ODBC
and some other stuff

Yet more complete select

```
SELECT selection_list  # Define what the columns in the relation will be

FROM table_list        # fill in the columns from the listed tables
                       # Does cross product if there are multiple tables

[INNER, RIGHT [OUTER], LEFT [OUTER], FULL [OUTER], NATURAL] JOIN (table [on
constraints])*         # default is to inner join
                       # natural will look for identical column names or
                       # "foreign key" relationships

WHERE constraint+      # Select the rows in the temp table after FROM completes
                       # such that the rows match the given constraint

GROUP BY columns       # groups the remaining rows by the given columns
    HAVING group constraints      # select the grouped rows by the constraint

ORDER BY sorting_cols  # Order the remaining rows by the given columns

LIMIT count;           # Limit on results
```

Joins

select count(*) from airports ***** join flights [on flights.origin = airports.tla;]

```
\d flights
     Column     |          Type          | Collation | Nullable |    Default
----------------+------------------------+-----------+----------+---------------
 date           | date                   |           |          | CURRENT_DATE
 departuretime  | time without time zone |           |          | CURRENT_TIME
 arrivaltime    | time without time zone |           |          | CURRENT_TIME
 carrier        | character varying(3)   |           |          |
 flightnum      | integer                |           |          | '-1'::integer
 arrivaldelay   | integer                |           |          | 0
 departuredelay | integer                |           |          | 0
 origin         | character(3)           |           |          |
 destination    | character(3)           |           |          |
 distance       | integer                |           |          | 0
 cancelled      | boolean                |           |          | false
Foreign-key constraints:
    "c2" FOREIGN KEY (origin) REFERENCES airports(tla)
    "c3" FOREIGN KEY (destination) REFERENCES airports(tla)
Count == 196431


flight=# \d airports
                  Table "public.airports"
 Column  |         Type          | Collation | Nullable | Default
---------+-----------------------+-----------+----------+---------
 name    | character varying(60) |           |          |
 country | character varying(50) |           |          |
 tla     | character(3)          |           | not null |
Indexes:
    "pk_name" PRIMARY KEY, btree (tla)
Referenced by:
    TABLE "flights" CONSTRAINT "c2" FOREIGN KEY (origin) REFERENCES airports(tla)
    TABLE "flights" CONSTRAINT "c3" FOREIGN KEY (destination) REFERENCES airports(tla)
Count = 1948
```

type	count
Natural (no "on")	382647588
inner	196431
right	196431
left	198320
full	198320

382647588=196431*1948

Right and Left

Joins

LEFT

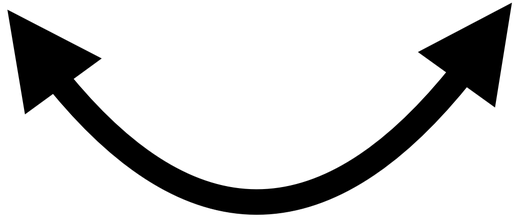
RIGHT

Keep everything on the left
and matching from the right

select count(*) from airports left join flights [on flights.origin = airports.tla;]

The same!!!!

select count(*) from flights right join airports [on flights.origin = airports.tla;]



WITH

the last(?) piece of select

- Prior to select create temporary relations that can be used in select as actual relations
  - Easy to test out the temporary ones

- Question: how may rows are then in the cross product of airports and flights
  - select count(*) from flights, airports;
    - takes about 9 seconds;
- Do this without actually generating the cross product:
  - SELECT count(*) from flights;
  - SELECT count(*) from airports;

- WITH air(cair) as (SELECT count(*) from airports),
  fli(cfli) as (SELECT count(*) from flights)
  SELECT air.cair*fli.cfli from air, fli;
  - takes about 10 milliseconds (100x speedup)

ODBC

Ch 5.1 in Sail book

Open DataBase Connectivity

- a standard API for accessing DBMS
  - Competing system called CLI -- Call Level Interface
  - API

- Central idea -- people should be able to interact with database programmatically in a standard way
  - the particular query may be different, but that way in which a program connects to the DB and receives information from the DB is the same regardless of DB

Java

JDBC

- Java DataBase Connectivity
  - A set of interface classes -- Virtually no implementations
  - `import java.sql.*;`
- Since JDBC is a standard specification, one Java program that uses the JDBC API can connect to any database management system (DBMS), as long as a driver exists for that particular DBMS.

