**IBM Netezza Use Case**

Team Name: Code Breakers

Sadhana Edukulla Class ID: 17

Pardha Saradhi Koye Class ID: 40

Koushik Nallani Class ID: 48

Nithin Sai Peram Class ID: 50

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Improving Performance of Big Data Queries** | | | | | | |
|  |  |  |  |  |  |  |
| **Setup** |  |  |  |  |  |  |
| Choose your favorite Database Management System (e.g. mySQL) | | | | | | |
| Create a database with several tables containing various table relations. | | | | | | |
| Insert a significant amount of data into the tables. | | | | | | |
|  |  |  |  |  |  |  |
| **Challenge** |  |  |  |  |  |  |
| Write queries involving multiple tables (i.e. using JOINs) to retrieve filtered rows and aggregated data from the tables. | | | | | | |
| Show how table design and predicate criteria can impact performance. | | | | | | |
| i.e. |  |  |  |  |  |  |
| Modify attributes so that query results are the same, but query performance is different. | | | | | | |
| Show timings comparing better/worse performance depending on implemention of different attributes. | | | | | | |
| Examples of attributes that affect query performance: | | | | | | |
| - Table definition (e.g. PK/FK, data types, column sizes) | | | | | | |
| - Query predicates (WHERE clause criteria) | | | | | | |
| - JOIN order | | | | | | |
| - Table distribution or indexing or partitioning (if the product allows) | | | | | | |

**Solution:**

We have created four tables EMPLOYEE, DEPARTMENTS, JOBS and JOB\_HISTORY to run different queries on individual tables and multiple tables. We are using oracle database to store data.

**Creation of Employees Table:**

CREATE TABLE "EMPLOYEES"

( "EMPLOYEE\_ID" NUMBER(6,0),

"FIRST\_NAME" VARCHAR2(20),

"LAST\_NAME" VARCHAR2(25) CONSTRAINT "EMP\_LAST\_NAME\_NN" NOT NULL ENABLE,

"EMAIL" VARCHAR2(25) CONSTRAINT "EMP\_EMAIL\_NN" NOT NULL ENABLE,

"PHONE\_NUMBER" VARCHAR2(20),

"HIRE\_DATE" DATE CONSTRAINT "EMP\_HIRE\_DATE\_NN" NOT NULL ENABLE,

"JOB\_ID" VARCHAR2(10) CONSTRAINT "EMP\_JOB\_NN" NOT NULL ENABLE,

"SALARY" NUMBER(8,2),

"COMMISSION\_PCT" NUMBER(2,2),

"MANAGER\_ID" NUMBER(6,0),

"DEPARTMENT\_ID" NUMBER(4,0),

CONSTRAINT "EMP\_SALARY\_MIN" CHECK (salary > 0) ENABLE,

CONSTRAINT "EMP\_EMAIL\_UK" UNIQUE ("EMAIL") ENABLE,

CONSTRAINT "EMP\_EMP\_ID\_PK" PRIMARY KEY ("EMPLOYEE\_ID") ENABLE,

CONSTRAINT "EMP\_DEPT\_FK" FOREIGN KEY ("DEPARTMENT\_ID")

REFERENCES "DEPARTMENTS" ("DEPARTMENT\_ID") ENABLE,

CONSTRAINT "EMP\_JOB\_FK" FOREIGN KEY ("JOB\_ID")

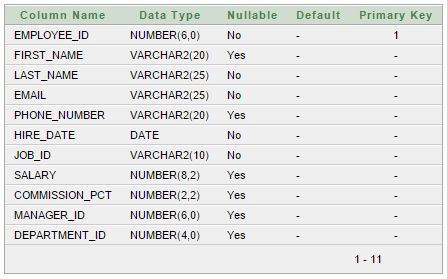
REFERENCES "JOBS" ("JOB\_ID") ENABLE,

CONSTRAINT "EMP\_MANAGER\_FK" FOREIGN KEY ("MANAGER\_ID")

