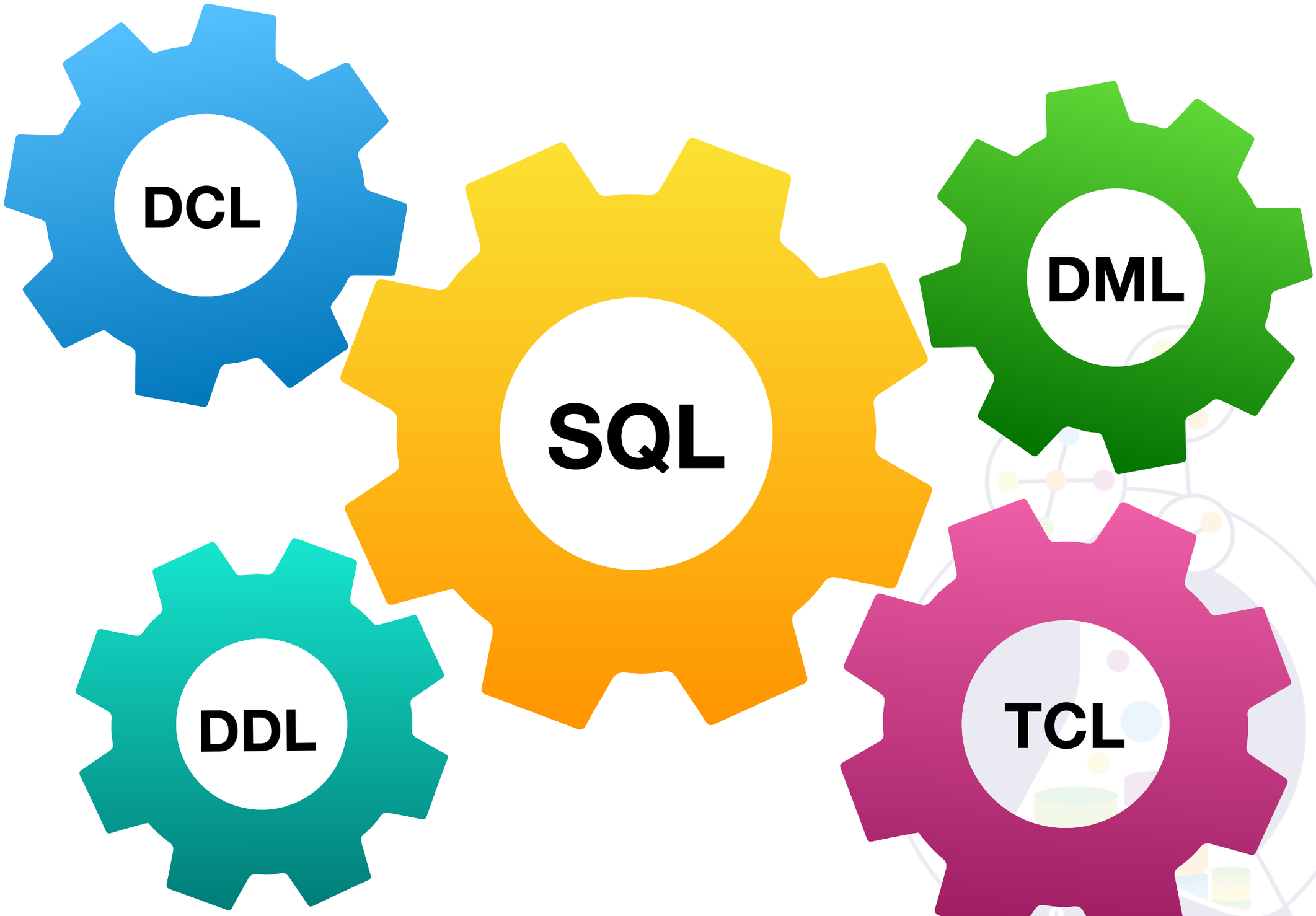
datalab

data is everywhere, value is hidden

Relational Databases

Lecturer: Азат Якупов (Azat Yakupov)
https://datalaboratory.one



1986
SQL-86

1989
SQL-89

1992
SQL-92

1999
SQL:1999

2003
SQL:2003

2006
SQL:2006

2008
SQL:2008

2011
SQL:2011

2016
SQL:2016

2020
SQL:2020+

DATE, TIME, TIMESTAMP, INTERVAL, BIT, VARCHAR, NCHAR

UNION JOIN, NATURAL JOIN, DIFFERENCE, INTERSECTION

CASE, CAST

ALTER , DROP

CHECK constraints

Database cursors

Transaction Isolation Levels, Dynamic SQL, Temporary Tables

information_schema is a metadata layer

Recursive Queries

Database triggers

Arrays

BOOLEAN data type

Common Table Expressions (CTE)

ROLLUP, CUBE, GROUPING SETS

CREATE ROLE

UNNEST

XML-related features (SQL/XML)

Window functions

Sequence generator

auto-generated columns

MERGE

CREATE TABLE AS …

CREATE TABLE LIKE …

BIT and BIT VARYING data types

OLAP extended with window functions

XML-related features (SQL/XML)

XQuery

XML manipulation in database

MERGE and DIAGNOSTIC statements

TRUNCATE TABLE

WHEN clauses in a CASE expression

INSTEAD OF database triggers

partitioned JOIN tables

FETCH clause

enhanced XQuery

enhancements to derived column names

chronological databases (~ temporal databases)