- Realistically this was intended for SQL DBS, but has been ported to noSQL DBs
  - PostgreSQL driver
    - postgresql-42.7.1.jar
  - mongo
    - mongo-jdbc-2.1.10.jar
    - mongo-java-driver-3.12.10.jar (not JDBC)
    - mongodb-driver-sync-4.11.1.jar (not JDBC)

JDBC steps

- Connect to DBMS and to a particular DB
  - get a "connection" object
- Pass query to DB
  - create a "statement object"
  - Put query into statement
  - Execute statement
    - returns a "ResultSet" object
- Read results
  - row by row from result object

JDBC: 1 Connect

- Consult with DB provider for connection URI
  - PG: "jdbc:postgresql://localhost:5432/" + dbName
    - E.G.:"jdbc:postgresql://localhost:5432/flight"
- Login using username & password
  - the account MUST have password authentication
    - At least I have not figured out another way

```
private final String user = "ME_123";
// the password -- IN CLEAR TEXT!!!
private final String password = "12345678";

/**
 * Connect to the PostgreSQL database
 *
 * @return a Connection object
 */
public Connection connect(String dbName) {
    Connection conn = null;
    try {
        String url = "jdbc:postgresql://localhost:5432/" + dbName;
        conn = DriverManager.getConnection(url, user, password);
        System.out.println("Connected!!");
    } catch (SQLException e) {
        System.err.println(e.getMessage());
    }
    return conn;
}
```

Must be either localhost or 127.0.0.1
loin.cs.brynmawr.edu and 165.106.10.133 DO NOT WORK

JDBC: 2--Get Query to DB

- 2 options
  - Get a Statement object
    - Put a full query into the statement
    - Execute
  - Get a PreparedStatement object
    - Put a wildcarded query into statement
    - Fill in wildcards
    - Execute

Query Using Statement

```
        Connection conn = app.connect(DB_NAME);
        try {
             Statement st = conn.createStatement();
             ResultSet rs = null;
            if () { // Querying the sakila database
                 rs = st.executeQuery("SELECT first_name, last_name from actor where last_name like
'A%'");
            } else { // Queryinf the univ database
                 int courseNum = Integer.parseInt(args[1]);
                 rs = st.executeQuery(String.format("SELECT course_id,title, dept_name, credits FROM
course WHERE cast(course_id as int)>%d LIMIT 10", courseNum));
            }

        } catch (Exception ee) {
             System.err.println(ee);
             ee.printStackTrace();
         }
```

Getting query parameter from command line

Sometimes automatic type coercion, sometimes not

Query Using PreparedStatement

```
// in the rocket database
PreparedStatement ps = conn.prepareStatement("Select sitecode, latitude from site where
latitude>?");
ps.setDouble(1, lati);
rs = ps.executeQuery();
```

- Prepared statement is MUCH safer
- If you are getting parameters from users, use PreparedStatement
  - SQL injection attacks
    - Evil users can delete your data!!!!

Reading Data

Working through the ResultSet object

- rs.next() goes to next row
- access columns by name or position
- position is 1 indexed
  - 0 throws an error!

