co (id)
JOIN customer_types AS ct FOR KEY c (customer_type_id) -> ct (id)
JOIN order_statuses AS os FOR KEY o (order_status_id) -> os (id)
JOIN order_items AS oi FOR KEY o (id) <- oi (order_id)
LEFT JOIN discount_types AS dt FOR KEY o (discount_type_id) -> dt (id)
JOIN shipping_methods AS sm FOR KEY o (shipping_method_id) -> sm (id)
JOIN payment_methods AS pm FOR KEY o (payment_method_id) -> pm (id)
JOIN currencies AS cur FOR KEY o (currency_id) -> cur (id)
JOIN order_priorities AS op FOR KEY o (order_priority_id) -> op (id)
JOIN tax_codes AS tc FOR KEY o (tax_code_id) -> tc (id)
WHERE o.id = 100;
```

Put the root table first

```
SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM orders AS o
JOIN customers AS c FOR KEY o (customer_id) -> c (id)
JOIN countries AS co FOR KEY c (country_id) -> co (id)
JOIN customer_types AS ct FOR KEY c (customer_type_id) -> ct (id)
JOIN order_items AS oi FOR KEY o (id) <- oi (order_id)
JOIN order_statuses AS os FOR KEY o (order_status_id) -> os (id)
LEFT JOIN discount_types AS dt FOR KEY o (discount_type_id) -> dt (id)
JOIN shipping_methods AS sm FOR KEY o (shipping_method_id) -> sm (id)
JOIN payment_methods AS pm FOR KEY o (payment_method_id) -> pm (id)
JOIN currencies AS cur FOR KEY o (currency_id) -> cur (id)
JOIN order_priorities AS op FOR KEY o (order_priority_id) -> op (id)
JOIN tax_codes AS tc FOR KEY o (tax_code_id) -> tc (id)
WHERE o.id = 100;
```

Put the root table first

```
SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM orders AS o
JOIN customers AS c FOR KEY o (customer_id) -> c (id)
JOIN countries AS co FOR KEY c (country_id) -> co (id)
JOIN order_items AS oi FOR KEY o (id) <- oi (order_id)
JOIN customer_types AS ct FOR KEY c (customer_type_id) -> ct (id)
JOIN order_statuses AS os FOR KEY o (order_status_id) -> os (id)
LEFT JOIN discount_types AS dt FOR KEY o (discount_type_id) -> dt (id)
JOIN shipping_methods AS sm FOR KEY o (shipping_method_id) -> sm (id)
JOIN payment_methods AS pm FOR KEY o (payment_method_id) -> pm (id)
JOIN currencies AS cur FOR KEY o (currency_id) -> cur (id)
JOIN order_priorities AS op FOR KEY o (order_priority_id) -> op (id)
JOIN tax_codes AS tc FOR KEY o (tax_code_id) -> tc (id)
WHERE o.id = 100;
```

Put the root table first

```
SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM orders AS o
JOIN customers AS c FOR KEY o (customer_id) -> c (id)
JOIN order_items AS oi FOR KEY o (id) <- oi (order_id)
JOIN countries AS co FOR KEY c (country_id) -> co (id)
JOIN customer_types AS ct FOR KEY c (customer_type_id) -> ct (id)
JOIN order_statuses AS os FOR KEY o (order_status_id) -> os (id)
LEFT JOIN discount_types AS dt FOR KEY o (discount_type_id) -> dt (id)
JOIN shipping_methods AS sm FOR KEY o (shipping_method_id) -> sm (id)
JOIN payment_methods AS pm FOR KEY o (payment_method_id) -> pm (id)
JOIN currencies AS cur FOR KEY o (currency_id) -> cur (id)
JOIN order_priorities AS op FOR KEY o (order_priority_id) -> op (id)
JOIN tax_codes AS tc FOR KEY o (tax_code_id) -> tc (id)
WHERE o.id = 100;
```

Put the root table first

```
SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM orders AS o
JOIN order_items AS oi FOR KEY o (id)          <- oi (order_id)
JOIN customers AS c FOR KEY o (customer_id)    -> c (id)
JOIN countries AS co FOR KEY c (country_id)    -> co (id)
JOIN customer_types AS ct FOR KEY c (customer_type_id) -> ct (id)
JOIN order_statuses AS os FOR KEY o (order_status_id) -> os (id)
LEFT JOIN discount_types AS dt FOR KEY o (discount_type_id) -> dt (id)
JOIN shipping_methods AS sm FOR KEY o (shipping_method_id) -> sm (id)
JOIN payment_methods AS pm FOR KEY o (payment_method_id) -> pm (id)
JOIN currencies AS cur FOR KEY o (currency_id) -> cur (id)
JOIN order_priorities AS op FOR KEY o (order_priority_id) -> op (id)
JOIN tax_codes AS tc FOR KEY o (tax_code_id) -> tc (id)
WHERE o.id = 100;
```

Put the root table first

```
SELECT o.id, o.order_date, oi.amount,
       c.name AS customer, co.country_name,
       os.status_name, sm.method_name AS shipping,
       pm.method_name AS payment, cur.currency_code,
       op.priority_name, tc.code_name AS tax_code,
       dt.type_name AS discount
FROM order_items      AS oi
JOIN orders            AS o   FOR KEY oi (order_id)      -> o (id)
JOIN customers         AS c   FOR KEY o (customer_id)     -> c (id)
JOIN countries         AS co  FOR KEY c (country_id)      -> co (id)
JOIN customer_types    AS ct  FOR KEY c (customer_type_id) -> ct (id)
JOIN order_statuses    AS os  FOR KEY o (order_status_id) -> os (id)
LEFT JOIN discount_types AS dt FOR KEY o (discount_type_id) -> dt (id)
JOIN shipping_methods  AS sm  FOR KEY o (shipping_method_id) -> sm (id)
JOIN payment_methods   AS pm  FOR KEY o (payment_method_id) -> pm (id)
JOIN currencies        AS cur FOR KEY o (currency_id)      -> cur (id)
JOIN order_priorities  AS op  FOR KEY o (order_priority_id) -> op (id)
JOIN tax_codes         AS tc  FOR KEY o (tax_code_id)      -> tc (id)
WHERE o.id = 100;
```

Which key joins preserve the referencing table's cardinality?

Two arrow directions × four join types = eight **FOR KEY** forms

