

# **CIS 353 Database**

## **Relational Model and Relational Algebra**

# The Relational Model

- Built around a single concept for modeling data: the relation or table.
  - A relational database is a collection of relations.
  - Each relation is a table with rows and columns.
- Supports high-level programming language (SQL).
  - Limited but very useful set of operations
- Most current DBMS are relational (Oracle, IBM DB2, MS SQL)

# The Relational Model

CoursesTaken

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

- Structure: Table (like an array of structs)
- Operations: Relational algebra (selection, projection, conditions, etc)
- Constraints: E.g., grades can be only {A, B, C, D, F}

# Relations

- A relation is a two-dimensional table:
  - Relation  $\Leftrightarrow$  table
  - Attribute  $\Leftrightarrow$  column name
  - Tuple  $\Leftrightarrow$  row (not the header row)
- Database  $\Leftrightarrow$  collection of relations
- A relation has two parts:
  - Schema defines column heads of the table (attributes)
  - Instance contains the data rows (tuples, rows, or records) of the table

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

CoursesTaken

# Schema

CoursesTaken

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

- The schema of a relation is the name of the relation followed by a parenthesized list of attributes

**CoursesTaken (Student, Course, Grade)**

- A design in a relational model consists of a set of schemas.
- Such a set of schemas is called a relational database schema.

# Relation and Schema

- Relation is a set of tuples
  - Order in which we present the tuples does not matter
- The attributes in a schema are also a set (not a list)
  - Schema is the same irrespective of order of attributes.

**CoursesTaken(Student, Grade, Course)**

- We specify a “standard” order when we introduce a schema

CoursesTaken

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

# Degree and Cardinality

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

- Degree is the number of fields/attributes in schema
  - 3 in the table above
- Cardinality is the number of tuples in relation
  - 4 in the table above

# Activity

Employee	ID	Name	SSN	Salary	Phone	Email
	101	John	AA	50000	12	j@sw
	102	Robin	BB	60000	13	r@yh
	103	Alya	CC	35000	14	a@hm
	104	Yusuf	DD	68000	15	y@ch
	105	John	EE	62000	89	j@in
	106	Raj	FF	45000	87	r@au
	107	Jayant	GG	25000	45	j@us
	108	John	HH	35000	15	j@de
	109	Neil	II	25000	12	n@uk



# Keys of Relations

---

# Keys of Relations

Employee	ID	Name	SSN	Salary	Phone	Email
	101	John	AA	50000	12	j@sw
	102	Robin	BB	60000	13	r@yh
	103	Alya	CC	35000	14	a@hm
	104	Yusuf	DD	68000	15	y@ch
	105	John	EE	62000	89	j@in
	106	Raj	FF	45000	87	r@au
	107	Jayant	GG	25000	45	j@us
	108	John	HH	35000	15	j@de
	109	Neil	II	25000	12	n@uk

# Keys of Relations

- Uniquely identifying each tuple
  - Super Keys
  - Candidate Keys
  - Primary Keys
  - Alternate Keys

Employee	ID	Name	SSN	Salary	Phone	Email
	101	John	AA	50000	12	j@sw
	102	Robin	BB	60000	13	r@yh
	103	Alya	CC	35000	14	a@hm
	104	Yusuf	DD	68000	15	y@ch
	105	John	EE	62000	89	j@in
	106	Raj	FF	45000	87	r@au
	107	Jayant	GG	25000	45	j@us
	108	John	HH	35000	15	j@de
	109	Neil	II	25000	12	n@uk

# Super Keys

- A super set of keys
- Uniquely identify a tuple
- May contain attributes with NULL value
- A relation can have many super keys
- Super keys may contain additional attributes

Employee	ID	Name	SSN	Salary	Phone	Email
	101	John	AA	50000	12	j@sw
	102	Robin	BB	60000	13	r@yh
	103	Alya	CC	35000	14	a@hm
	104	Yusuf	DD	68000	15	y@ch
	105	John	EE	62000	89	j@in
	106	Raj	FF	45000	87	r@au
	107	Jayant	GG	25000	45	j@us
	108	John	HH	35000	15	j@de
	109	Neil	II	25000	12	n@uk

# Super Keys

Superkeys:

{ID}, {SSN}, {ID, Name},  
{ID, SSN}, {ID, Phone},  
{Name, Phone}, {ID, Email},  
{Name, SSN, Phone},  
{Name, Email},  
{ID, SSN, Phone} . . . . .

