Database Programming with SQL
kurs 2017 – database design and programming with sql
students slajdovi

9-1 Using Group By Having Clauses

- Za dobijanje srednje visine studenata: SELECT AVG(height) FROM students;
- Ali kako dobiti srednju visinu studenata po godini studija na kojoj su ?
  SELECT AVG(height) FROM students WHERE year_in_school = 10;
  itd za svaku godinu studija
- Jednostavnije bi bilo koristiti GROUP BY i HAVING izraze

GROUP BY Use

- GROUP BY se koristi za podelu redova u tabeli u manje grupe
- Onda se funkcije grupe koriste za vraćanje sumarne informacije za svaku od grupa

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
ORDER BY department_id;
```

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333333333333333
90	19333.3333333333333333
110	10150
-	7000

- U SELECT iskazu, redovi su grupisani sa department_id
- AVG funkcija se onda primenjuje na svaku grupu
- Primer: Kako naći max platu zaposlenih u svakom sektoru ?

- We use a GROUP BY clause stating which column to use to group the rows.

```
SELECT MAX(salary)
FROM employees
GROUP BY department_id;
```

DEPT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
...	...

MAX(SALARY)
7000
24000
13000
...

- Ali kako znati koja maksimalna plata pripada kojem sektoru ?

GROUP BY in SELECT

- Obično se uključuje GROUP BY kolona u SELECT listi

```
SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id;
```

DEPT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
...	...

DEPT_ID	MAX(SALARY)
-	7000
90	24000
20	13000
...	...

GROUP BY Clause

- Funkcije grupe zahtevaju da bilo koja kolona u listi u SELECT izrazu koja nije deo funkcije grupe mora biti izlistana u GROUP BY izrazu
- Šta je loše sa sledećim primerom ?

```
SELECT job_id, last_name, AVG(salary)
FROM employees
GROUP BY job_id;
```

ORA-00979: not a GROUP BY expression

- Job_id je dobro u SELECT listi, ali last_name nije pošto svaka jedinstvena grupa od job_id proizvodi samo jedan red izlaza (kao da je GROUP BY kolona). Ipak, može postojati mnogo različitih zaposlenih koji imaju isti job_id, npr postoje tri zaposlena sa job_id od SA_REP

COUNT

- Ovaj primer pokazuje koliko zemalja postoji u kojem regionu
- Zapamtiti da funkcije grupe ignorišu null vrednosti, tako da ako bilo koja zemlja nema ime, neće biti uključena u COUNT

```
SELECT COUNT(country_name), region_id
FROM wf_countries
GROUP BY region_id
ORDER BY region_id;
```

COUNT(COUNTRY_NAME)	REGION_ID
15	5
28	9
21	11
8	13
7	14
8	15
5	17
17	18

- Naravno, ovo nije uobičajeno, ali pri konstrukciji SQL iskaza moramo da razmišljamo o svim mogućnostima
- Bolje bi bilo koristiti COUNT(*)
- Ovime će se izbrojati svi redovi u svakoj grupi regiona, bez potrebe za proverom koja kolona sadrži NULL vrednosti

```
SELECT COUNT(*), region_id
FROM wf_countries
GROUP BY region_id
ORDER BY region_id;
```

WHERE Clause

- Takođe se može koristiti WHERE izraz za isključivanje redova pre nego što se preostali redovi formiraju po grupama

```
SELECT department_id, MAX(salary)
FROM employees
WHERE last_name != 'King'
GROUP BY department_id;
```

LAST_NAME	DEPT_ID	SALARY
King	90	24000
Kochhar	90	17000
De Haan	90	17000
Hunold	60	9000
Ernst	60	6000
Lorentz	60	4200
...	...	...

DEPT_ID	MAX(SALARY)
-	7000
90	17000
20	13000
...	...

- Zaposleni King je isključen sa WHERE izrazom, MAX(salary) za sektor 90 je 17000

More GROUP BY Examples

- Pokazati srednju vrednost populacije u svim zemljama u svakom regionu
- Zaokružiti srednju vrednost na ceo broj

```
SELECT region_id, ROUND(AVG(population)) AS population
FROM wf_countries
GROUP BY region_id
ORDER BY region_id;
```

- Count the number of spoken languages for all countries.