REFERENCES "EMPLOYEES" ("EMPLOYEE\_ID") ENABLE

);

**Employees Schema:**



**Creation of Departments Table:**

CREATE TABLE "DEPARTMENTS"

( "DEPARTMENT\_ID" NUMBER(4,0),

"DEPARTMENT\_NAME" VARCHAR2(30) CONSTRAINT "DEPT\_NAME\_NN" NOT NULL ENABLE,

"MANAGER\_ID" NUMBER(6,0),

"LOCATION\_ID" NUMBER(4,0),

CONSTRAINT "DEPT\_ID\_PK" PRIMARY KEY ("DEPARTMENT\_ID") ENABLE,

CONSTRAINT "DEPT\_LOC\_FK" FOREIGN KEY ("LOCATION\_ID")

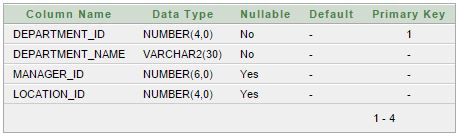
REFERENCES "LOCATIONS" ("LOCATION\_ID") ENABLE,

CONSTRAINT "DEPT\_MGR\_FK" FOREIGN KEY ("MANAGER\_ID")

REFERENCES "EMPLOYEES" ("EMPLOYEE\_ID") ENABLE

);

**Departments Schema:**



**Creation of Jobs Table:**

CREATE TABLE "JOBS"

( "JOB\_ID" VARCHAR2(10),

"JOB\_TITLE" VARCHAR2(35) CONSTRAINT "JOB\_TITLE\_NN" NOT NULL ENABLE,

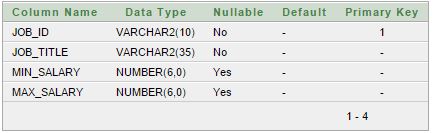
"MIN\_SALARY" NUMBER(6,0),

"MAX\_SALARY" NUMBER(6,0),

CONSTRAINT "JOB\_ID\_PK" PRIMARY KEY ("JOB\_ID") ENABLE

);

**Jobs Schema:**



**Creation of Job\_History Table:**

CREATE TABLE "JOB\_HISTORY"

( "EMPLOYEE\_ID" NUMBER(6,0) CONSTRAINT "JHIST\_EMPLOYEE\_NN" NOT NULL ENABLE,

"START\_DATE" DATE CONSTRAINT "JHIST\_START\_DATE\_NN" NOT NULL ENABLE,

"END\_DATE" DATE CONSTRAINT "JHIST\_END\_DATE\_NN" NOT NULL ENABLE,

"JOB\_ID" VARCHAR2(10) CONSTRAINT "JHIST\_JOB\_NN" NOT NULL ENABLE,

"DEPARTMENT\_ID" NUMBER(4,0),

CONSTRAINT "JHIST\_DATE\_INTERVAL" CHECK (end\_date > start\_date) ENABLE,

CONSTRAINT "JHIST\_EMP\_ID\_ST\_DATE\_PK" PRIMARY KEY ("EMPLOYEE\_ID", "START\_DATE") ENABLE,

CONSTRAINT "JHIST\_JOB\_FK" FOREIGN KEY ("JOB\_ID")

REFERENCES "JOBS" ("JOB\_ID") ENABLE,

CONSTRAINT "JHIST\_EMP\_FK" FOREIGN KEY ("EMPLOYEE\_ID")

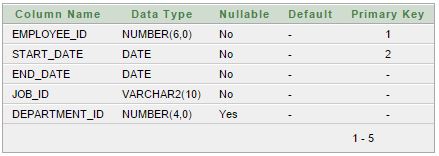
REFERENCES "EMPLOYEES" ("EMPLOYEE\_ID") ENABLE,

CONSTRAINT "JHIST\_DEPT\_FK" FOREIGN KEY ("DEPARTMENT\_ID")