PERIOD FOR

enhancements for window functions and FETCH clause

JSON support

regular expressions

Date and time formatting and parsing

LISTAGG

Data type DECFLOAT

Polymorphic table functions

Multi-Dimensional Arrays

SQL / GraphQL

DML

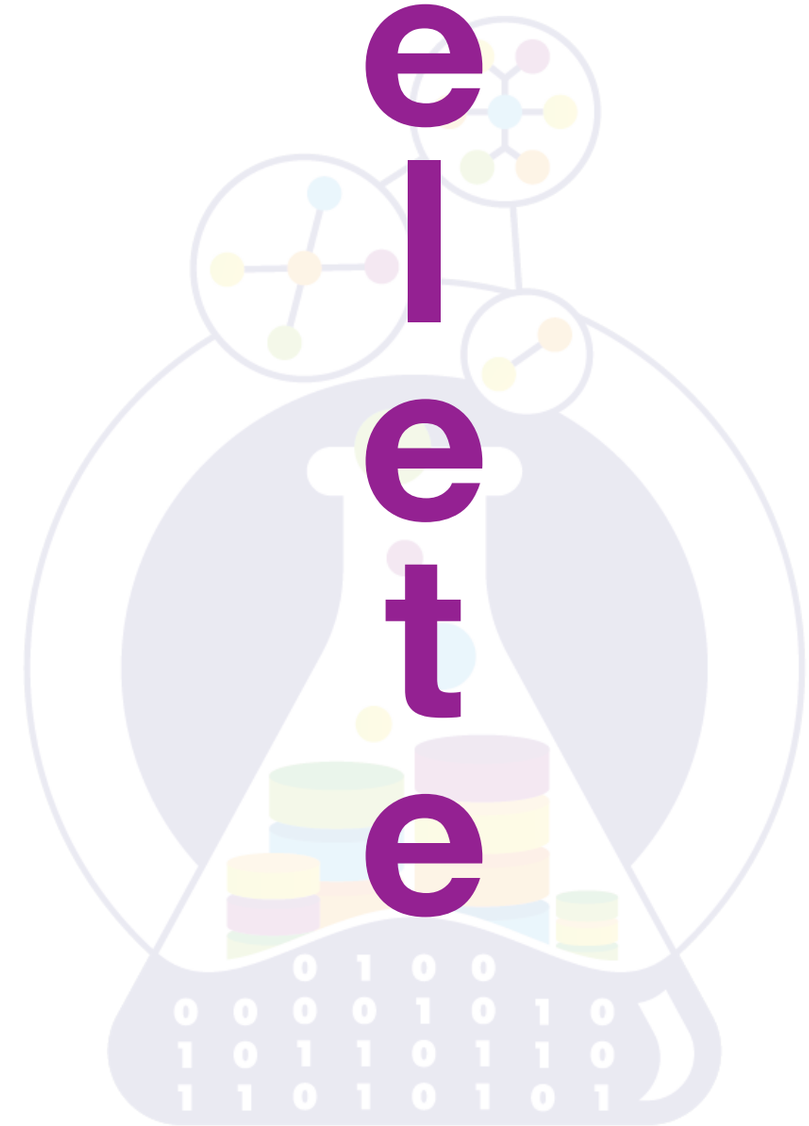
CRUD

Create

Read

Update

Delete



C

INSERT

R

SELECT

U

UPDATE

D

DELETE

```
INSERT INTO Student
  (StudentID, FirstName, SecondName,
   LastName, Address, Phone)
VALUES (1, ‘Peter’, ‘Petrov’, ‘Petrovich’,
        ‘Kazan’, ‘5-44-9247’);
```

```
INSERT INTO Student
  (StudentID, FirstName, SecondName,
  LastName, Address, Phone)
VALUES (DEFAULT, ‘Peter’, ‘Petrov’,
‘Petrovich’, ‘Kazan’, ‘5-44-9247’);
```

```
INSERT INTO Student
DEFAULT VALUES;
```

```
INSERT INTO Student
  (StudentID, FirstName, SecondName,
   LastName, Address, Phone)
VALUES (1, ‘Peter’, ‘Petrov’, ‘Petrovich’,
        ‘Kazan’, ‘5-44-9247’),
       (2, ‘Ivan’, ‘Ivanov’, ‘Ivanovich’,
        ‘Moscow’, ‘3-14-7381’) ;
```

```
CREATE TABLE s_kazan
AS
SELECT *
  FROM Student
WHERE Address = ‘Kazan’;
```

```
INSERT INTO s_kazan
SELECT *
FROM Student
WHERE Address = ‘Kazan’;
```

```
INSERT INTO Student
  (StudentID, FirstName, SecondName,
   LastName, Address, Phone)
VALUES (1, ‘Peter’, ‘Petrov’, ‘Petrovich’,
        ‘Kazan’, ‘5-44-9247’);
```

```
INSERT ALL
  INTO Student
    (StudentID, FirstName, SecondName,
     LastName, Address, Phone)
  VALUES (1, ‘Peter’, ‘Petrov’, ‘Petrovich’,
                ‘Kazan’, ‘5-44-9247’)
  INTO Student
    (StudentID, FirstName, SecondName,
     LastName, Address, Phone)
  VALUES(2, ‘Ivan’, ‘Ivanov’, ‘Ivanovich’,
                ‘Moscow’, ‘3-14-7381’)
  INTO Fun (FanId, Name)
  VALUES(1, ‘Party’)
SELECT * FROM DUAL;
```

```
CREATE TABLE s_kazan
AS
SELECT *
  FROM Student
WHERE Address = ‘Kazan’;
```

```
INSERT INTO s_kazan
SELECT *
FROM Student
WHERE Address = ‘Kazan’;
```

```
INSERT INTO Student
  (StudentID, FirstName, SecondName,
   LastName, Address, Phone)
VALUES (1, ‘Peter’, ‘Petrov’, ‘Petrovich’,
        ‘Kazan’, ‘5-44-9247’);
```

```
CREATE TABLE s_kazan
AS
SELECT *
  FROM Student
WHERE Address = ‘Kazan’;
```

```
INSERT INTO s_kazan
SELECT *
FROM Student
WHERE Address = ‘Kazan’;
```

```
INSERT INTO Student
  (StudentID, FirstName, SecondName,
   LastName, Address, Phone)
VALUES (1, ‘Peter’, ‘Petrov’, ‘Petrovich’,
          ‘Kazan’, ‘5-44-9247’),
         (2, ‘Ivan’, ‘Ivanov’, ‘Ivanovich’,
          ‘Moscow’, ‘3-14-7381’) ;
```

```
UPDATE Student
  SET Address = 'Kazan',
      Phone = DEFAULT
WHERE StudentId = 1;
```

```
WITH cte_upd AS
  (UPDATE Student
        SET Address = 'Kazan'
    WHERE StudentID > 1000
     RETURNING *)
INSERT INTO Student_log
  SELECT *, current_timestamp
    FROM cte_upd;
```

```
UPDATE Student
SET Address = ‘Kazan’
WHERE StudentId = 1;
```

```
UPDATE Student
SET Address = ‘Kazan’
WHERE StudentId = 1;
```

```
UPDATE Student
SET StudentId = StudentId + 1
ORDER BY StudentId DESC;
```

```
UPDATE Student, Addresses
SET Student.Address =
        Addresses.Address
WHERE Student.ID =
        Addresses.Student_ID
```

```
UPDATE Student
   SET Phone = CONCAT('555', Phone)
LIMIT 100;
```

```
DELETE FROM Student;
```

```
DELETE FROM Student
WHERE StudentId = 1;
```

```
DELETE FROM Student
WHERE StudentId IN (1,2,3);
```

```
DELETE FROM Student;
```

```
DELETE FROM Student
WHERE StudentId = 1;
```

```
DELETE FROM Student
WHERE StudentId IN (1,2,3);
```

```
DELETE FROM Student;
```

```
DELETE FROM Student
WHERE StudentId = 1;
```

```
DELETE FROM Student
WHERE StudentId IN (1,2,3);
```

TRUNCATE Student

```
INSERT INTO Student
  (StudentID, FirstName, SecondName,
   LastName, Address, Phone)
VALUES (1, ‘Peter’, ‘Petrov’, ‘Petrovich’,
        ‘Kazan’, ‘5-44-9247’)
ON CONFLICT (StudentID) DO NOTHING;
```

```
INSERT INTO Student
  (StudentID, FirstName, SecondName,
   LastName, Address, Phone)
VALUES (1, ‘Peter’, ‘Petrov’, ‘Petrovich’,
                ‘Kazan’, ‘5-44-9247’)
ON CONFLICT (StudentID)
DO UPDATE SET
   Address =  EXCLUDED.Address,
                        Phone = ‘123’;
```

```
INSERT INTO Student
  (StudentID, FirstName, SecondName,
   LastName, Address, Phone)
VALUES (1, ‘Peter’, ‘Petrov’, ‘Petrovich’,
        ‘Kazan’, ‘5-44-9247’)
ON CONFLICT ON CONSTRAINT
 student_pk DO NOTHING;
```

```
MERGE INTO Student
USING dual ON (StudentID = 1)
WHEN MATCHED THEN
  UPDATE SET Address = 'Moscow'
WHEN NOT MATCHED THEN
  INSERT (StudentID, FirstName,
SecondName,
   LastName, Address, Phone)
  VALUES (1, 'Peter', 'Petrov', 'Petrovich',
              'Kazan', '5-44-9247');
```

```
MERGE INTO StudentKazan sk
USING (SELECT *
            FROM Student
           WHERE Address = 'Kazan') s
ON (sk.StudentID = s.StudentID)
WHEN MATCHED THEN
  UPDATE SET sk.Address = s.Address
WHEN NOT MATCHED THEN
  INSERT (StudentID, FirstName, SecondName,
   LastName, Address, Phone)
  VALUES (s.StudentID, s.FirstName,
s.SecondName,
   s.LastName, s.Address, s.Phone);
```

```
INSERT INTO Student
  (StudentID, FirstName, SecondName,
   LastName, Address, Phone)
VALUES (1, ‘Peter’, ‘Petrov’, ‘Petrovich’,
        ‘Kazan’, ‘5-44-9247’)
ON DUPLICATE KEY
UPDATE FirstName = ‘Peter’,
       SecondName = ‘Petrov’,
       LastName = ‘Petrovich’,
       Address = ‘Kazan’
       Phone = ‘5-44-9247’;
```

REPLACE INTO Student
VALUES (1, ‘Peter’, ‘Petrov’, ‘Petrovich’, ‘Kazan’, ‘5-44-9247’);

SELECT ...

FROM ...

WHERE ...

GROUP BY ...

HAVING ...

ORDER BY ...

```
SELECT *
  FROM document
  WHERE short_content = ‘тест’
  LIMIT 100;
```

```
SELECT *
  FROM document
 WHERE deleted = 1
 LIMIT 100;
```

```
SELECT *
  FROM document
WHERE short_content = ‘тест’
            AND deleted = 1
LIMIT 100;
```

```
SELECT *
  FROM document
  WHERE short_content = ‘тест’
  OR short_content = ‘Для теста’
  LIMIT 100;
```

```
SELECT *
  FROM document
WHERE (short_content = ‘тест’
        AND deleted = 1)
       OR
       (short_content = ‘Для теста’)
LIMIT 100;
```

```
SELECT *
  FROM document
WHERE delivery_type = 12 OR
                delivery_type = 21 OR
                delivery_type = 36
LIMIT 100;
```

```
SELECT *
  FROM document
WHERE delivery_type IN (12, 21, 36)
LIMIT 100;
```

```
SELECT *
  FROM document
WHERE delivery_type
                BETWEEN 12 AND 36
LIMIT 100;
```

```
SELECT *
  FROM document
  WHERE cdate <= now()
  LIMIT 100;
```

```
SELECT *
  FROM document
  WHERE cdate <= ‘2020-01-20’
  LIMIT 100;
```

```
SELECT *
  FROM document
WHERE cdate BETWEEN
 '2020-01-20'  AND '2021-01-20'
LIMIT 100;
```

```
SELECT *
  FROM document
WHERE cdate BETWEEN '2020-01-20'
                AND '2021-01-20'
ORDER BY cdate
LIMIT 100;
```

```
SELECT *
  FROM document
WHERE cdate BETWEEN '2020-01-20'
                AND '2021-01-20'
ORDER BY cdate DESC
LIMIT 100;
```

```
SELECT *
  FROM document
 WHERE short_content IS NULL;
```

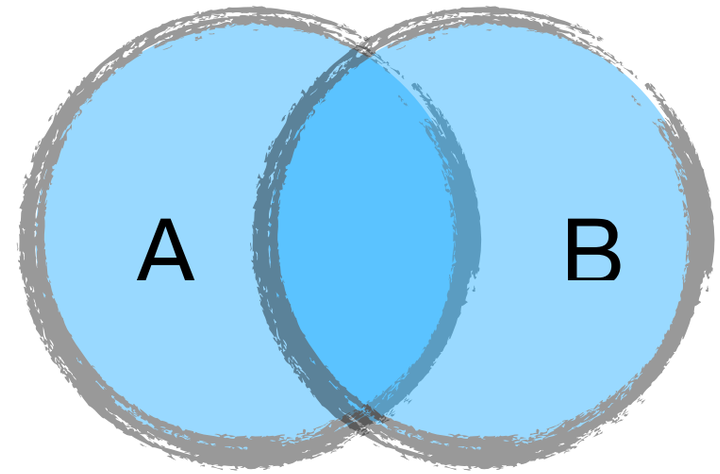
```
SELECT *
  FROM document
 WHERE short_content IS NOT NULL;
```

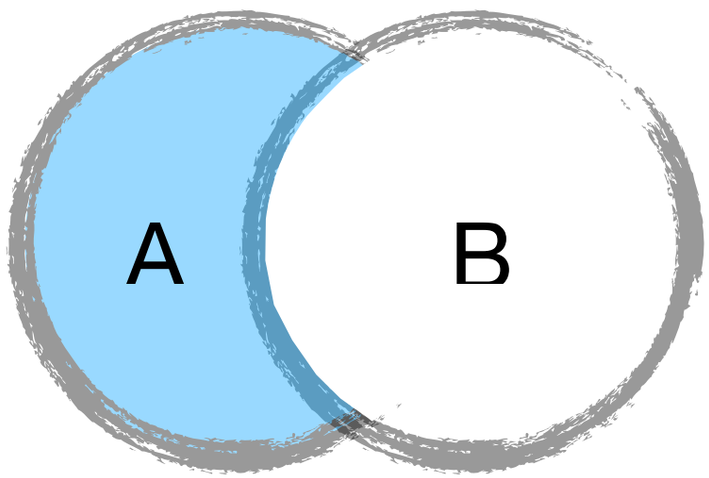
```
SELECT *
  FROM document
WHERE cdate BETWEEN '2020-01-20'
                AND '2021-01-20'
ORDER BY cdate
DESC NULLS FIRST
LIMIT 100;
```

```
SELECT *
  FROM document
WHERE cdate BETWEEN '2020-01-20'
                AND '2021-01-20'
ORDER BY cdate
DESC NULLS LAST
LIMIT 100;
```