```
SELECT country_id, COUNT(language_id) AS "Number of languages"
FROM wf_spoken_languages
GROUP BY country_id;
```

GROUP BY Guidelines

- Važne smernice u korišćenju GROUP BY su:
  - ako se uključi funkcija grupe (AVG, SUM, COUNT, MAX, MIN, STDDEV, VARIANCE) u SELECT izrazu zajedno sa bilo kojom individualnom kolonom, svaka individualna kolona mora takođe da se pojavi u GROUP BY izrazu
  - nemože se koristiti alijas kolone u GROUP BY izrazu
  - WHERE izraz isključuje redove pre nego su podeljeni u grupe

Groups Within GROUPS

- Ponekad treba podeliti grupe u manje grupe
- Npr, ako treba grupisati sve zaposlene po sektorima a zatim unutar svakog sektora grupisati ih po poslovima
- Sledeći primer pokazuje koliko zaposlenih rade svaki posao unutar svakog sektora

```
SELECT department_id, job_id, count(*)
FROM employees
WHERE department_id > 40
GROUP BY department_id, job_id;
```

DEPT_ID	JOB_ID	COUNT(*)
110	AC_ACCOUNT	1
50	ST_CLERK	4
80	SA_REP	2
90	AD_VP	2
50	ST_MAN	1
...	...	...

Nesting Group Functions

- Funkcije grupe se mogu nestovati u dubinu do dva kada se koristi GROUP BY

```
SELECT max(avg(salary))
FROM employees
GROUP by department_id;
```

- Koliko vrednosti se može vratiti sa ovim upitom ? Odgovor je jedan, upit će naći srednju vrednost za platu za svaki sektor, a onda iz te liste izabrati jednu najveću vrednost

HAVING

- Ako je potrebno pronaći maksimum plate u svakom sektoru, ali samo za one sektore koji imaju više od jednog zaposlenog
- Zašto ovaj primer nije dobar:

```
SELECT department_id, MAX(salary)
FROM employees
WHERE COUNT(*) > 1
GROUP BY department_id;
```

ORA-00934: group function is not allowed here

- WHERE izraz se može koristiti samo za uključenje/isključenje individualnih redova a ne grupe redova. Zato se ne može koristiti funkcija grupe u WHERE izrazu
- Na isti način kako se koristio WHERE izraz za restrikciju redova koji su izabrani, može se koristiti HAVING izraz za restrikciju grupa
- U upitu koji koristi GROUP BY i HAVING iskaz, redovi su prvo grupisani, funkcije grupe su primenjene a onda samo one grupe koje se podudaraju sa HAVING izrazom su prikazane
- WHERE izraz se koristi za restrikciju redova; HAVING izraz se koristi za restrikciju grupa vraćenih iz GROUP BY izraza

```
SELECT department_id,MAX(salary)
FROM employees
GROUP BY department_id
HAVING COUNT(*)>1
ORDER BY department_id;
```

DEPARTMENT_ID	MAX(SALARY)
20	13000
50	5800
60	9000
80	11000
90	24000
110	12000

- Prvi upit nalazi MAX platu za svaki sektor u tabeli employee. HAVING izraz onda ograničava grupe vraćene na one sektore koji imaju više od jednog zaposlenog
- Sledeći upit nalazi srednju vrednost populacije zemalja u svakom regionu
- Zatim vraća samo grupe regiona sa najmanjom populacijom većom od 300000

```
SELECT region_id,
  ROUND(AVG(population))
FROM wf_countries
GROUP BY region_id
HAVING MIN(population)>300000
ORDER BY region_id;
```

REGION_ID	ROUND(AVG(POPULATION))
14	27037687
17	18729285
30	193332379
34	173268273
143	12023602
145	8522790
151	28343051

- HAVING i GROUP BY izrazi mogu koristiti različite kolone. Primer grupiše po region_id ali HAVING izraz ograničava grupe bazirano na populaciji
- Iako je HAVING izraz pre GROUP BY izraza u SELECT iskazu, preporuka je da se svaki iskaz piše po prikazanom redosledu (ORDER BY je uvek poslednji)