Employee	ID	Name	SSN	Salary	Phone	Email
	101	John	AA	50000	12	j@sw
	102	Robin	BB	60000	13	r@yh
	103	Alya	CC	35000	14	a@hm
	104	Yusuf	DD	68000	15	y@ch
	105	John	EE	62000	89	j@in
	106	Raj	FF	45000	87	r@au
	107	Jayant	GG	25000	45	j@us
	108	John	HH	35000	15	j@de
	109	Neil	II	25000	12	n@uk

# Superkeys

- A **superkey** is defined as a **subset of attribute types** of a relation  $R$  with the property that no two tuples in any relation state should have the same combination of values for these attribute types
- A superkey specifies a **uniqueness constraint**
- A superkey can have redundant attribute types
  - Example: (ID, SSN, Phone)

# Candidate Keys

- Minimal super keys

Superkeys:

{ID}, {SSN}, {ID, Name},  
{ID, SSN}, {ID, Phone},  
{Name, Phone}, {ID, Email},  
{Name, SSN, Phone},  
{Name, Email},  
{ID, SSN, Phone} . . . . .

Employee	ID	Name	SSN	Salary	Phone	Email
	101	John	AA	50000	12	j@sw
	102	Robin	BB	60000	13	r@yh
	103	Alya	CC	35000	14	a@hm
	104	Yusuf	DD	68000	15	y@ch
	105	John	EE	62000	89	j@in
	106	Raj	FF	45000	87	r@au
	107	Jayant	GG	25000	45	j@us
	108	John	HH	35000	15	j@de
	109	Neil	II	25000	12	n@uk

# Candidate Keys

- Minimal super keys

Superkeys:

{ID}, {SSN}, {ID, Name},  
{ID, SSN}, {ID, Phone},  
{Name, Phone}, {ID, Email},  
{Name, SSN, Phone},  
{Name, Email}, {Email}  
{ID, SSN, Phone} . . . . .

Employee	ID	Name	SSN	Salary	Phone	Email
	101	John	AA	50000	12	j@sw
	102	Robin	BB	60000	13	r@yh
	103	Alya	CC	35000	14	a@hm
	104	Yusuf	DD	68000	15	y@ch
	105	John	EE	62000	89	j@in
	106	Raj	FF	45000	87	r@au
	107	Jayant	GG	25000	45	j@us
	108	John	HH	35000	15	j@de
	109	Neil	II	25000	12	n@uk

Candidate Keys:

{ID}, {SSN}, {Name, Phone},  
{Email}



# (Candidate) Keys

- A **key**  $K$  of a relation scheme  $R$  is a **superkey** of  $R$  with the additional property that removing any attribute type from  $K$  leaves a set of attribute types that is no superkey of  $R$
- A **key** does not have any redundant attribute types
- The **key** constraint states that every relation must have at least one key that allows uniquely identifying its tuples
- All super keys can't be candidate keys. All candidate keys are super keys

# Primary Keys

- Candidate key with no NULL value
- Generally, never changed

Employee	ID	Name	SSN	Salary	Phone	Email
	101	John	AA	50000	12	j@sw
	102	Robin	BB	60000	13	r@yh
	103	Alya	CC	35000	14	a@hm
	104	Yusuf	DD	68000	15	y@ch
	105	John	EE	62000	89	j@in
	106	Raj	FF	45000	87	r@au
	107	Jayant	GG	25000	45	j@us
	108	John	HH	35000	15	j@de
	109	Neil	II	25000	12	n@uk