REFERENCES "DEPARTMENTS" ("DEPARTMENT\_ID") ENABLE

);

**Jobs\_History Schema:**



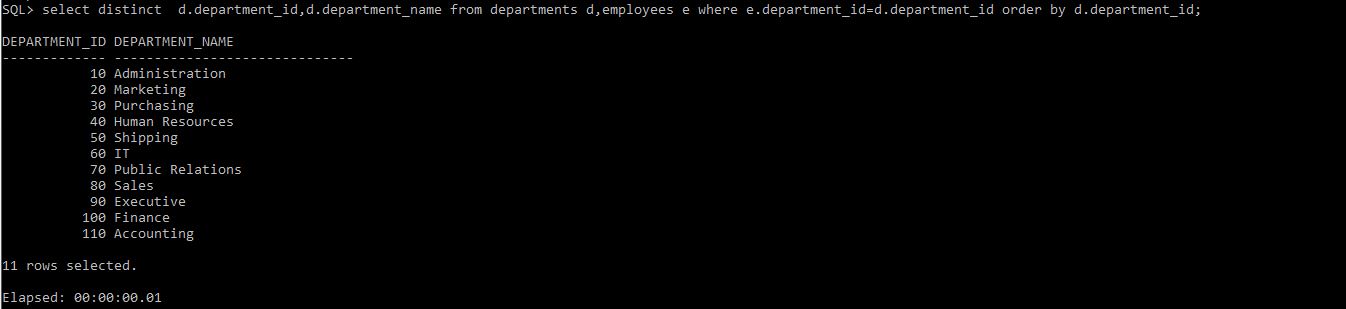
**Queries:**

Below are some of the normal and optimized queries that have different execution times

1. Get department id and name which contains at least one employee using join

**Normal query**

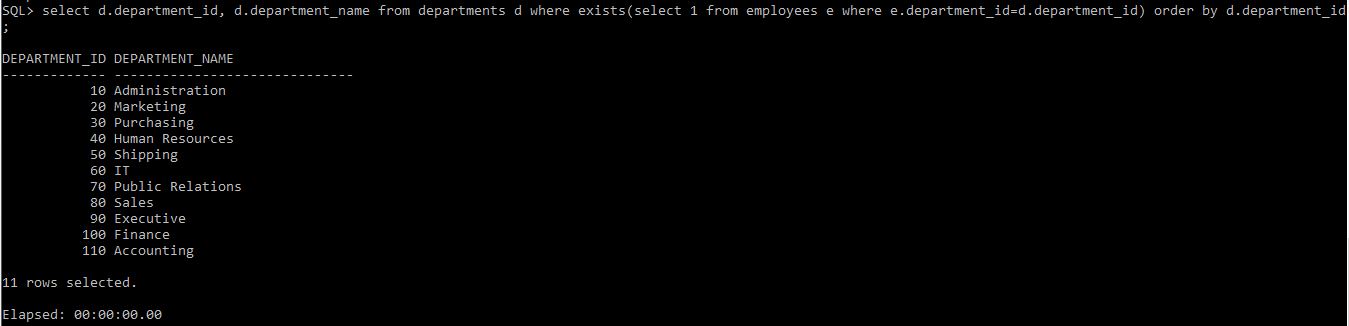
SELECT DISTINCT D.DEPARTMENT\_ID,D.DEPARTMENT\_NAME FROM DEPARTMENTS D,EMPLOYEES E WHERE E.DEPARTMENT\_ID = D.DEPARTMENT\_ID ORDER BY D.DEPARTMENT\_ID;



Above query takes 1ms to output result.