UNION

UNION ALL



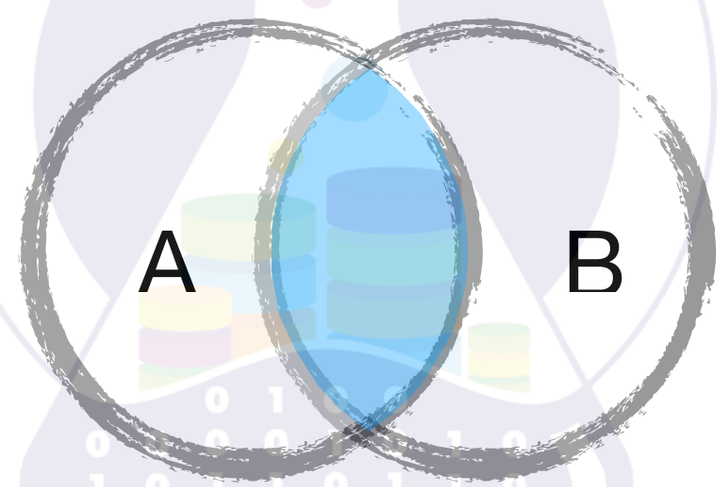


EXCEPT / MINUS

EXCEPT ALL

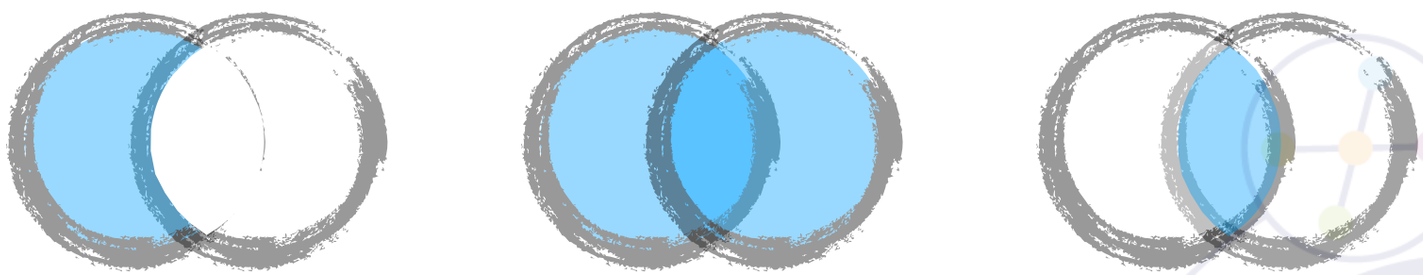
INTERSECT

INTERSECT ALL



A

```
(SELECT FirstName AS name,
        LastName AS desc
 FROM Student)
```



B

```
(SELECT FunName,
        Description
 FROM Fun);
```

```
SELECT id
  FROM og_author_type
UNION
SELECT id
  FROM og_result_ref
```

```
SELECT id
  FROM og_author_type
UNION ALL
SELECT id
  FROM og_result_ref
```

```
SELECT id, name
  FROM og_author_type
UNION
SELECT id, new_name
  FROM og_result_ref
```

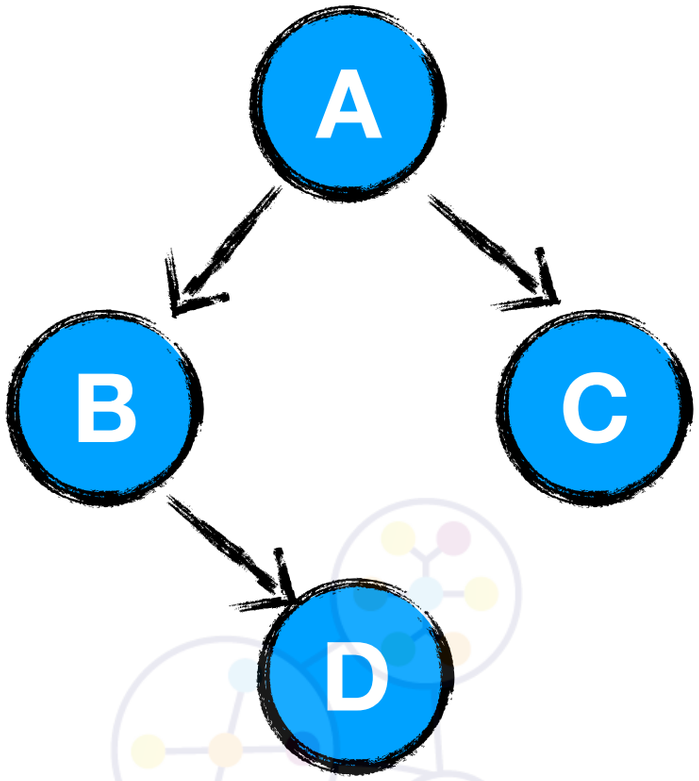
```
SELECT id, name
  FROM og_author_type
EXCEPT
SELECT id, new_name
  FROM og_result_ref
```

```
SELECT id, name
  FROM og_author_type
INTERSECT
SELECT id, new_name
  FROM og_result_ref
```

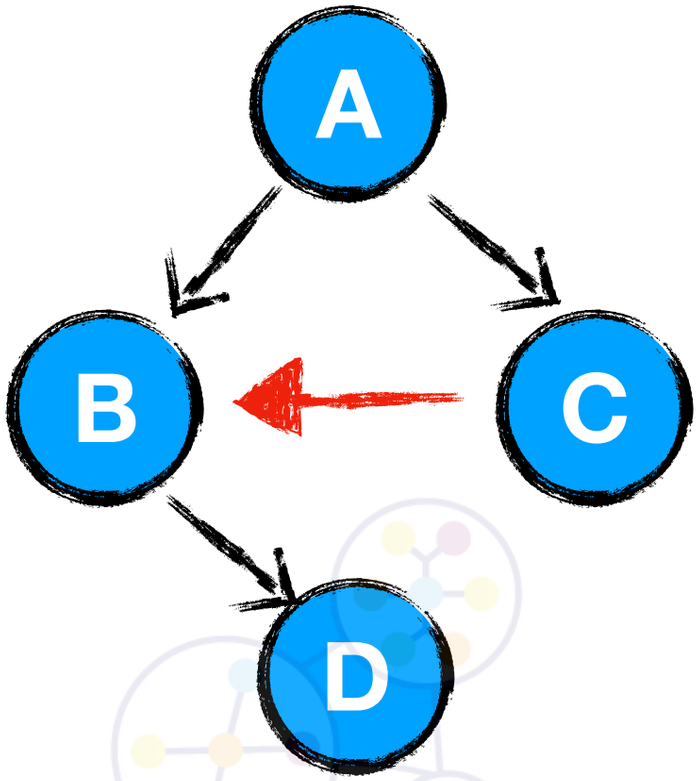
```
WITH _t1 AS (
    SELECT ...
      FROM ...
     WHERE ... ) ,
        _t2 AS (
    SELECT ...
      FROM ...
     WHERE ...)
SELECT ...
FROM A INNER JOIN _t2 ON ...
            LEFT JOIN  _t1 ON ...
```

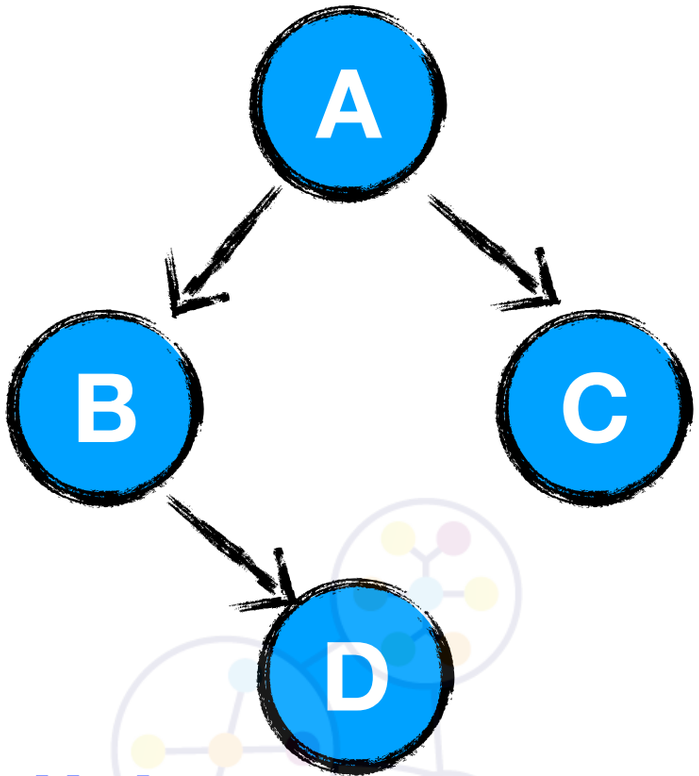
```sql
WITH regional_sales AS (
        SELECT region, SUM(amount) AS total_sales
          FROM orders
    GROUP BY region),
         top_regions AS (
        SELECT region
          FROM regional_sales
        WHERE total_sales > (SELECT SUM(total_sales)/10
                                FROM regional_sales) )
      SELECT region,
                product,
                SUM(quantity) AS product_units,
                SUM(amount) AS product_sales
        FROM orders
      WHERE region IN (SELECT region FROM top_regions)
  GROUP BY region, product;
```

```
WITH RECURSIVE _t AS
  (SELECT num, 1 AS level
    FROM t
   WHERE par_id IS NULL
   UNION
   SELECT t.num, _t.level + 1 AS level
    FROM t INNER JOIN _t ON (_t.num = t.par_id) )
 SELECT num, level
   FROM _t
```



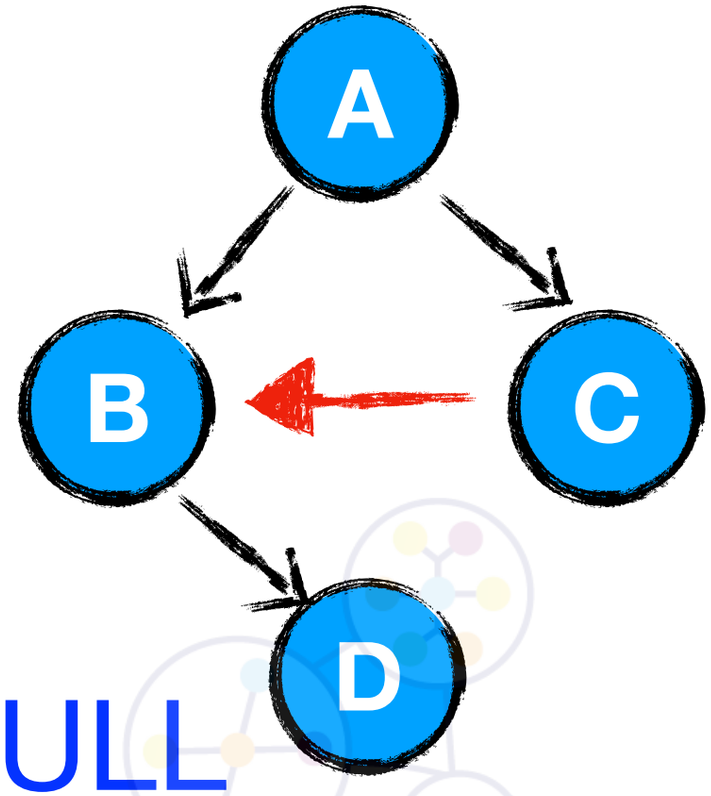
```
WITH RECURSIVE _t AS
 (SELECT num,
          array[num] AS path,
          FALSE AS cycle
   FROM t
  WHERE par_id IS NULL
  UNION ALL
  SELECT t.num,
          _t.path || t.num AS path,
          t.num = ANY (_t.path) AS cycle
   FROM t INNER JOIN _t ON (_t.num = t.par_id)
                           AND NOT cycle)
SELECT num, path
  FROM _t
```






```
SELECT num, LEVEL
   FROM t
START WITH par_id IS NULL
CONNECT BY PRIOR num = par_id
```

```
SELECT num, LEVEL
   FROM t
START WITH par_id IS NULL
CONNECT BY NOCYCLE
PRIOR num = par_id
```