```
SELECT column, group_function
FROM table
WHERE
GROUP BY
HAVING
ORDER BY
```

9-2 Using Rollup and Cube Operations, and Grouping Sets

- Šta ako, pošto su grupe selektovane i izračunati svi agregati preko ovih grupa, treba takođe izračunati subtotals po grupi i grand total od svih izabranih redova
- Mogli bi se importovati rezultati u tabelearnu aplikaciju, koristiti kalkulator ili manuelno izračunati sume
- Ali još bolje je koristiti ekstenzije za GROUP BY izraze posebno kreirane za ove svrhe: ROLLUP, CUBE i GROUPING SETS

ROLLUP

- U GROUP BY upitima često se zahteva dobijanje međuzbirova i zbirova, a ROLLUP upravo to i radi; ROLLUP kreira međuzbirove između različitih nivoa sve do konačnog zbira korišćenjem liste za grupisanje specifirane u GROUP BY izrazu
- ROLLUP koristi redoslednu listu grupisanih kolona kao svoju listu argumenata
- Prvo, izračunava standardnu agregatnu vrednost specificiranu u GROUP BY izrazu
- Dalje, kreira progresivno međuzirove višeg nivoa, sa desna na levo kroz listu grupisanih kolona
- Na kraju kreira konačni zbir

ROLLUP Result Table

- U rezultantnoj tabeli, naglašeni su redovi generisani od strane ROLLUP operacije:

```
SELECT department_id, job_id, SUM(salary)
FROM employees
WHERE department_id < 50
GROUP BY ROLLUP (department_id, job_id);
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)	
10	AD_ASST	4400	
10	-	4400	← Subtotal for dept_id 10
20	MK_MAN	13000	
20	MK_REP	6000	
20	-	19000	← Subtotal for dept_id 20
-	-	23400	← Grand Total for report

ROLLUP Result Formula

- Broj kolona ili izraza koji se pojavljuju u ROLLUP listi argumenata određuje broj grupa
- Formula je (broj kolona) + 1 gde je broj kolona u ROLLUP argument listi
- Npr, dve kolone su na listi u ROOLUP argument liste i zato postojaće tri vrednosti generisane automatski

```
SELECT department_id, job_id, SUM(salary)
FROM employees
WHERE department_id < 50
GROUP BY ROLLUP (department_id, job_id);
```

Without ROLLUP

- Ako se koristi GROUP BY bez ROLLUP za isti uipt:

```
SELECT department_id, job_id, SUM(salary)
FROM   employees
WHERE department_id < 50
GROUP BY (department_id, job_id);
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
20	MK_MAN	13000
10	AD_ASST	4400
20	MK_REP	6000

- Moralo bi da se izvrši više upita za dobijanje međuzbirova nastalih od ROLLUP

CUBE

- CUBE, poput ROLLUP, je ekstenzija GROUP BY izraza i pravi cross-tabulation izveštaje
- Može da se primeni na sve agregatne funkcije (AVG, SUM, MIN, MAX, COUNT)

- Kolone izlistane sa GROUP BY izrazom su cross-refernced za kreiranje superseta od grupa
- Agregatne funkcije specificirane u SELECT listi su primenjene na ove grupe za kreiranje sume za dodatne super-agregatne redove
- Svaka moguća kombinacija redova je agregatna sa CUBE
- Ako postoji n kolona u GROUP BY izrazu, postojaće 2n mogućih super-agregatnih kombinacija
- Matematički, ove kombinacije formiraju n-dimenzionu kocku
- CUBE se koristi u upitima koji koriste kolone iz odvojenih tabela pre nego odvojene kolone iz jedne tabele
- Npr, korisnik pravi upit nad Sales tabelom za kompaniju kao što je AMAZON.COM
- Obično zahtevan cross-tabulation izveštaj može uključiti međuzbirove za sve moguće kombinacije prodaja za Month, Region i Product
- U sledećem primeru redovi u crvenom su generisani od strane CUBE operacije:

```
SELECT department_id, job_id, SUM(salary)
FROM employees
WHERE department_id < 50
GROUP BY CUBE (department_id, job_id);
```

Total for report →
Subtotal for MK_MAN →
Subtotal for MK_REP →
Subtotal for AD_ASST →
Subtotal for dept 10 →
Subtotal for dept 20 →

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
-	-	23400
-	MK_MAN	13000
-	MK_REP	6000
-	AD_ASST	4400
10	-	4400
10	AD_ASST	4400
20	-	19000
20	MK_MAN	13000
20	MK_REP	6000

- Prethodni primer ROLLUP je generisao međuzbir za svaki sektor i Total za izveštaj. CUBE daje ovu informaciju, ali dodaje međuzbirove za svaki osao kroz sve sektore

GROUPING SETS

- GROUPING SETS je sledeća ekstenzija GRUOP BY izraza
- Koristi se za specificiranje višestrukog grupisanja podataka
- Daje funkcionalnost postojanju višestrukog GROUP BY izraza u jednom SELECT iskazu, što nije dozvoljeno u normalnoj sintaksi
- Ako treba videti podatke iz EMPLOYEES tabele grupisane sa (department_id, job_id, manager_id) ali takođe grupisane i od (department_id, manager_id) i grupisane od (job_id, manager_id) onda bi se morale napraviti tri različita SELECT iskaza gde bi različito bilo samo GROUP BY
- Za db, ovo znači dobijanje istih podataka u tri različita puta a to može biti zamorno i ovde se vidi da korišćenje GROUPING SETS je efikasno za pisanje kompleksnih izveštaja
- U sledećm iskazu, redovi posebno naznačeni u bojama su generisani od strane GROUPING SETS operacije:

```
SELECT department_id, job_id, manager_id, SUM(salary)
FROM employees
WHERE department_id < 50
GROUP BY GROUPING SETS
((job_id, manager_id), (department_id, job_id), (department_id, manager_id));
```

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
-	MK_MAN	100	13000
-	MK_MAN	201	6000
-	AD_ASST	101	4400
10	AD_ASST	-	4400
20	MK_MAN	-	13000
20	MK_REP	-	6000
10	-	101	19000
20	-	100	13000
20	-	201	6000

GROUPING Functions

- Kada se koriste ROLLUP ili CUBE za kreiranje izveštaja sa međuzbirovima, mora se znati koji redovi na izlazu su zapravo redovi vraćeni iz db a koji su izračunati međurezultati korišćenjem ROLLUP ili CUBE operacija
- Ako se posmatra izveštaj sa desne strane, da li je to moguće razlikovati ?
- Da li je moguće razlikovati smeštenu NULL vrednost vraćenu od strane upita i NULL vrednost kreiranu od strane ROLLUP ili CUBE?

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
-	-	23400
-	MK_MAN	13000
-	MK_REP	6000
-	AD_ASST	4400
10	-	4400
10	AD_ASST	4400
20	-	19000
20	MK_MAN	13000
20	MK_REP	6000

- Za rešavanje ovih problema se koriste GROUPING funkcije
- Korišćenjem jedne kolone iz upita kao njenog argumenta, funkcija GROUPING će vratiti 1 kao agrregated red a 0 za non-aggregated red
- Sintaksa je : GROUPING(column_name)
- Koristi se samo u SELECT izrazu i uzima samo jedan izraz kolonu kao argument
- Primer:

```
SELECT department_id, job_id, SUM(salary),
   GROUPING(department_id) AS "Dept sub total",
   GROUPING(job_id) AS "Job sub total"
FROM employees
WHERE department_id < 50
GROUP BY CUBE (department_id, job_id);
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)	Dept sub total	Job sub total
-	-	23400	1	1
-	MK_MAN	13000	1	0
-	MK_REP	6000	1	0
-	AD_ASST	4400	1	0
10	-	4400	0	1
10	AD_ASST	4400	0	0
20	-	19000	0	1
20	MK_MAN	13000	0	0
20	MK_REP	6000	0	0

9-3 Using Set Operators

- Set operatori se koriste za kombinovanje rezultata iz različitih SELECT iskaza u jedan rezultat na izlazu
- Ponekad je potreban jedan izlaz iz više od jedne tabele

- Ako bi se udružile tabele, redovi koji ispunjavaju join uslov se vrate, ali šta ako join vrati set rezultata koji ne ispunjavaju potrebe ? Za to se koriste SET operatori
- Oni mogu da vrate redove nađene sa više SELECT iskaza, redove koji su u jednoj tabeli a nisu u drugoj ili redove koji su zajednički za oba iskaza

Setting the Stage

- Da bi se objasnili SET operatori, sledeće dve liste će se koristiti:
  A = {1, 2, 3, 4, 5} B ={4, 5, 6, 7, 8}
- Ili kao dve tabele:

A

A_ID
1
2
3
4
5

B

B_ID
4
5
6
7
8

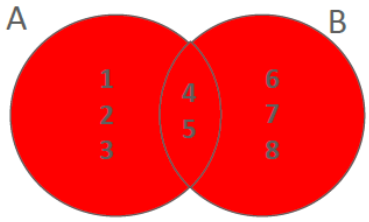
Rules to Remember

- Nekoliko pravila pri korišćenju SET operatora:
  - broj kolona i tipova podataka u kolonama mora biti identičan u svim SELECT iskazima korišćenim u upitu
  - imena kolona ne moraju biti ista
  - imena kolona na izlazu se uzimaju iz imena kolona u prvom SELECT iskazu
- Tako da bilo koji alijas kolone mora biti unet u prvom iskazu ako se želi videti na izveštaju

UNION

- Operator UNION vraća sve redove iz obe tabele pošto eleiminiše duplikate

```
SELECT a_id
FROM a
 UNION
SELECT b_id
FROM b;
```

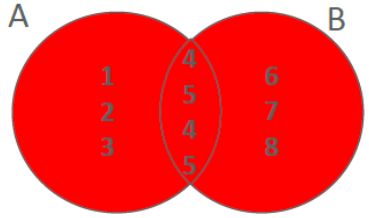


- Rezultat listanja svih elemenata iz A i B eliminisanjem duplikata je {1, 2, 3, 4, 5, 6, 7, 8}
- Ako se udruže A i B dobija se samo {4, 5}. Morala bi da se primeni full outer join za dobijanje istog rezultata kao taj

UNION ALL

- UNINON ALL operator vraća sve redove iz obe tabele, bez eliminisanja duplikata

```
SELECT a_id
FROM a
 UNION  ALL
SELECT b_id
FROM b;
```

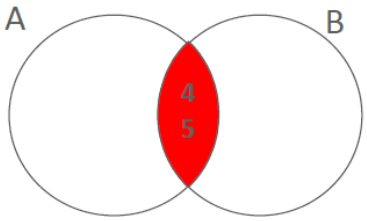


- Rezultat listanja svih elemenata iz A i B bez eliminisanja duplikata će biti {1, 2, 3, 4, 5, 4, 5, 6, 7, 8}

INTERSECT

- Operator INERSECT vraća sve redove zajedničke za obe tabele

```
SELECT a_id
FROM a
 INTERSECT
SELECT b_id
FROM b;
```

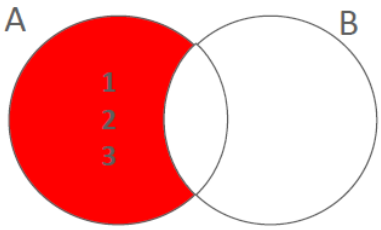


- Rezultat listanja svih elemenata nađenih istih za A i B je {4, 5}

MINUS

- Operator MINUS vraća sve redove nađene u jednoj tabeli ali ne i u drugoj

```
SELECT a_id
FROM a
 MINUS
SELECT b_id
FROM b;
```



- Rezultat listanja svi elemenata iz A ali ne i iz B je {1, 2, 3}
- Rezultat B MINUS A je {6, 7, 8}

Set Operator Examples

- Ponekad ako birate redove iz tabela koje nemaju zajedničke kolone, treba da kreirate sopstvene kolone da bi se poklopilo broj kolona u upitima
- Najjednostavniji način za tako nešto je uključiti jednu ili više NULL vrednost u SELECT listi
- Zapamtiti da treba dati svakoj odgovarajući alijas i odgovarajući tip podataka
- Npr:
  - employees tabela sadrži hire_date, employee_id i job_id
  - job_history tabela sadrži employee_id i job_id ali nema hire_date kolonu
  - dve tabele imaju employee_id i job_id zajedničko, ali job_history nema start_date
- Može se koristiti TO_CHAR(NULL) funkciju za kreiranje odgovarajuće kolone

```
SELECT hire_date, employee_id, job_id
FROM employees
 UNION
SELECT TO_DATE(NULL),employee_id, job_id
FROM job_history;
```