**Optimized query**

SELECT D.DEPARTMENT\_ID, D.DEPARTMENT\_NAME FROM DEPARTMENTS D WHERE EXISTS (SELECT 1 FROM EMPLOYEES E WHERE E.DEPARTMENT\_ID = D.DEPARTMENT\_ID) ORDER BY D.DEPARTMENT\_ID;

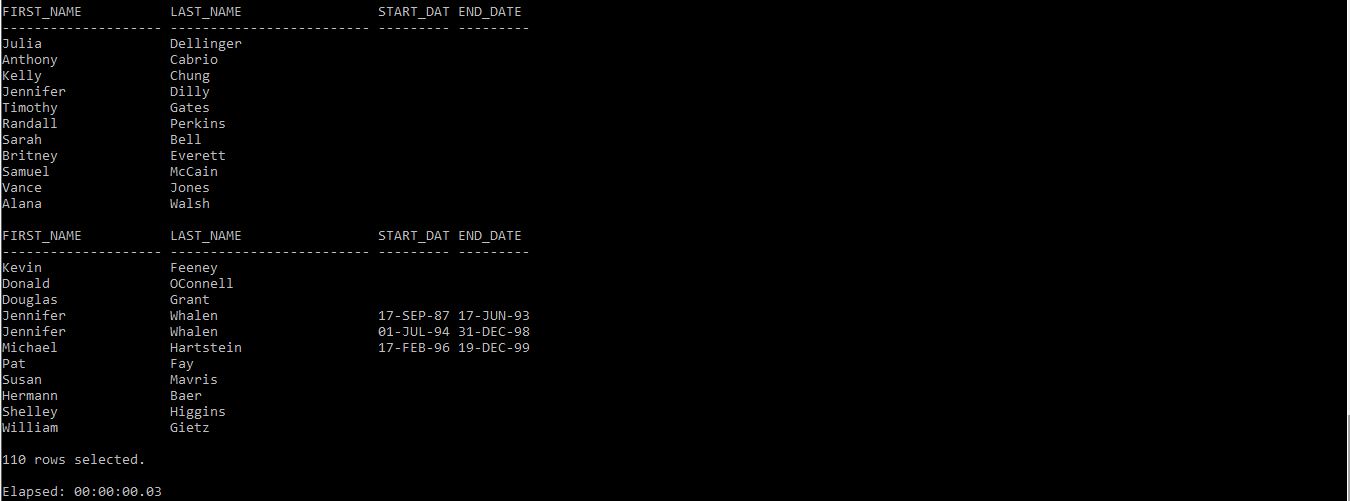


Above query takes 0ms to output result.

1. Optimizing by using different join orders:

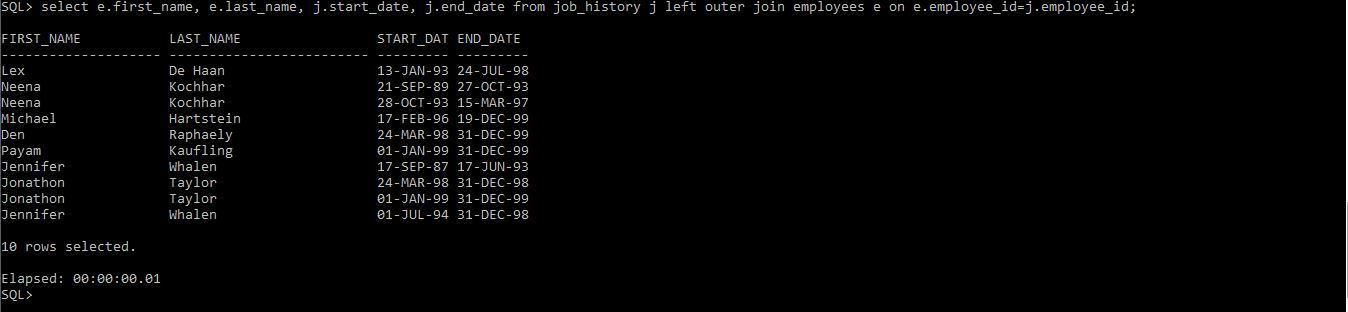
**Normal query**

SELECT E.FIRST\_NAME, E.LAST\_NAME, J.START\_DATE, J.END\_DATE FROM JOB\_HISTORY J LEFT OUTER JOIN EMPLOYEES E ON E.EMPLOYEE\_ID = J.EMPLOYEE\_ID;