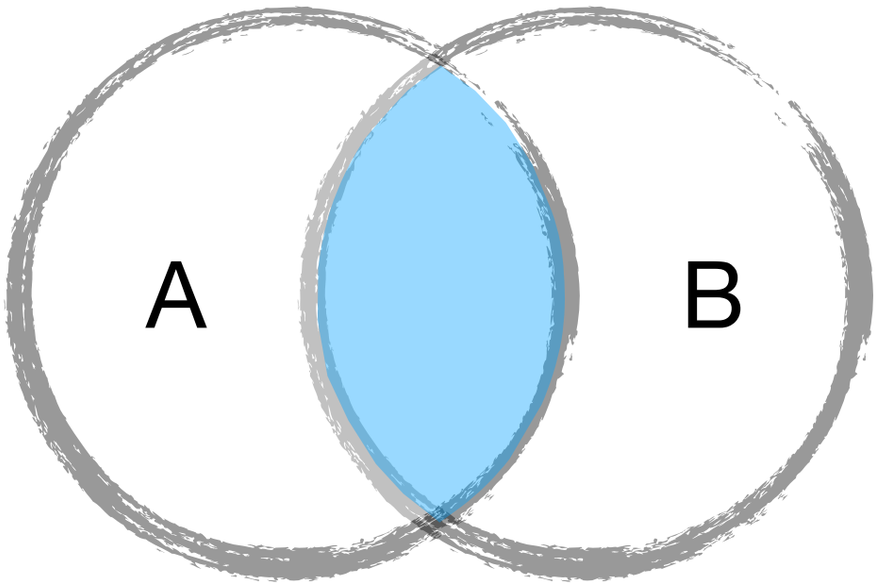
```
WITH RECURSIVE f (a,b) AS
  (SELECT 1 AS a, 1 AS b
   UNION ALL
   SELECT b, a+b
     FROM f
    WHERE b<2000)
SELECT a
  FROM f;
```

```
WITH RECURSIVE f (a,b) AS
  (SELECT 1 AS a, 1 AS b
   UNION
   SELECT a+1 AS a,
                b*(a+1) AS b
     FROM f WHERE a<10)
SELECT a, b
  FROM f;
```

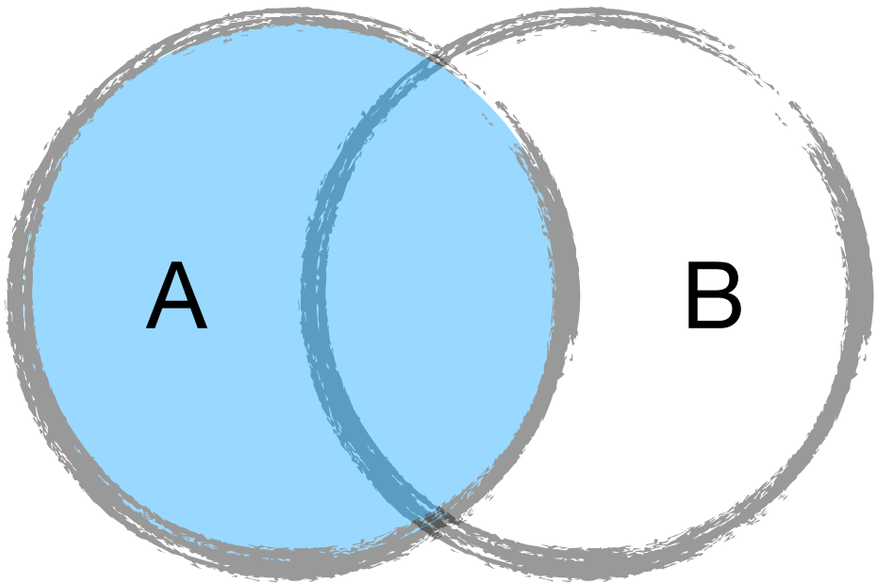
Cartesian Product (cross join)
Equi-join (inner join)
Self join
Left join
Right join
Full join
Natural join
Anti-join (NOT IN, NOT EXISTS)
Semi-join (EXISTS, IN)
Lateral join

```
SELECT *
  FROM og_result AS og
       CROSS JOIN
       og_region AS ogr
ORDER BY og.id, ogr.id
```

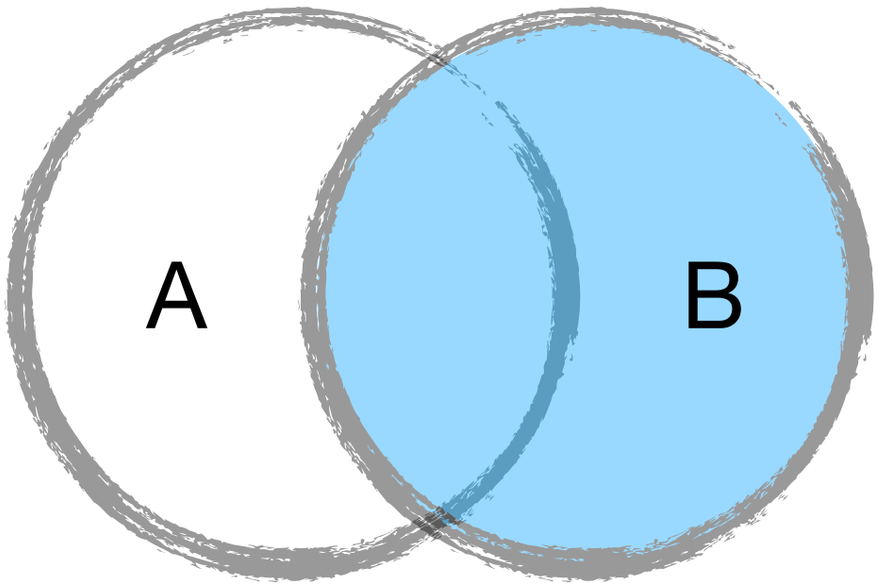
```
SELECT *
  FROM og_result AS og,
       og_region AS ogr
ORDER BY og.id, ogr.id
```



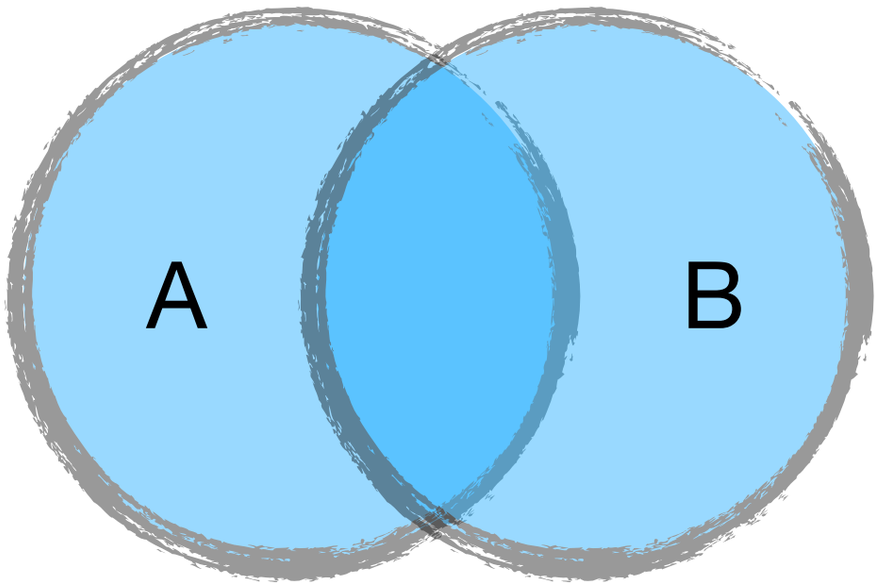

```
SELECT *
FROM A INNER JOIN B
ON A.key = B.key
```



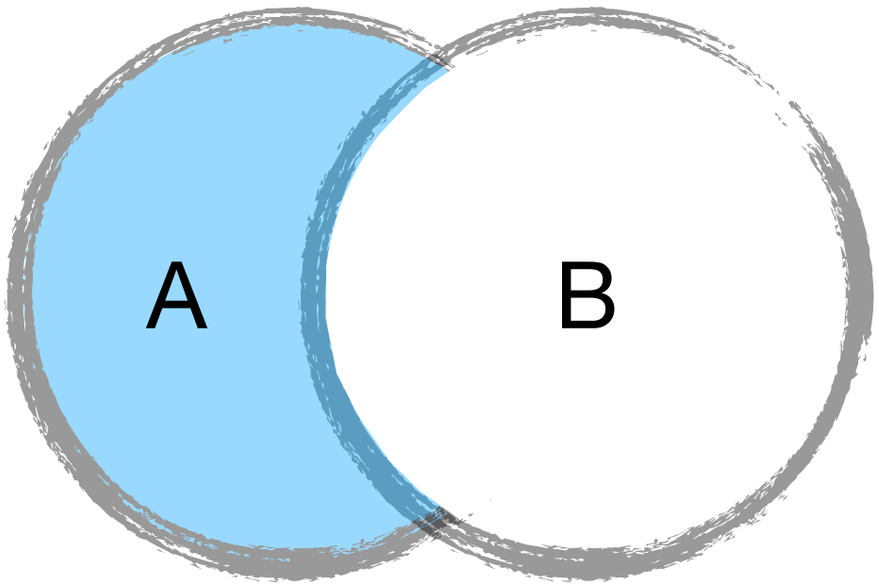

```
SELECT *
FROM A LEFT JOIN B
    ON A.key = B.key
```