# Alternate Keys

- Candidate keys except the primary key

Employee	ID	Name	SSN	Salary	Phone	Email
	101	John	AA	50000	12	j@sw
	102	Robin	BB	60000	13	r@yh
	103	Alya	CC	35000	14	a@hm
	104	Yusuf	DD	68000	15	y@ch
	105	John	EE	62000	89	j@in
	106	Raj	FF	45000	87	r@au
	107	Jayant	GG	25000	45	j@us
	108	John	HH	35000	15	j@de
	109	Neil	II	25000	12	n@uk

# Primary Keys, and Alternative Keys

- The **primary key** is used to identify tuples in the relation, to establish connections to other relations, and for storage purposes
  - Entity integrity constraint: attribute types that make up the primary key should always satisfy a NOT NULL constraint
- Only one Candidate Key can be Primary Key
- Other candidate keys are then referred to as **alternate keys**

# Foreign Keys: Referential Integrity

Student

S_ID	Name	Dept_Code	Credits
101	John	101	12
102	Robin	102	14
103	Alya	103	20
104	Yusuf	104	10

Dept

Dept_Code	Dept_Name
101	CSE
102	EEE
103	ECE
104	MECH

# Foreign Keys

- A **set of attribute types** FK in a relation  $R_1$  is a **foreign key** of  $R_1$  if two conditions are satisfied (referential integrity constraint)
  - The attribute types in FK have the same domains as the primary key attribute types PK of a relation  $R_2$
  - A value FK in a tuple  $t_1$  of the current state  $r_1$  either occurs as a value of PK for some tuple  $t_2$  in the current state  $r_2$  or is NULL

# Relational Constraints

<b>Domain constraint</b>	The value of each attribute type A must be an atomic and single value from the domain.
<b>Key constraint</b>	Every relation has a key that allows uniquely identifying its tuples.
<b>Entity integrity constraint</b>	The attribute types that make up the primary key should always satisfy a NOT NULL constraint.
<b>Referential integrity constraint</b>	A foreign key FK has the same domain as the primary key PK attribute type(s) it refers to and either occurs as a value of PK or NULL.

# Example Relational Data Model

**SUPPLIER**(SUPNR:integer, SUPNAME:string, SUPADDRESS:string,  
SUPCITY:string, SUPSTATUS:integer)

**PRODUCT**(PRODNR:integer, PRODNAME:string, PRODTYPE:string, AVAILABLE  
QUANTITY:integer)

**SUPPLIES**(SUPNR, PRODNR:integer, PURCHASE\_PRICE:real,  
DELIV\_PERIOD:integer)

**PURCHASE\_ORDER**(PONR:integer, PODATE:date, *SUPNR:integer*)

**PO\_LINE**(*PONR:integer*, *PRODNR:integer*, QUANTITY:integer)



# Example Relational Data Model

Supplier

SUPNR	SUPNAME	SUPADDRESS	SUPCITY	SUPSTATUS
21	Deliwines	240, Avenue of the Americas	New York	20
32	Best Wines	660, Market Street	San Francisco	90
...				

Product

PRODNR	PRODNAME	PRODTYPE	AVAILABLE_QUANTITY
0119	Chateau Miraval, Cotes de Provence Rose, 2015	rose	126
0384	Dominio de Pingus, Ribera del Duero, Tempranillo, 2006	red	38
...			

Supplies

SUPNR	PRODNR	PURCHASE_PRICE	DELIV_PERIOD
21	0119	15.99	1
21	0384	55.00	2
...			

Purchase\_Order

PONR	PODATE	SUPNR
1511	2015-03-24	37
1512	2015-04-10	94
...		

PO\_Line

PONR	PRODNR	QUANTITY
1511	0212	2
1511	0345	4
...		