**Optimized query:**

SELECT E.FIRST\_NAME, E.LAST\_NAME, J.START\_DATE, J.END\_DATE FROM EMPLOYEES E LEFT OUTER JOIN JOB\_HISTORY J ON E.EMPLOYEE\_ID = J.EMPLOYEE\_ID;



1. Executing all in one query:

**Normal query**

SELECT COUNT (\*) FROM employees WHERE salary < 5000;

SELECT COUNT (\*) FROM employees WHERE salary BETWEEN 5000 AND 15000;

SELECT COUNT (\*) FROM employees WHERE salary < 15000;



**Advanced query**

SELECT COUNT (CASE WHEN salary < 5000

THEN 1 ELSE null END) count1,

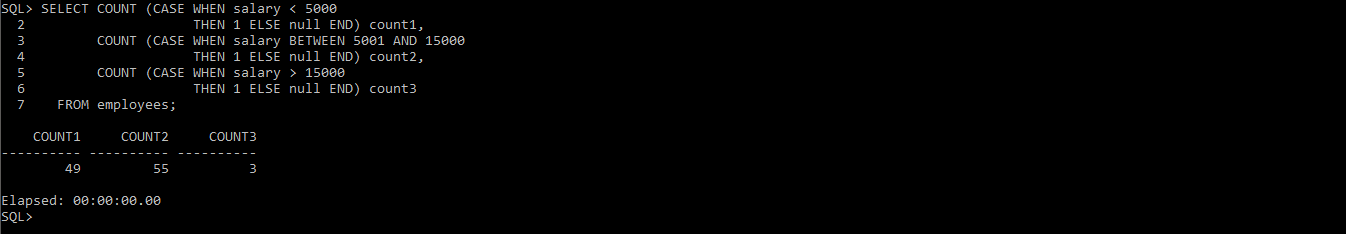
COUNT (CASE WHEN salary BETWEEN 5001 AND 15000

THEN 1 ELSE null END) count2,

COUNT (CASE WHEN salary > 15000

THEN 1 ELSE null END) count3

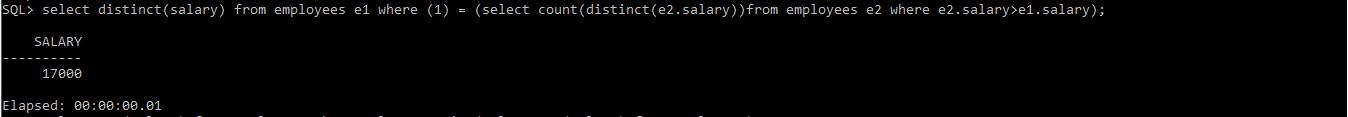
FROM employees;



1. Get second highest salary from employees table

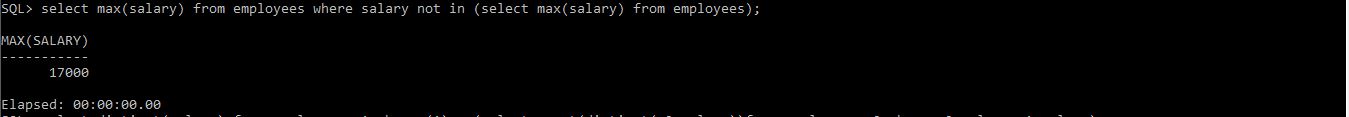
Normal query

SELECT DISTINCT(SALARY) FROM EMPLOYEES E1 WHERE (1) = (SELECT COUNT (DISTINCT(E2.SALARY))FROM EMPLOYEES E2 WHERE E2.SALARY > E1.SALARY);