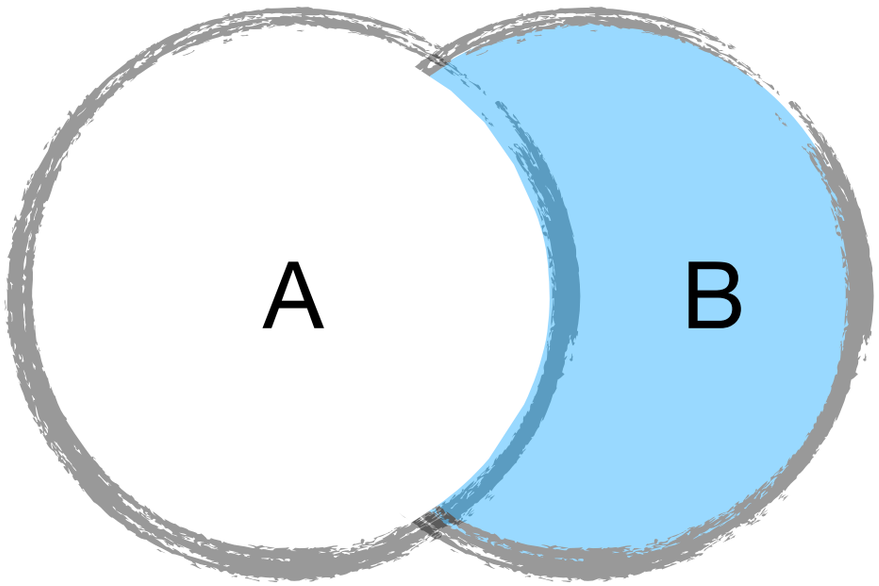

```
SELECT *
FROM A RIGHT JOIN B
ON A.key = B.key
```



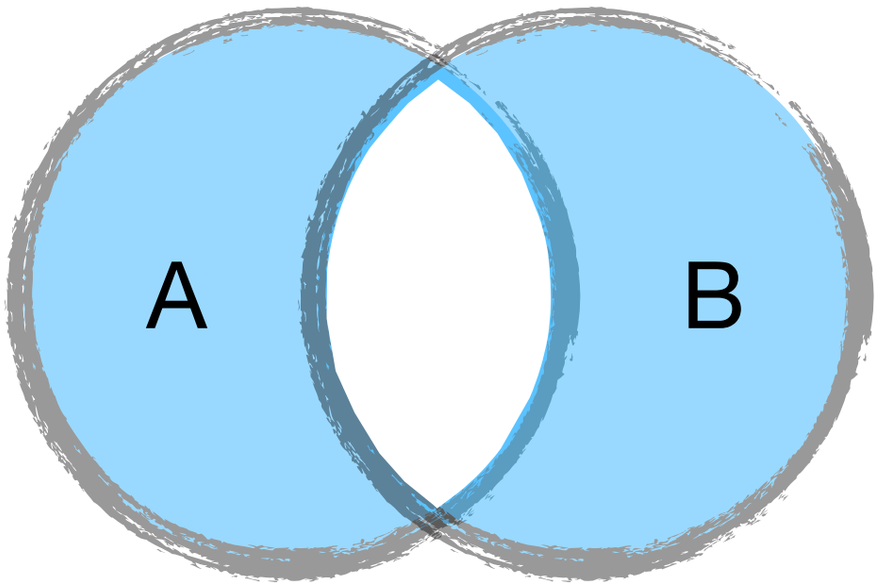

```
SELECT *
FROM A FULL JOIN B
    ON A.key = B.key
```



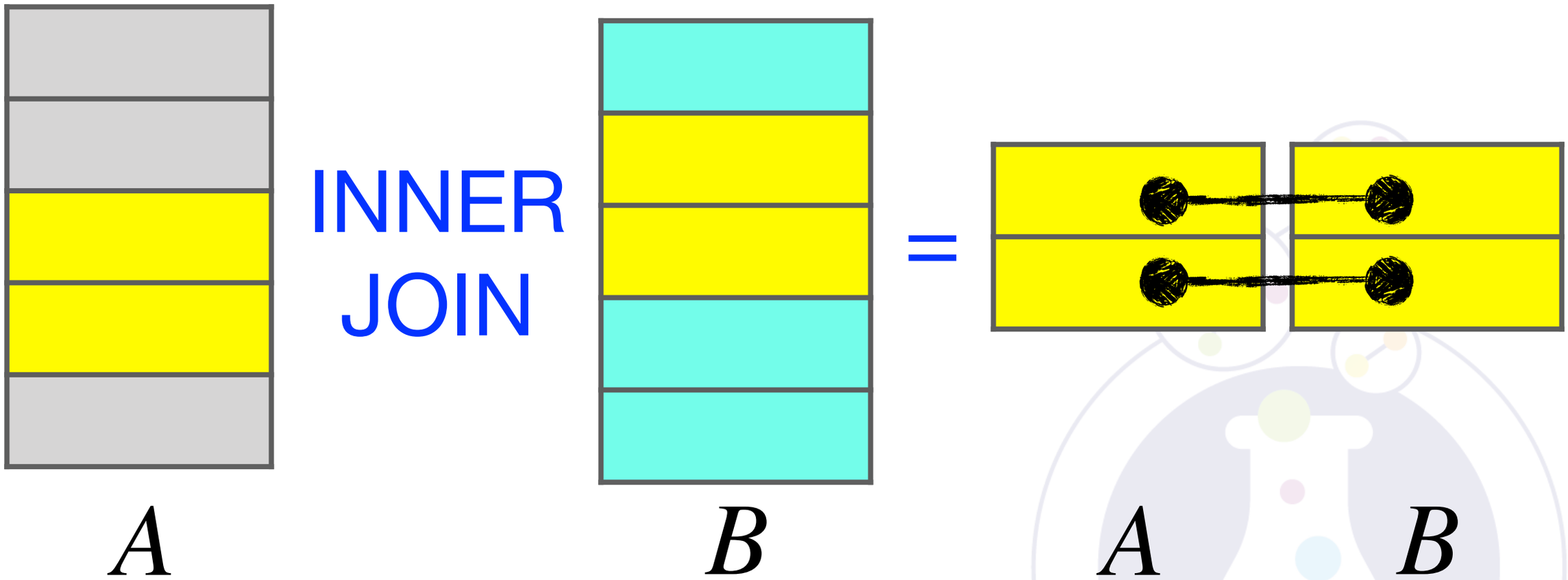

```
SELECT *
  FROM A LEFT JOIN B
    ON A.key = B.key
WHERE B.key IS NULL
```

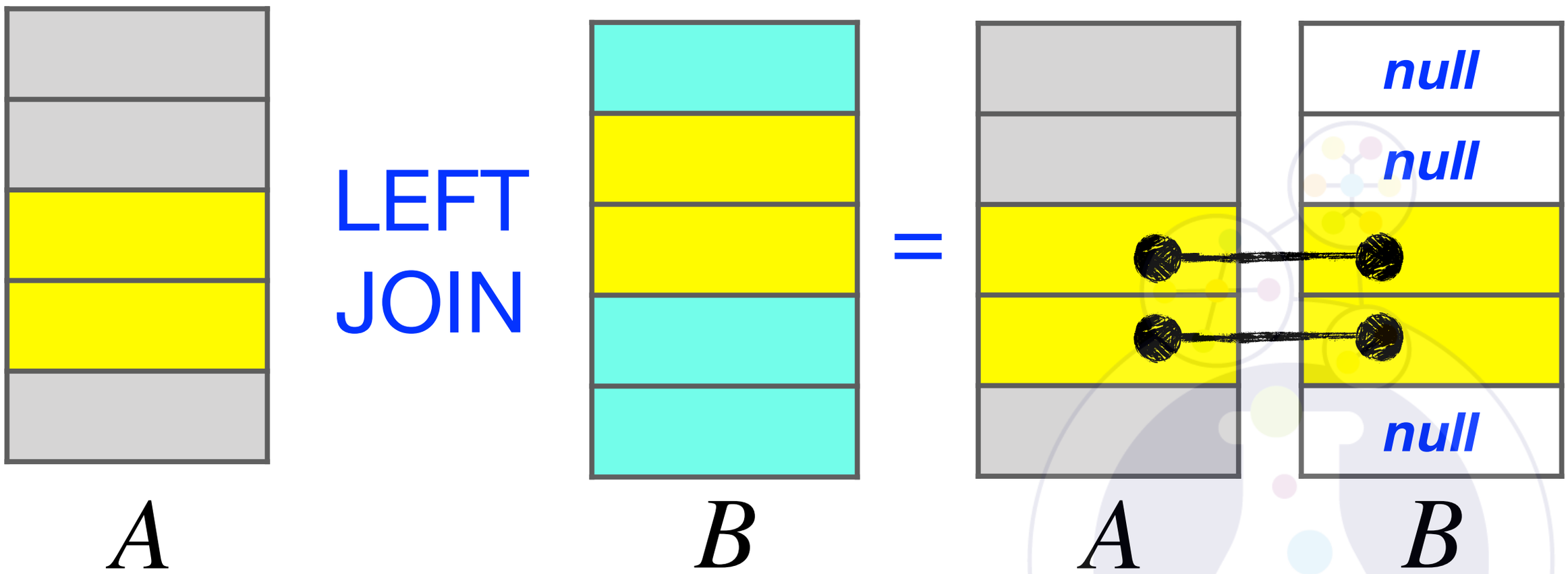


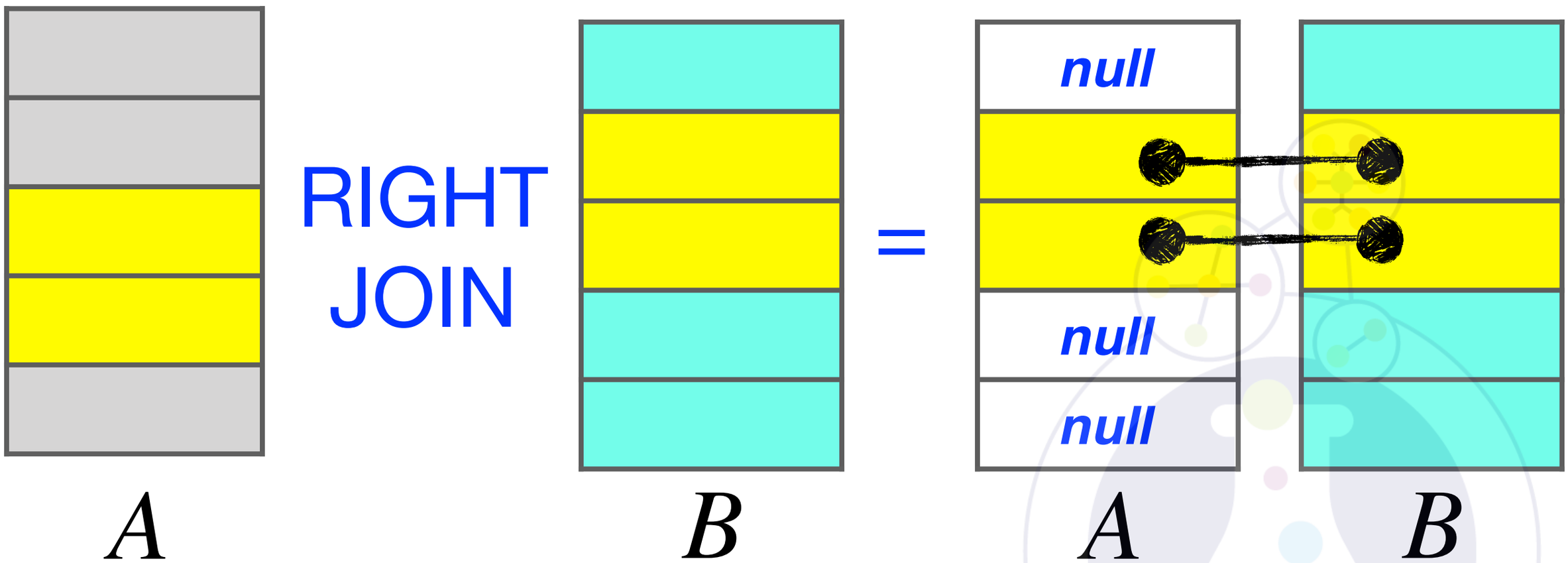

```
SELECT *
  FROM A RIGHT JOIN B
    ON A.key = B.key
WHERE A.key IS NULL
```