# Recap

sid	fname	lname	login	age	GPA
C123456	John	Smith	smithx02@ece.abc.edu	19	3.3
C234561	Karen	Johns	johnsx05@cs.abc.edu	18	3.0
C345612	Mary	Anderson	anderx10@math.abc.edu	20	3.5
C456123	Helen	Henderson	hendex05@ece.abc.edu	18	2.9
C561234	John	Smith	smithx99@cs.abc.edu	19	3.8
C612345	Aidan	Cocke	cockex35@ece.abc.edu	18	3.3

# Relational Query Languages

Query languages: Allow manipulation and retrieval of data from a database.

Relational model supports simple, powerful QLs:

- Strong formal foundation based on logic.
- Allows for optimization.

Query Languages != programming languages!

- QLs not intended to be used for complex calculations.
- QLs support easy, efficient access to large data sets.

# Formal Relational Query Languages

Two mathematical Query Languages form the basis for “real” languages (e.g. SQL), and for implementation:

- Relational Algebra: More operational (imperative), very useful for representing execution plans. (a procedural programming language)
- Relational Calculus: Lets users describe what they want, rather than how to compute it. (Non-operational, declarative, basis for SQL.)

# Relational Algebra

- Operator takes in a relation and output a different relation
- Pure relational algebra has set semantics
  - No duplicate tuples in a relation instance
- Basic operators:
  - **Selection** (  $\sigma$  ) Selects a subset of rows from relation.
  - **Projection** (  $\pi$  ) Deletes unwanted columns from relation.
  - **Cross-product** (  $\times$  ) Allows us to combine two relations.
  - **Set-difference** (  $-$  ) Tuples in reln. 1, but not in reln. 2.
  - **Union** (  $\cup$  ) Tuples in reln. 1 and in reln. 2.
- Additional operators:
  - **Intersection, join, renaming**: Not essential, but (very!) useful.
- Since each operator returns a relation, operators can be composed!

# Relational Algebra Operators: Unary

- Unary Operators: on **single relation**
- **Projection** ( $\pi$ ): Retains only desired columns (vertical)
- **Selection** ( $\sigma$ ): Selects a subset of rows (horizontal)
- **Renaming** ( $\rho$ ): Rename attributes and relations.

# Projection ( $\pi$ )

- Discards attributes that are not in *projection list*.
- *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- No duplicates in result!
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)

# Projection ( $\pi$ )

**Sailor (sid, sname, rating, age)**

- **Show the names and ratings of all the sailors**



# Projection ( $\pi$ )

**Sailor (sid, sname, rating, age)**

- **Show the names and ratings of all the sailors**

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

# Projection ( $\pi$ )

**Sailor (sid, sname, rating, age)**

- Show the names and ratings of all the sailors

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

*Sailor*

$\pi_{sname, rating}$  (Sailor)



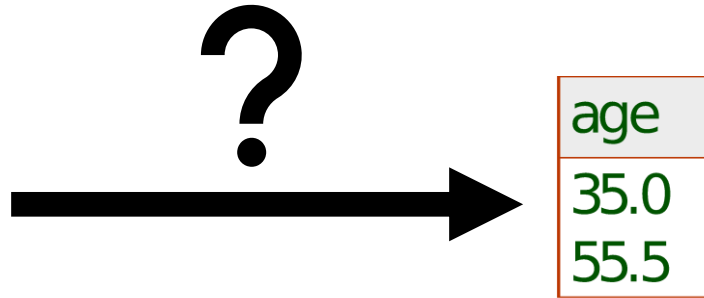
sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

# Projection ( $\pi$ )

**Sailor (sid, sname, rating, age)**

- List the age of all the sailors

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0



# Selection ( $\sigma$ )

- Selects rows that satisfy *selection condition*.
- *Schema* of result identical to schema of (only) input relation.
- Selects a subset of rows (horizontal) *Result* relation can be the *input* for another relational algebra operation!  
(*Operator composition.*)