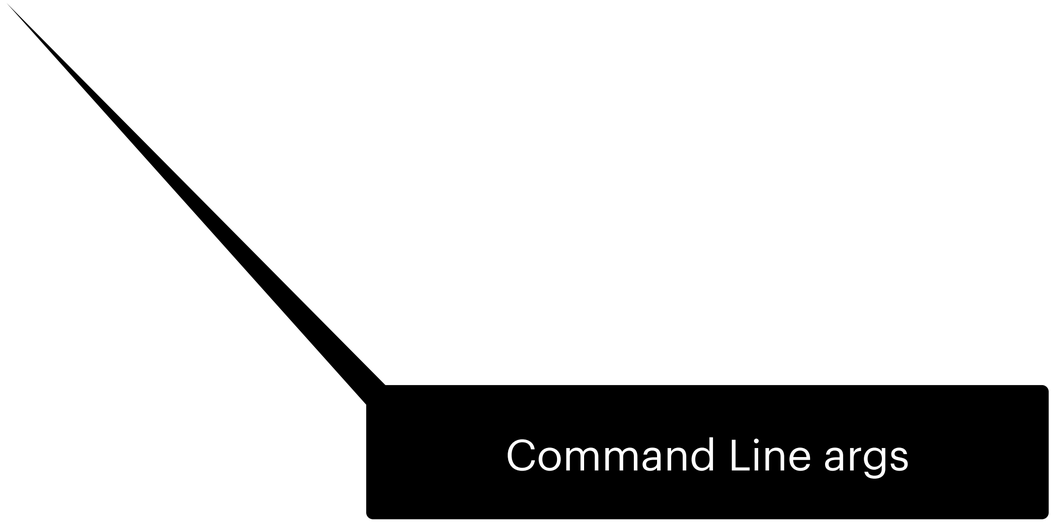
```
ResultSet rs = st.executeQuery......
while (rs.next()) {
     System.out.format("first %s last:%s\n",
rs.getString("first_name"),
rs.getString("last_name"));

      System.out.format("c: %s t:%s\n",
rs.getString(1), rs.getString(2));
}
```

Compile and Run

Finally!

- javac GTJDBC.java

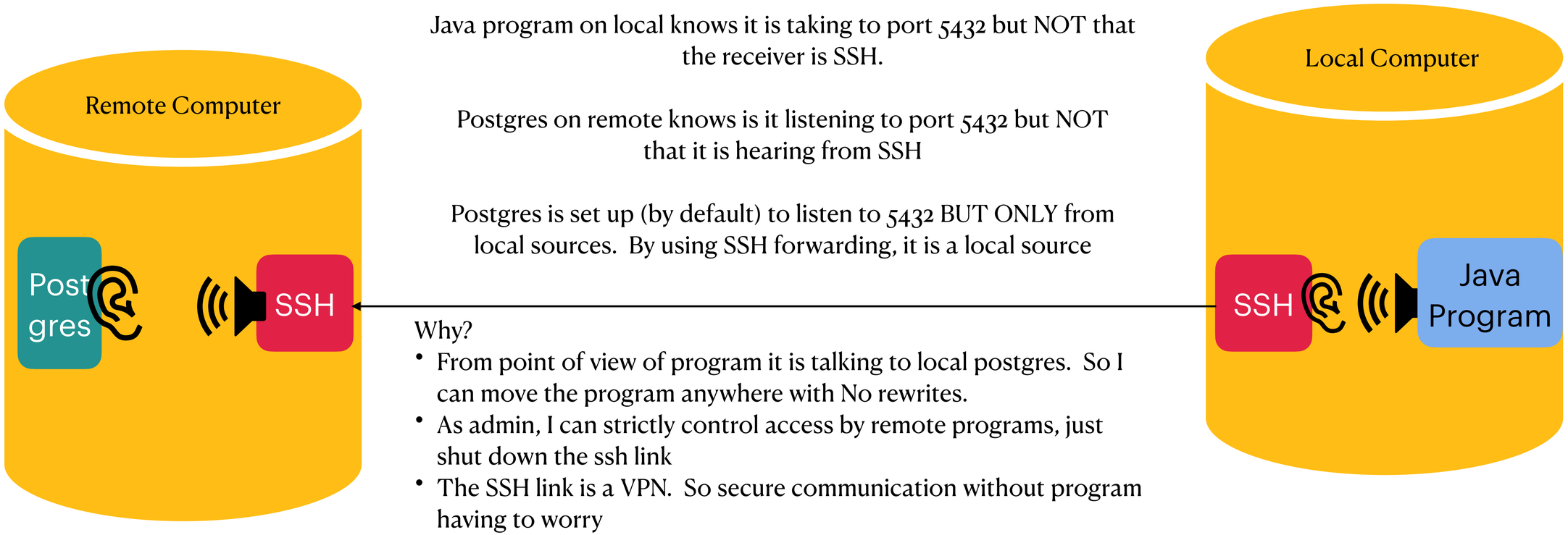
- java -cp .:postgresql-42.7.1.jar GTJDBC univ 900



Local Forward in SSH

For example using java and postgres

Recall: LocalForward 5432 localhost:5432



Java program on local knows it is taking to port 5432 but NOT that the receiver is SSH.