Optimized query

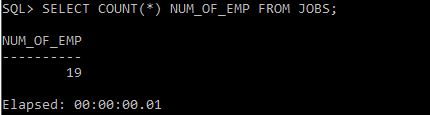
SELECT MAX(SALARY) FROM EMPLOYEES WHERE SALARY NOT IN (SELECT MAX(SALARY) FROM EMPLOYEES);



1. Aggregate Functions

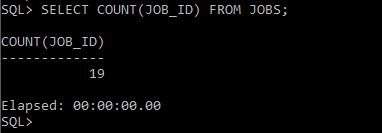
**Normal Query**

SELECT COUNT(\*) NUM\_OF\_EMP FROM JOBS;



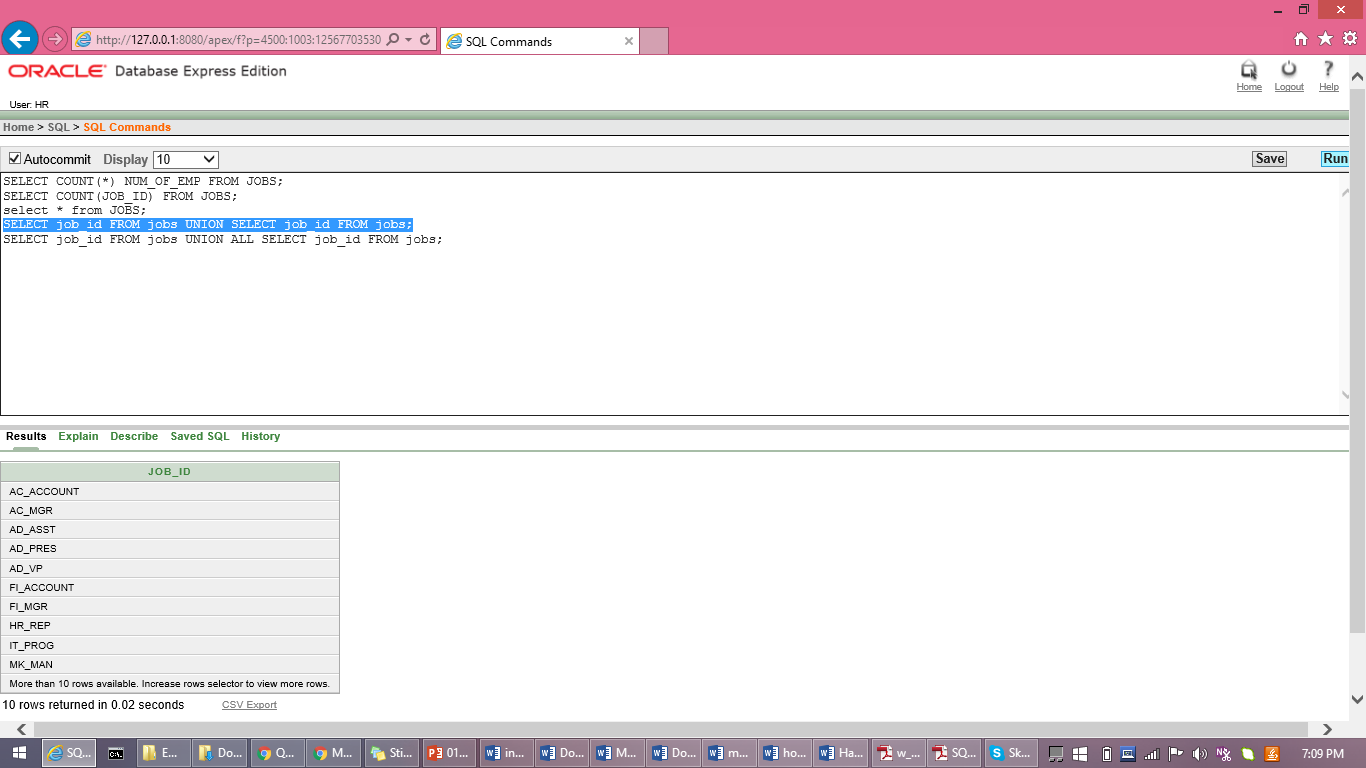
**Optimized Query**

SELECT COUNT(JOB\_ID) FROM JOBS;