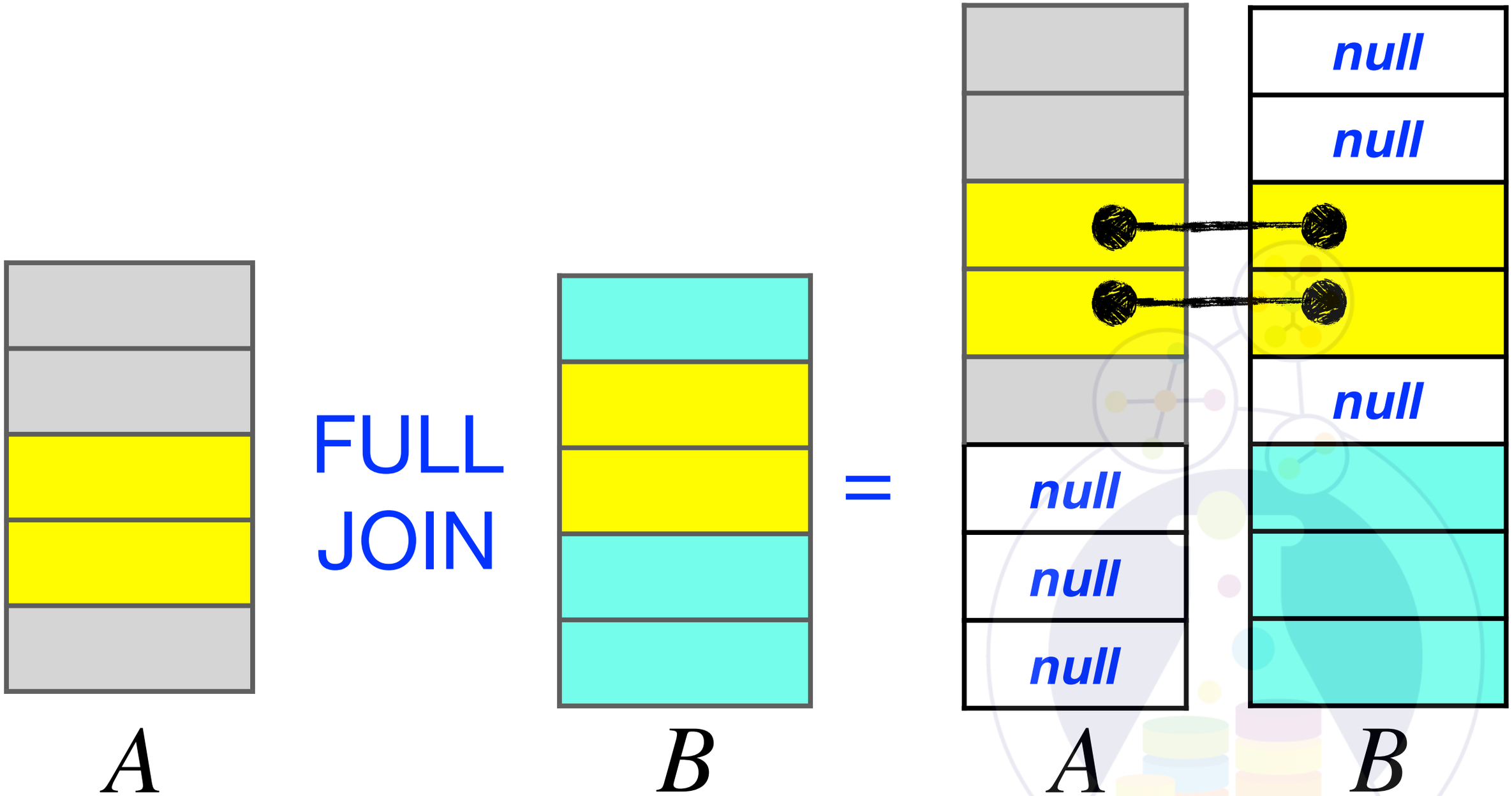



```
SELECT *
  FROM A FULL JOIN B
    ON A.key = B.key
WHERE A.key IS NULL OR
    B.key IS NULL
```









```
SELECT doc.id,
       doc.short_content,
       dk.project
FROM document doc
     INNER JOIN document_kind dk
     ON doc.document_kind = dk.id
```

```
SELECT doc.id,
       doc.short_content,
       dk.project
FROM document doc
     LEFT JOIN document_kind dk
     ON doc.document_kind = dk.id
```

```
SELECT doc.id,
       doc.short_content,
       dk.project
FROM document doc
     RIGHT JOIN document_kind dk
     ON doc.document_kind = dk.id
```

```
SELECT doc.id,
       doc.short_content,
       dk.project
FROM document doc
       FULL JOIN document_kind dk
       ON doc.document_kind = dk.id
```

```
SELECT doc.id,
       doc.short_content,
       dk.project
  FROM document doc
    INNER JOIN document_kind dk
      ON doc.document_kind = dk.id
    INNER JOIN document_delivery_types ddt
      ON doc.delivery_type = ddt.id
```

```
SELECT doc.id,
       doc.short_content,
       dk.project
  FROM document doc
    RIGHT JOIN document_kind dk
      ON doc.document_kind = dk.id
    LEFT JOIN document_delivery_types ddt
      ON doc.delivery_type = ddt.id
```

```
SELECT doc.id,
       doc.short_content,
       dk.project
  FROM document doc
    RIGHT JOIN document_kind dk
      ON doc.document_kind = dk.id
    LEFT JOIN document_delivery_types ddt
      ON doc.delivery_type = ddt.id
WHERE doc.deleted != 1
```

```
SELECT *
    FROM Student
WHERE StudentID IN
                (SELECT id
                    FROM Hobby)
```

```
SELECT *
    FROM Student
WHERE StudentID NOT IN
                (SELECT id
                    FROM Hobby)
```

```
SELECT *
  FROM document doc
WHERE document_kind IN
        (SELECT id
           FROM document_kind
          WHERE category = 1)
```

```
SELECT *
   FROM Student s
 WHERE EXISTS
          (SELECT 1
             FROM Hobby sh
            WHERE s.id = sh.StudentID)
```

```
SELECT *
   FROM Student s
 WHERE NOT EXISTS
          (SELECT 1
             FROM Hobby sh
            WHERE s.id = sh.StudentID)
```

```
SELECT *
  FROM document doc
WHERE EXISTS
        (SELECT 1
           FROM document_kind dk
          WHERE doc.document_kind = dk.id
            LIMIT 1)
```

```
SELECT *
  FROM document doc
WHERE EXISTS
        (SELECT 1
          FROM document_kind dk
         WHERE doc.document_kind = dk.id
               AND category = 1
         LIMIT 1)
```

```
SELECT *
  FROM document doc
WHERE EXISTS
        (SELECT 0
           FROM document_kind dk
          WHERE doc.document_kind = dk.id
            LIMIT 1)
```

```
SELECT *
  FROM document doc
WHERE EXISTS
        (SELECT null
           FROM document_kind dk
          WHERE doc.document_kind = dk.id
            LIMIT 1)
```

```
SELECT *
FROM P NATURAL JOIN S
```

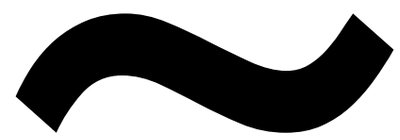
```
SELECT *
  FROM P NATURAL JOIN S
```

$\sim$

```
SELECT *
  FROM P CROSS JOIN S
 WHERE P.ID = S.ID
```

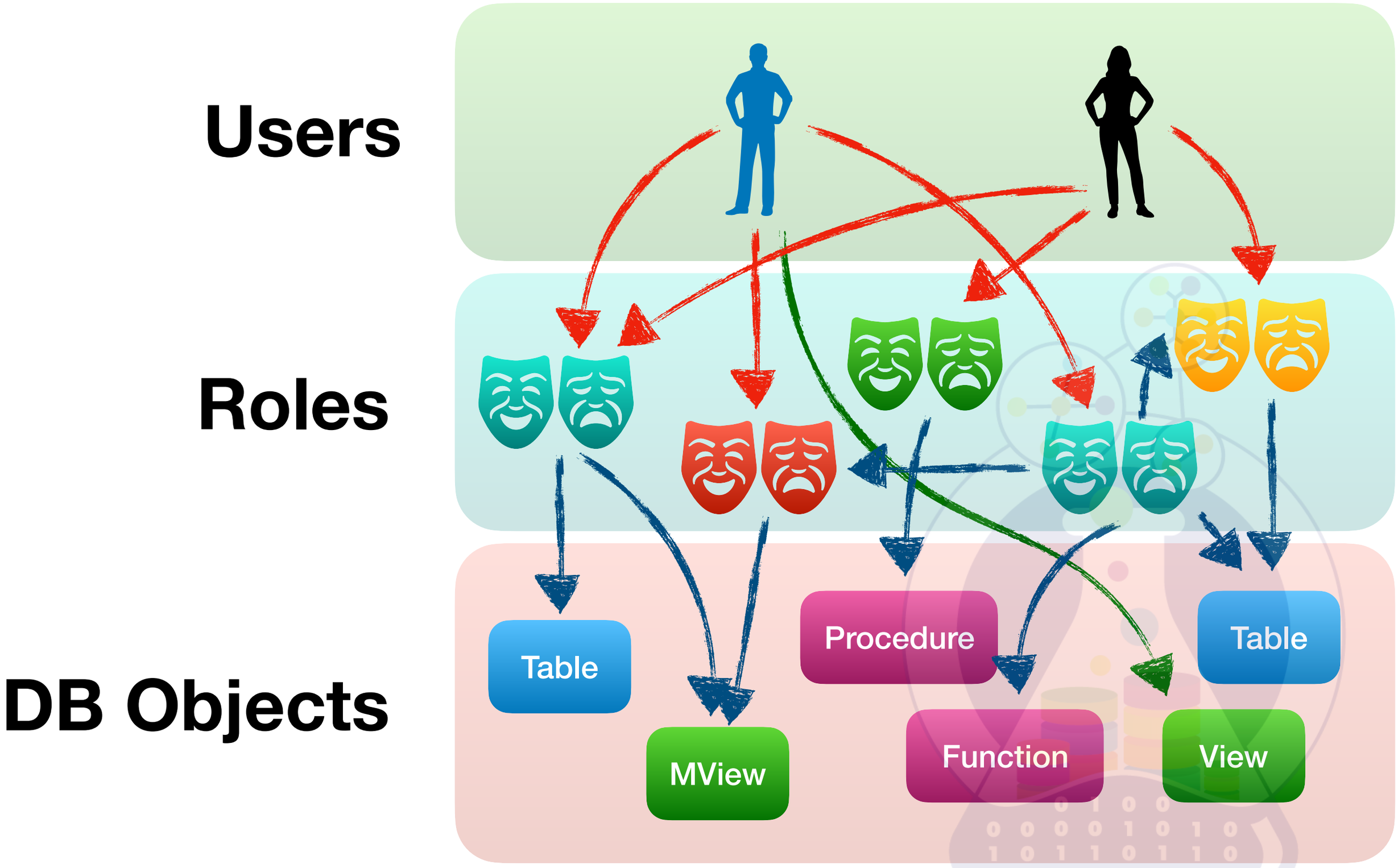
```
SELECT *
  FROM document doc
      CROSS JOIN LATERAL
  (SELECT id
     FROM document_kind dk
   WHERE category = 1 AND
           dk.id = doc.document_kind) AS t1
```