Postgres on remote knows is it listening to port 5432 but NOT that it is hearing from SSH

Postgres is set up (by default) to listen to 5432 BUT ONLY from local sources. By using SSH forwarding, it is a local source

Why?

- From point of view of program it is talking to local postgres. So I can move the program anywhere with No rewrites.
- As admin, I can strictly control access by remote programs, just shut down the ssh link
- The SSH link is a VPN. So secure communication without program having to worry
- Run compute heavy analysis NOT on DB server

use pg8000
rather
than psycopg2

Remote Forward in SSH

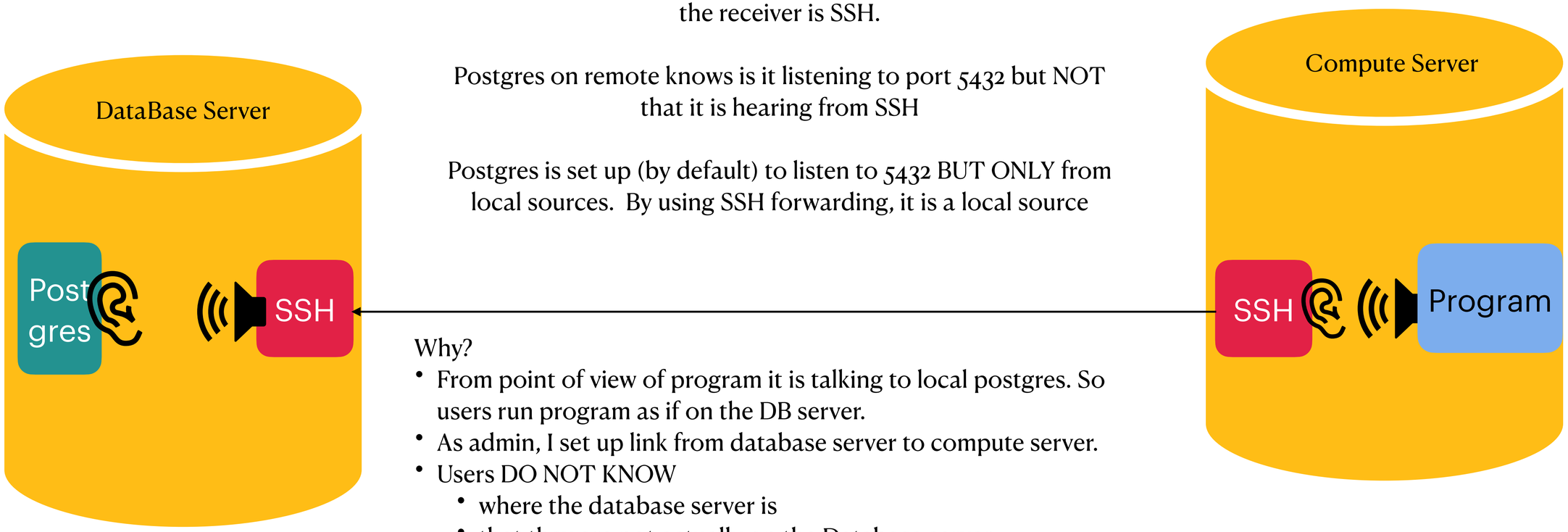
plus a little more

RemoteForward 5432 localhost:5432

Java program on local knows it is taking to port 5432 but NOT that the receiver is SSH.

Postgres on remote knows is it listening to port 5432 but NOT that it is hearing from SSH

Postgres is set up (by default) to listen to 5432 BUT ONLY from local sources. By using SSH forwarding, it is a local source



Why?