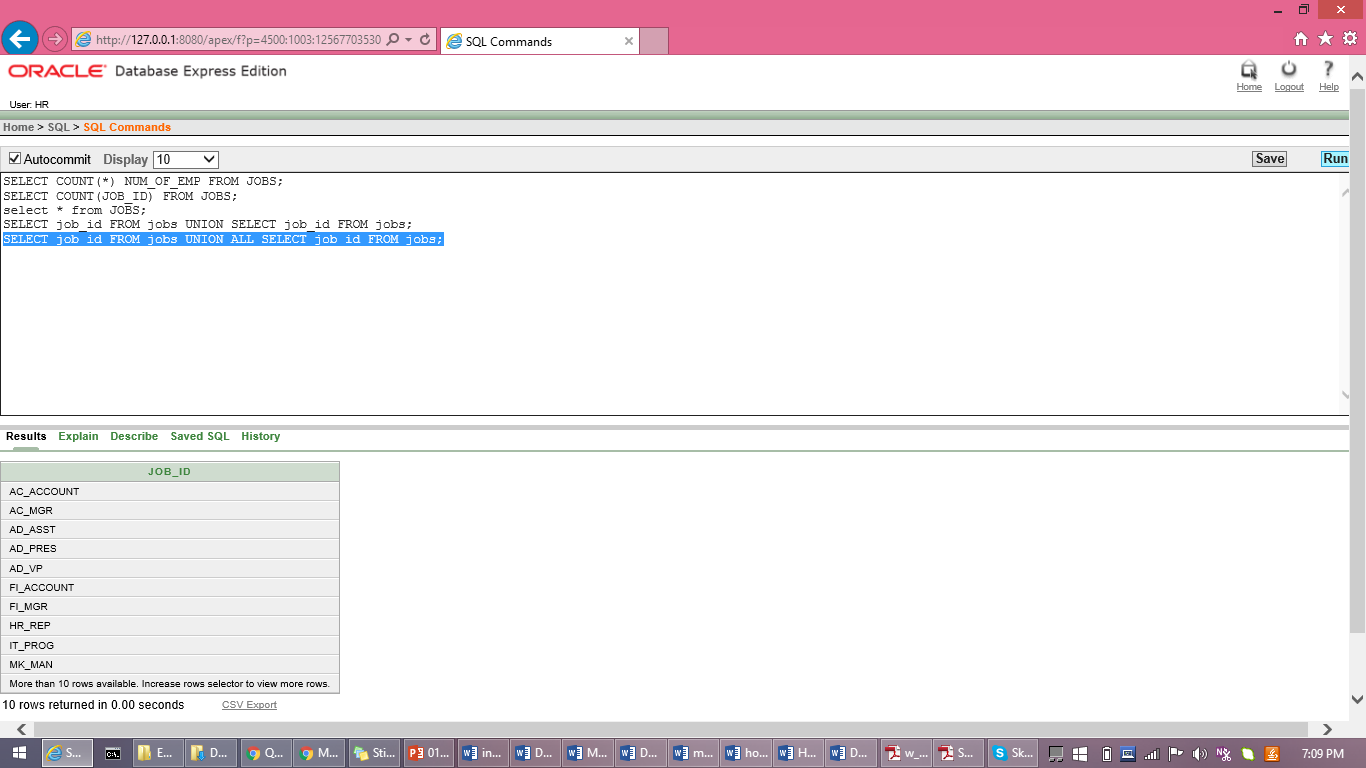
1. Unions example

SELECT job\_id FROM jobs UNION SELECT job\_id FROM jobs;