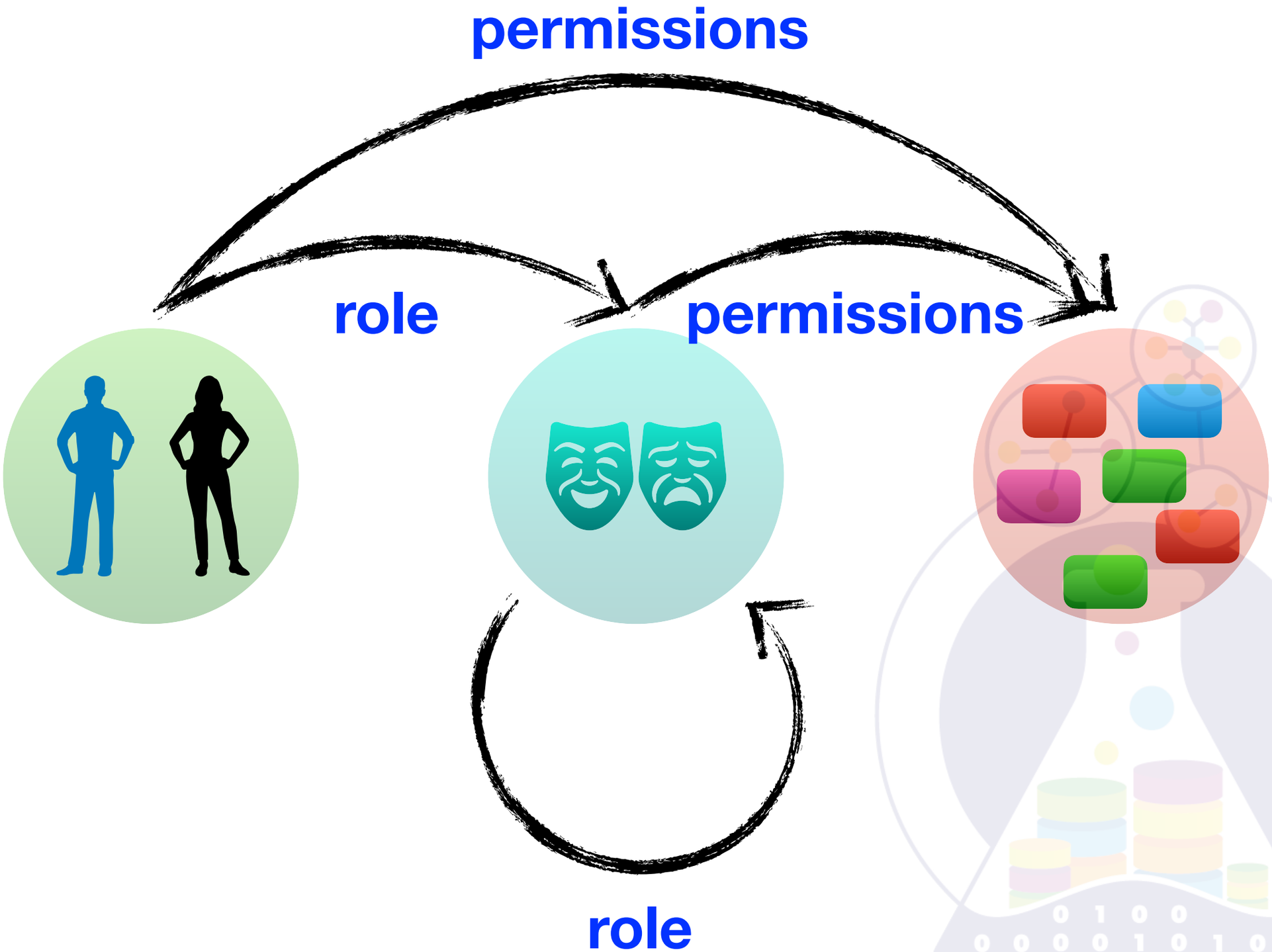
```
SELECT *
  FROM document doc
     CROSS JOIN LATERAL
     (SELECT id FROM document_kind dk
       WHERE category = 1 AND
                 dk.id = doc.document_kind) AS t1
```

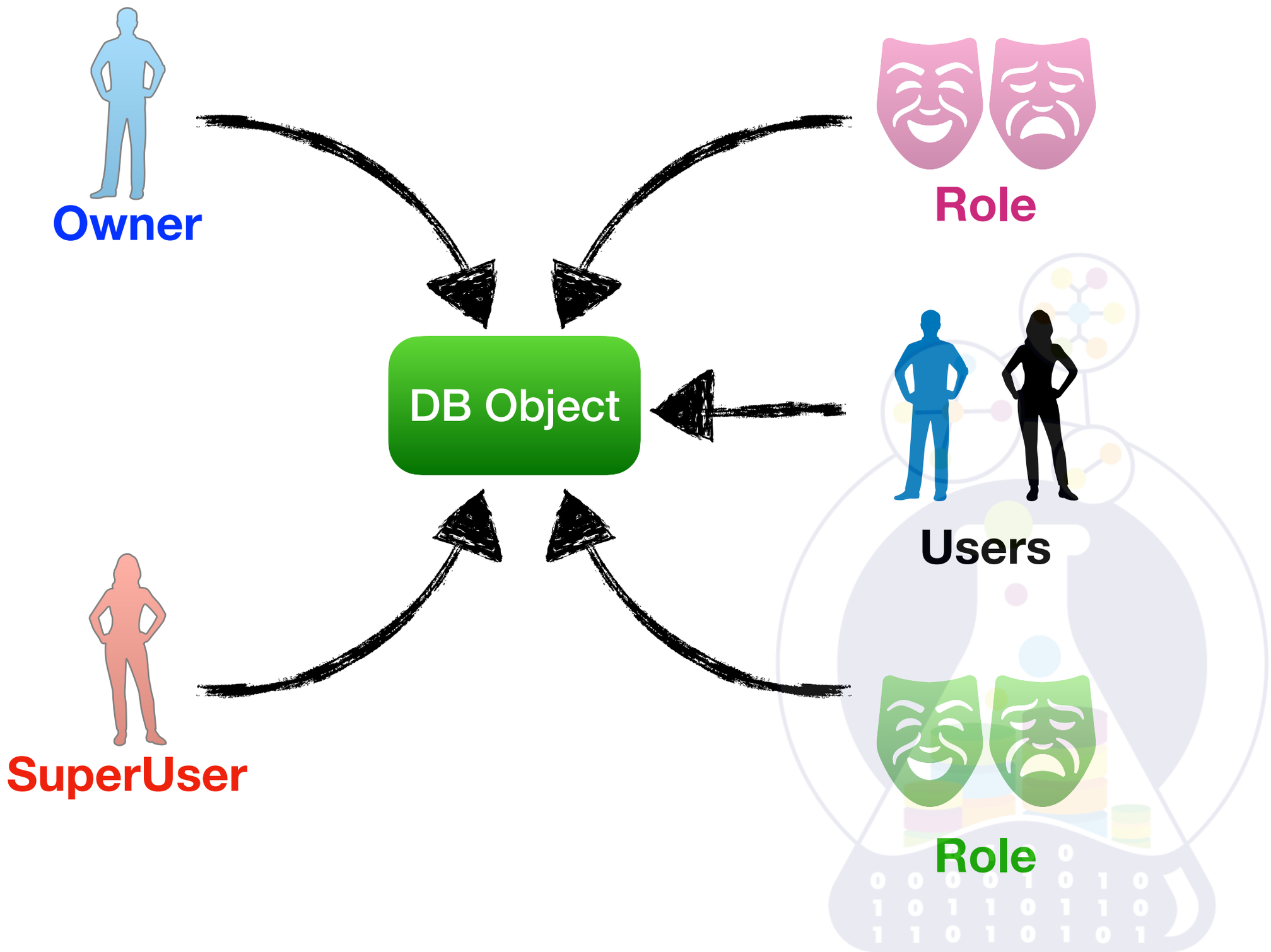


```
SELECT *
  FROM document doc
WHERE document_kind IN
            (SELECT id
               FROM document_kind
             WHERE category = 1)
```

DCL







GRANT **SELECT** ON … TO …

GRANT **INSERT** ON … TO …

GRANT **UPDATE** ON … TO …

GRANT **DELETE** ON … TO …

GRANT **TRUNCATE** ON … TO …

GRANT **ALTER** ON … TO …

GRANT **INDEX** ON … TO …

GRANT **REFERENCES** ON … TO …

GRANT **CREATE** ON … TO …

GRANT **EXECUTE** ON … TO …

GRANT **USAGE** ON … TO …

GRANT **TEMPORARY** ON … TO …

GRANT **CONNECT** ON … TO …

GRANT **TRIGGER** ON … TO …

GRANT SELECT ON Student TO test;

GRANT TRUNCATE ON Student TO test1;

GRANT **ALL** ON Student TO test2;

GRANT **SELECT(Phone), UPDATE(Address)** ON Student TO test;

```
REVOKE SELECT ON Student FROM test;

REVOKE ALL ON Student FROM test2;
```