# Selection ( $\sigma$ )

**Sailor (sid, sname, rating, age)**

- **List the tuples where sailor rating is more than 8**

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

# Selection ( $\sigma$ )

**Sailor (sid, sname, rating, age)**

- List the tuples where sailor rating is more than 8

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

*Sailor*

$\sigma_{rating > 8}$  (Sailor)




sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

# Composing Select and Project

- Which sailors have a rating more than 8

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0


$$\pi_{sname, rating}(\sigma_{rating > 8}(S2))$$

sname	rating
yuppy	9
rusty	10

What about:

$$\sigma_{rating > 8}(\pi_{sname}(S1))$$

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

S1

# Relational Algebra Operators: Binary

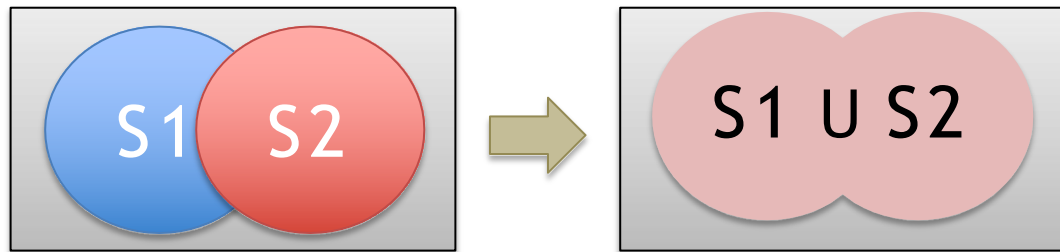
- Binary Operators: on **pairs of relations**
- **Union** ( $\cup$ ): Tuples in  $r_1$  or in  $r_2$ .
- **Set-difference** ( $-$ ): Tuples in  $r_1$ , but not in  $r_2$ .
- **Cross-product** ( $\times$ ): Allows us to combine two relations.



# Union (U)

All of these operations take two input relations, which must be union-compatible:

- Same number of fields.
- ‘Corresponding’ fields have the same type.



# Union (U)

Relational Instance **S1**

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Relational Instance **S2**

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0



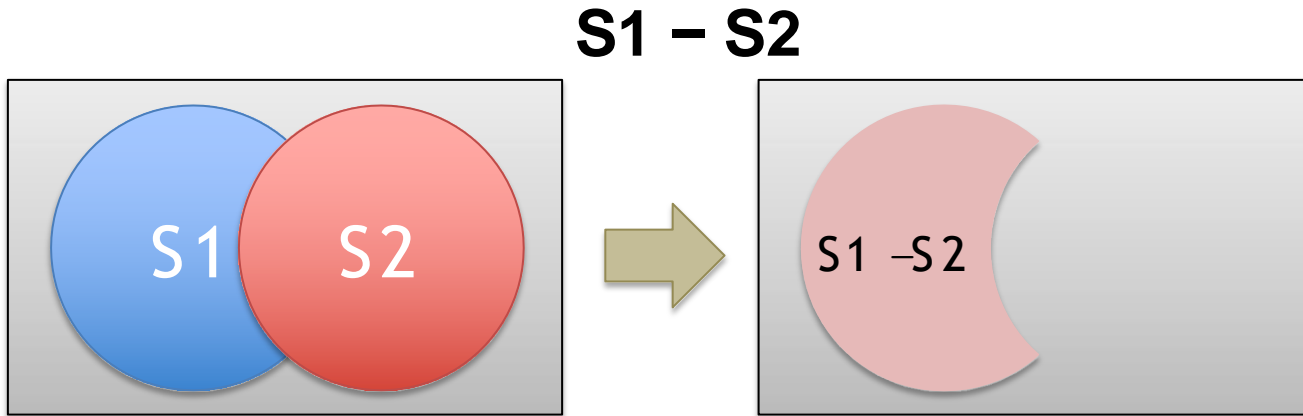
**S1 U S2**

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
28	yuppy	9	35.0
44	guppy	5	35.0

# Set Difference ( - )

Same as with union, both input relations must be *compatible*.