**Optimized query**

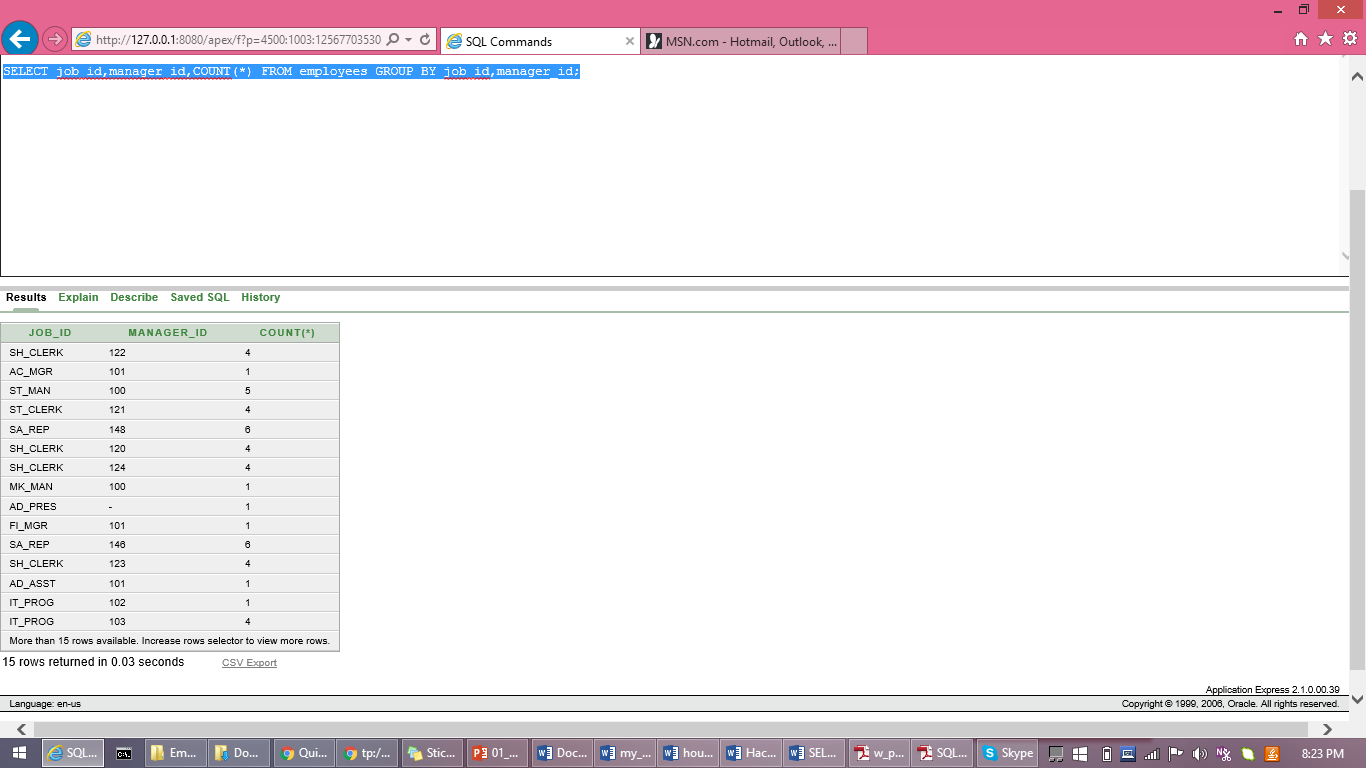
SELECT JOB\_ID FROM JOBS UNION ALL SELECT JOB\_ID FROM JOBS;



1. Group by command

**Normal query**

SELECT manager\_id,job\_id,COUNT(\*) FROM employees GROUP BY manager\_id,job\_id;



**Optimized query**

SELECT job\_id,manager\_id,COUNT(\*) FROM employees GROUP BY job\_id,manager\_id;