- From point of view of program it is talking to local postgres. So users run program as if on the DB server.
- As admin, I set up link from database server to compute server.
- Users DO NOT KNOW
  - where the database server is
  - that they are not actually on the Database server
- As admin, I can turn off user access to the DB server by breaking the SSH connection to the compute server
- Guarantees that compute heavy analysis is NOT on DB server

use pg8000
rather
than psycopg2

ODBC with Python

- import psycopg2
- connect
  - again, clear text password
- Set query
- Execute
- Read/Print Results
- Close things neatly

```
import psycopg2

try:
    connection = psycopg2.connect(user="USR_123",
                                  password="12345678",
                                  host="127.0.0.1",
                                  port="5432",
                                  database="rocket")
    cursor = connection.cursor()
    postgreSQL_select_Query = "select sitecode, type, country from site"
    cursor.execute(postgreSQL_select_Query)
    mobile_records = cursor.fetchall()

    for row in mobile_records:
        print("sitecode = {0} {1} {2}".format(row[1], row[1], row[2] ))

except (Exception) as error:
    print("Error while fetching data from PostgreSQL", error)

finally:
    # closing database connection.
    if connection:
        cursor.close()
        connection.close()
        print("PostgreSQL connection is closed")
```

Python and SQL injection

- Do NOT use string formatting to build query
- Do build query in the execute statement

```
select_query = """select * from Hospital where Hospital_Id = %s"""
cursor.execute(select_query, (hospital_id,))
```

python and postgres

psycopg2 vs pg8000

- 2 ODBC packages for postgres from python
  - psycopg2
    - from postgreSQL
    - pretty much the standard
    - very actively maintained
    - better when running on the DB server
  - pg8000
    - "pure python" so installs / runs anwhere
    - IMO better when port forwarding
      - literally replace any "psycopg2" with pg8000

Functions in SQL

There are lots of functions that can be used to create columns

- count()
- row_number() OVER (xxx)
  - select column, row_number() over (xxx) from table;
    - xxx can be empty uses table ordering
    - xxx 'order by user_id' may be different from table ordering
- rank()
  - select column, rank( ) from table;
    - xxx can NOT be blank
    - order by column
      - partition by column
- date_part(‘partname’, column_name) // column should be a date
  - partnames: month, day, year, ...
  - date_part is NOT in SQL standard, but almost every RDBMS has it
- substring(col, first index, length)
  - substring

in Sakila database

How many entries in actor table?

make a table with row numbers showing each actor whose name begins with W so that row number are alphabetical by last name. Sort the table by first name!

make a table of actors whose last name ends with 'E' such that there is a column showing ranking of each actor where ranks are within same last name and ordered by first name

make a table of all actors whose record was updated in February

make a table showing only the second and third characters of each actors last name such that the second and third characters are 'AW'.

SQL Problem

- From each department, find the names of the people earning the 2 highest salaries

- Suggestion -- use rank() function
  rank() over (partition by dept_name order by salary)

- BUT
- select Name, salary, rank() over (partition by dept_name order by salary) from instructor;
- this get you everyone, ranked and you only want the top 2!
- univ=# select * from (select Name, salary, rank() over (partition by dept_name order by salary) as rank from instructor) as sq where sq.rank<=2;
- or using with
- univ=# with sq(name, salary, rank) as (select Name, salary, rank() over (partition by dept_name order by salary) as rank from instructor) select * from sq where sq.rank<=2;

id	name	dept_name	salary
31955	Moreira	Accounting	71351.42
79081	Ullman	Accounting	47307.10
43779	Romero	Astronomy	79070.08
63287	Jaekel	Athletics	103146.87
16807	Yazdi	Athletics	98333.65
81991	Valtchev	Biology	77036.18
80759	Queiroz	Biology	45538.32
34175	Bondi	Comp. Sci.	115469.11
3335	Bourrier	Comp. Sci.	80797.83
90376	Bietzk	Cybernetics	117836.50
63395	McKinnon	Cybernetics	94333.99

- From each department, find the names of the people earning the 2 highest salaries

```
with aaa as (select distinct dept_name from instructor),
     bbb as (select aaa.dept_name as deptt, salary as sall
                  from aaa join lateral (select dept_name, salary from instructor sii
                                         where sii.dept_name=aaa.dept_name
                                         order by salary desc
                                         limit 2) as lsi
                                  on lsi.dept_name=aaa.dept_name
                  order by aaa.dept_name asc, lsi.salary desc)

select id, name, dept_name, salary
from instructor as i1, bbb
where i1.dept_name=bbb.deptt and i1.salary=bbb.sall
order by dept_name asc, salary desc;
```

"Lateral" Join is postgreSQL only