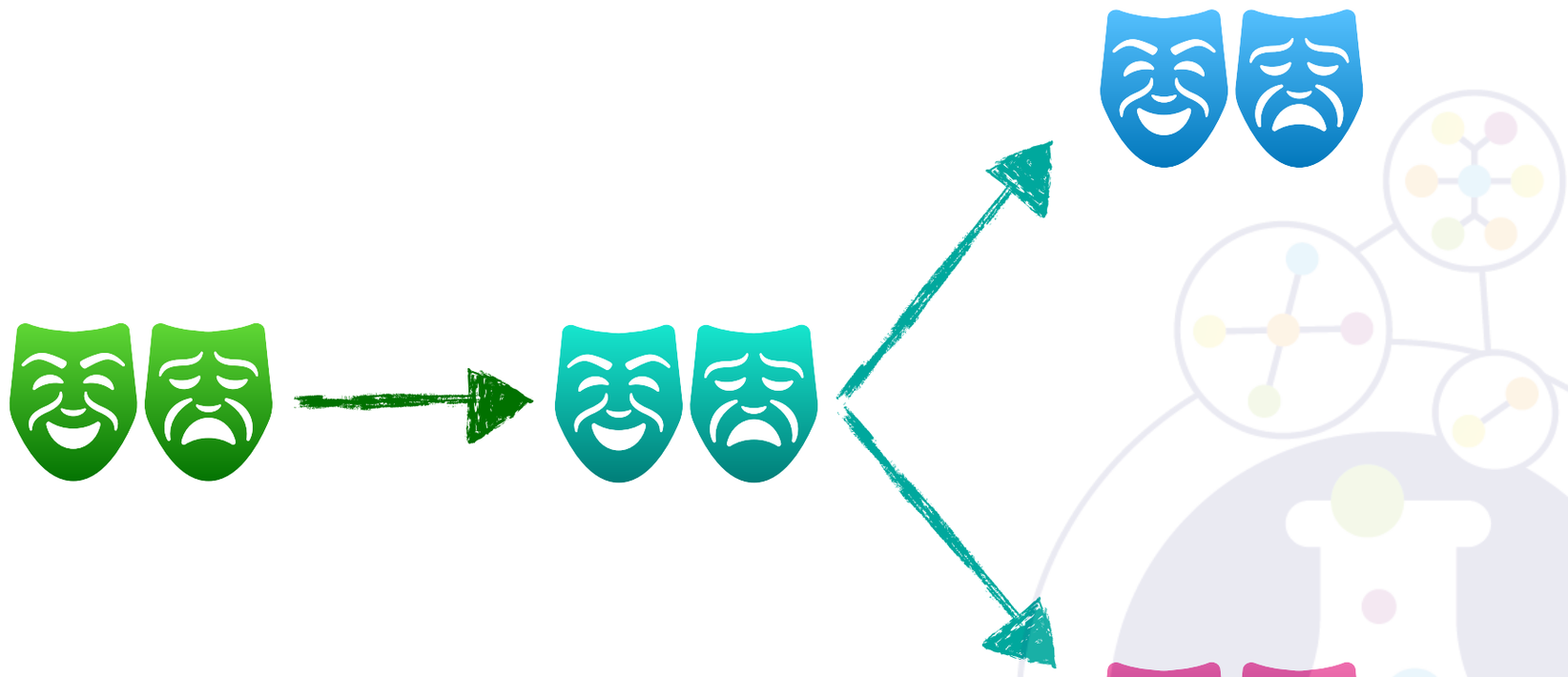
CREATE USER test WITH PASSWORD ‘123’;

CREATE USER test IDENTIFIED BY ‘123’;

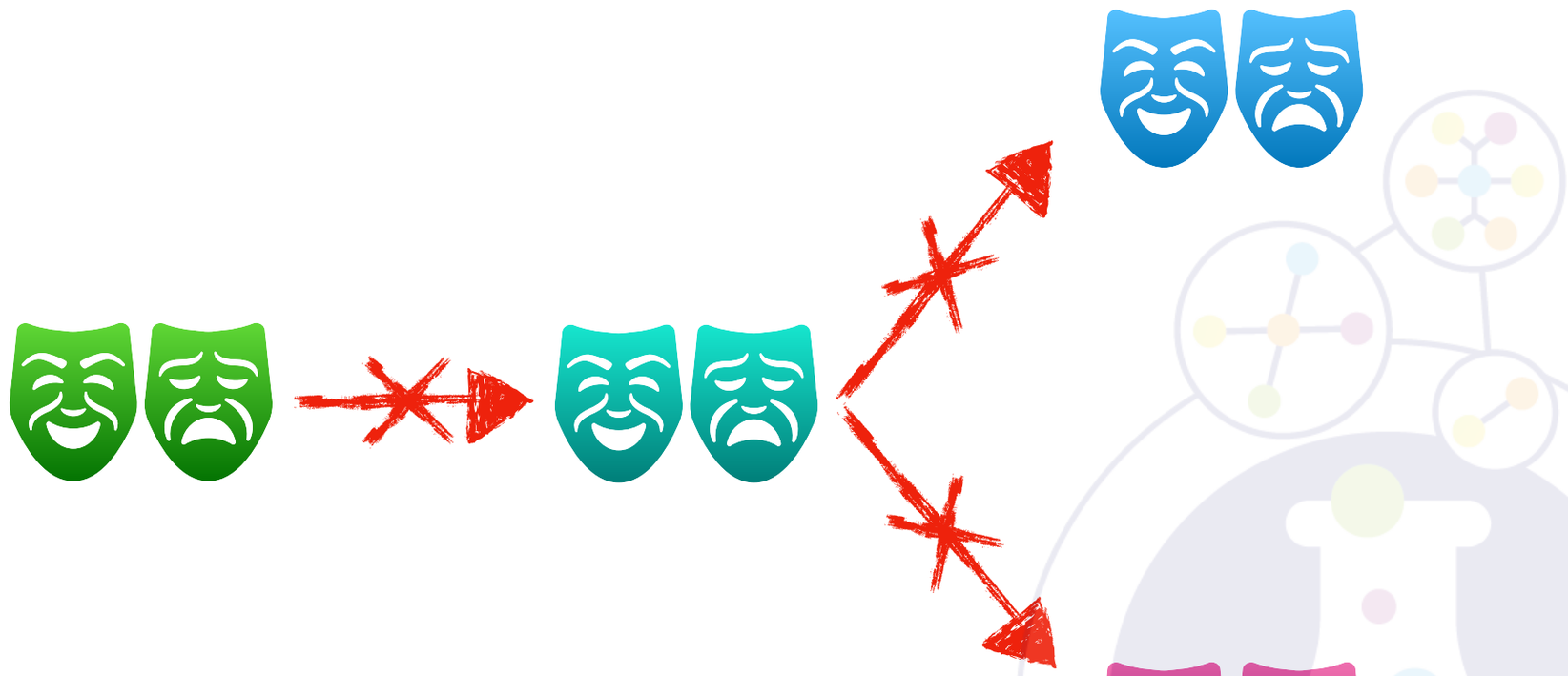
CREATE USER test@‘%’ IDENTIFIED BY ‘123’;

```
CREATE ROLE test1;
CREATE ROLE test2;
GRANT test1 TO test2;
REVOKE test1 FROM test2;
```

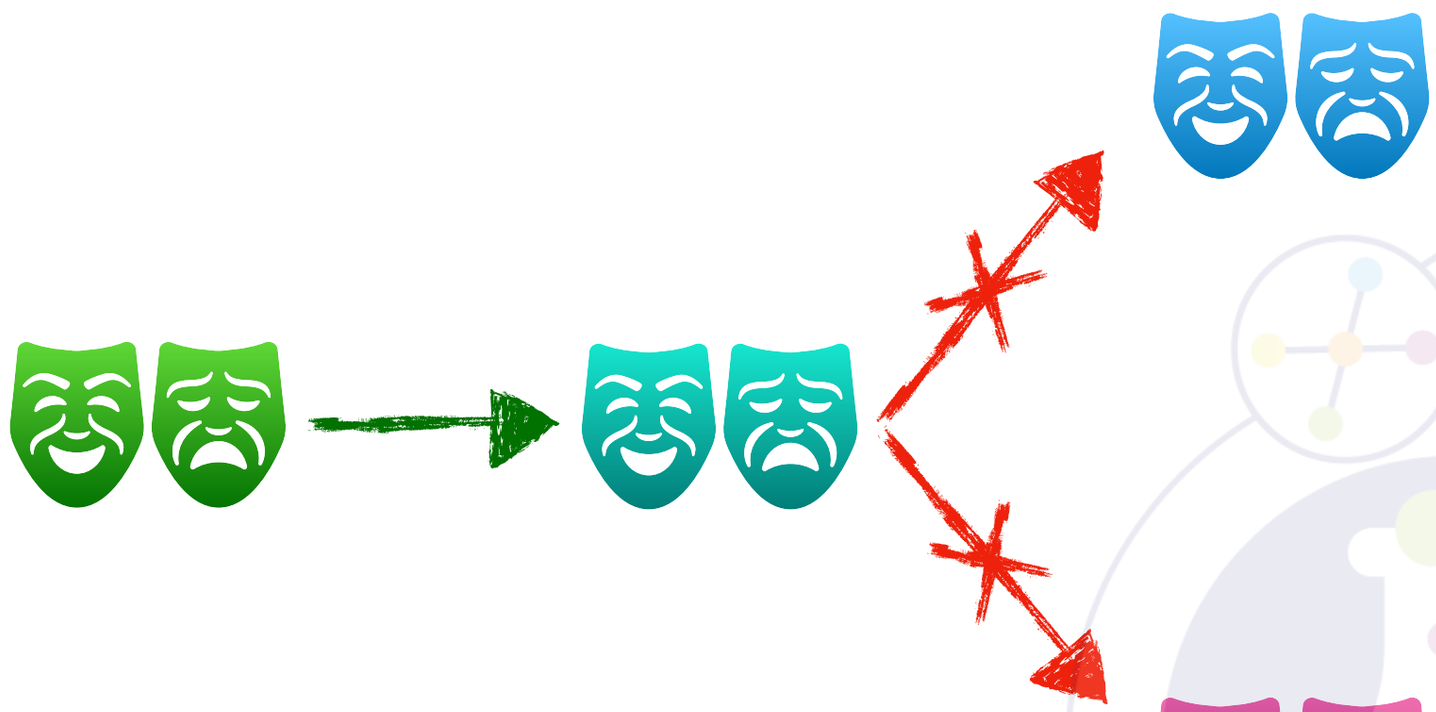
GRANT SELECT ON Student TO test
WITH GRANT OPTION;



REVOKE SELECT ON Student FROM test **CASCADE**;



REVOKE GRANT OPTION FOR
SELECT ON Student FROM test
CASCADE;



COMMIT;