SQL Expression: EXCEPT



# Set Difference ( – ), cont.

Relational *Instance* **S1**

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Relational *Instance* **S2**

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

**S1 – S2**

<u>sid</u>	sname	rating	age
22	dustin	7	45

**S2 – S1**

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
44	guppy	5	35.0

# Cross-Product ( $\times$ )

- **$R1 \times S1$** : Each row of **R1** paired with each row of **S1**
- Result schema has one field per field of S1 and R1, with field names 'inherited' if possible.
- Conflict: Both S1 and R1 have a field called sid.

How many rows in result?  $|R1| \times |S1|$

Schema compatibility? Not needed.

Duplicates? None generated.

# Cross-Product (×)

R1:

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

×

S1:

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

sid	bid	day	sid	sname	rating	age
22	101	10/10/96	22	dustin	7	45.0
22	101	10/10/96	31	lubber	8	55.5
22	101	10/10/96	58	rusty	10	35.0

=

sid	bid	day	sid	sname	rating	age
58	103	11/12/96	22	dustin	7	45.0
58	103	11/12/96	31	lubber	8	55.5
58	103	11/12/96	58	rusty	10	35.0

sid	bid	day	sid	sname	rating	age
22	101	10/10/96	22	dustin	7	45.0
22	101	10/10/96	31	lubber	8	55.5
22	101	10/10/96	58	rusty	10	35.0
58	103	11/12/96	22	dustin	7	45.0
58	103	11/12/96	31	lubber	8	55.5
58	103	11/12/96	58	rusty	10	35.0

# Renaming ( $\rho = \text{“rho”}$ )

- *Renames relations and their attributes:*
- Note that relational algebra doesn't require names.
  - We could just use positional arguments.

$\rho(\text{Temp1}(1 \rightarrow \text{sid1}, 4 \rightarrow \text{sid2}), R1 \times S1)$

Output  
Relation  
Name

Renaming List  
position  $\rightarrow$  New Name

Input  
Relation

$R1 \times S1$

sid	bid	day	sid	sname	rating	age
22	101	10/10/96	22	dustin	7	45.0
22	101	10/10/96	31	lubber	8	55.5
22	101	10/10/96	58	rusty	10	35.0
58	103	11/12/96	22	dustin	7	45.0
58	103	11/12/96	31	lubber	8	55.5
58	103	11/12/96	58	rusty	10	35.0



Temp1

sid1	bid	day	sid2	sname	rating	age
22	101	10/10/96	22	dustin	7	45.0
22	101	10/10/96	31	lubber	8	55.5
22	101	10/10/96	58	rusty	10	35.0
58	103	11/12/96	22	dustin	7	45.0
58	103	11/12/96	31	lubber	8	55.5
58	103	11/12/96	58	rusty	10	35.0

1. What are the names of employees who have a salary of more than 7000?
2. Find the employee ids whose first name is James, have a salary less than 10000 and are from department number 40

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOINING_DATE	JOB_ID	SALARY	DEPARTMENT_ID
100	Gerald	Cambrault	34675	AD_PRES	5500	10
101	Renske	Ladwig	34837	AD_VP	15000	20
102	Janette	King	35230	AD_VP	7000	20
103	Sarath	Sewall	35477	IT_PROG	12000	30
104	William	Gietz	35627	IT_PROG	5100	30
105	Jennifer	Whalen	35662	IT_PROG	4900	30
106	Britney	Everett	35733	IT_PROG	5800	30
107	Anthony	Cabrio	35788	IT_PROG	5600	30
108	Alexis	Bull	35861	FI_MGR	7500	40
109	Adam	Fripp	36033	FI_ACCOUNT	8000	40
110	James	Marlow	36066	FI_ACCOUNT	9000	50
111	James	Landry	36174	FI_ACCOUNT	8500	50
112	Payam	Kaufling	36260	FI_ACCOUNT	9500	50
113	Shelley	Higgins	36480	FI_ACCOUNT	8500	50
114	Shanta	Vollman	36501	PU_MAN	10500	50
115	Irene	Mikkilineni	36506	PU_CLERK	10000	50
116	Mozhe	Atkinson	36593	PU_CLERK	9500	50