HIRE_DATE	EMPLOYEE_ID	JOB_ID
17-Jun-1987	100	AD_PRES
17-Sep-1987	200	AD_ASST
21-Sep-1989	101	AD_VP
03-Jan-1990	103	IT_PROG
21-May-1991	104	IT_PROG
13-Jan-1993	102	AD_VP
07-Jun-1994	205	AC_MGR
07-Jun-1994	206	AC_ACCOUNT
17-Oct-1995	141	ST_CLERK
17-Feb-1996	201	MK_MAN
11-May-1996	174	SA_REP
29-Jan-1997	142	ST_CLERK
17-Aug-1997	202	MK_REP
15-Mar-1998	143	ST_CLERK
24-Mar-1998	176	SA_REP
09-Jul-1998	144	ST_CLERK
07-Feb-1999	107	IT_PROG
24-May-1999	178	SA_REP
16-Nov-1999	124	ST_MAN
29-Jan-2000	149	SA_MAN
-	101	AC_ACCOUNT
-	101	AC_MGR
-	102	IT_PROG
-	114	ST_CLERK
-	122	ST_CLERK
-	176	SA_MAN
-	176	SA_REP
-	200	AC_ACCOUNT
-	200	AD_ASST
-	201	MK_REP

- Tabela job_history sadrži 10 redova, tako da kada se doda 20 redova iz employees tabele, 30 redova se vraćaju. Redovi sa NULL za hire_date su iz job_history tabele
- Službena reč NULL se može koristiti za uparenje kolona u SELECT listi
- Jedan NULL je uključen za svaku nedostajuću kolonu

- I dalje, NULL je formatiran da odgovara tipu podataka kolone na koju se odnosi, tako da TO_CHAR, TO_DATE ili TO_NUMBER funkcije se koriste za dobijanje identične SELECT liste

SET Operations ORDER BY

- Ako treba kontrolisati redosled vraćenih redova kada se koristi SET operatori u upitu, ORDER BY iskaz mora samo da se jednom koristi u poslednjem SELECT iskazu u upitu
- Korišćenjem primera prethodnog upita, moglo bi se ORDER BY employee_id da bi se videlo jobs koji ima svaki zaposleni

```
SELECT hire_date, employee_id, job_id
FROM employees
 UNION
SELECT TO_DATE(NULL),employee_id, job_id
FROM job_history
ORDER BY employee_id;
```

HIRE_DATE	EMPLOYEE_ID	JOB_ID
17-Jun-1987	100	AD_PRES
21-Sep-1989	101	AD_VP
-	101	AC_ACCOUNT
-	101	AC_MGR
13-Jan-1993	102	AD_VP
-	102	IT_PROG
03-Jan-1990	103	IT_PROG
21-May-1991	104	IT_PROG
07-Feb-1999	107	IT_PROG
-	114	ST_CLERK
...	...	...

- Može se poboljšati čitljivost izlaza uključivanjem start_date i end_date kolona iz job_history tabele; za to treba uporediti kolone u oba upita dodavanjem dve TO_DATE(NULL) kolone u prvom upitu

```
SELECT hire_date, employee_id, TO_DATE(null) start_date,
   TO_DATE(null) end_date, job_id, department_id
FROM   employees
 UNION
SELECT TO_DATE(null), employee_id, start_date, end_date, job_id,
   department_id
FROM job_history
ORDER BY employee_id;
```

- Pošto naslovi kolona za upit izlaz su uzeti iz prvog upita, dati su im alijasi istog imena kao što su uparene kolone u drugom upitu

HIRE_DATE	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
17-Jun-1987	100	-	-	AD_PRES	90
21-Sep-1989	101	-	-	AD_VP	90
-	101	21-Sep-1989	27-Oct-1993	AC_ACCOUNT	110
-	101	28-Oct-1993	15-Mar-1997	AC_MGR	110
13-Jan-1993	102	-	-	AD_VP	90
-	102	13-Jan-1993	24-Jul-1998	IT_PROG	60
03-Jan-1990	103	-	-	IT_PROG	60
21-May-1991	104	-	-	IT_PROG	60
07-Feb-1999	107	-	-	IT_PROG	60
-	114	24-Mar-1998	31-Dec-1999	ST_CLERK	50
-	122	01-Jan-1999	31-Dec-1999	ST_CLERK	50
16-Nov-1999	124	-	-	ST_MAN	50
17-Oct-1995	141	-	-	ST_CLERK	50
29-Jan-1997	142	-	-	ST_CLERK	50
15-Mar-1998	143	-	-	ST_CLERK	50
...	...	...	...	...	...