

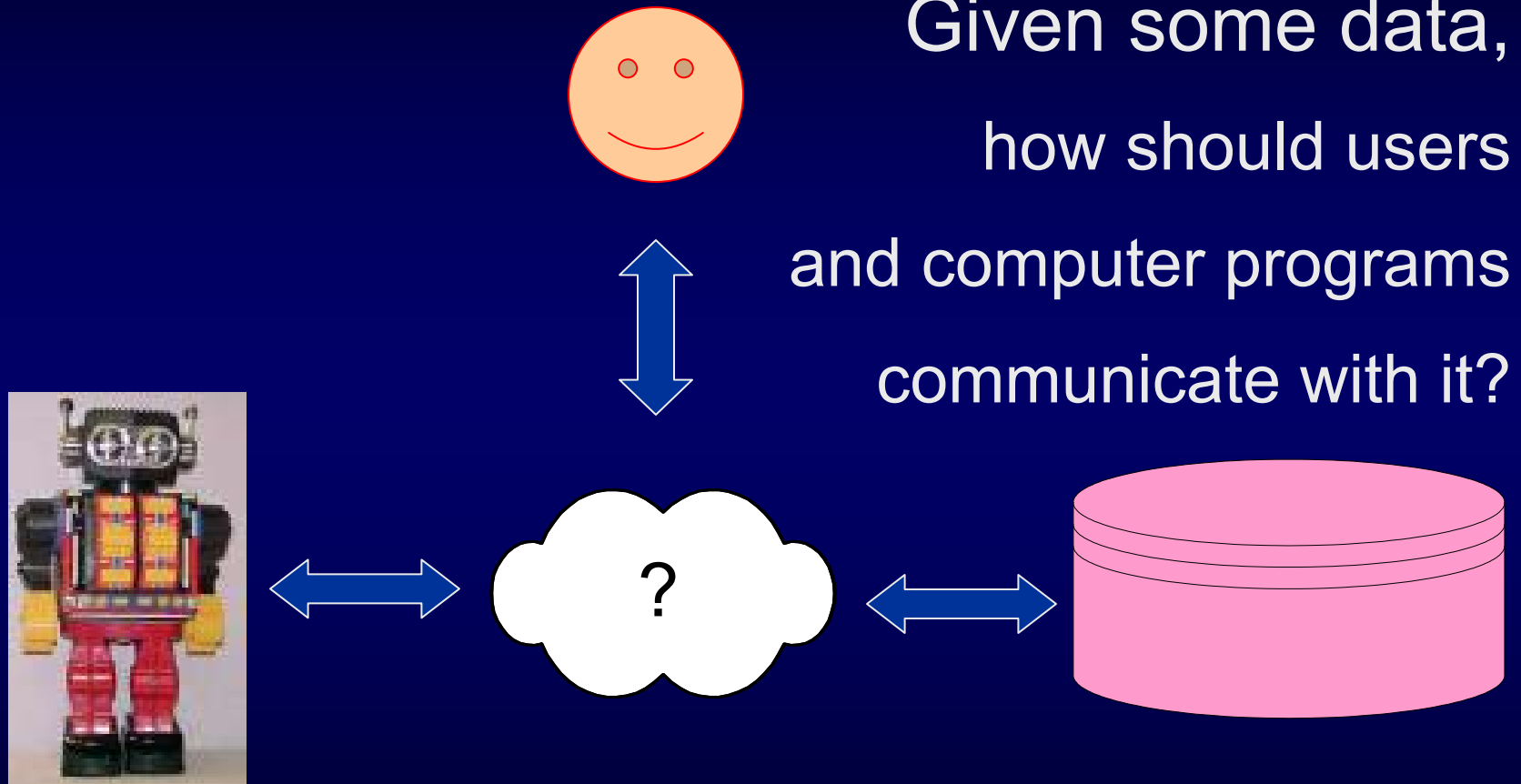
# Introduction to Standard Query Language

Erik Zeitler

UDBL

[erik.zeitler@it.uu.se](mailto:erik.zeitler@it.uu.se)

# Why a query language?



we need an interface to the data

# SQL does the job

- Data Definition Language (DDL)
  - Define/re-define database structure
- Data Manipulation Language (DML)
  - Updates
  - Queries
- Additional facilities
  - Views
  - Security, authorization
  - Integrity constraints
  - Transaction constraints
  - Rules for embedding SQL statements into other languages

# Outline

- Overview
  - What can SQL do for you?
- Background
  - and a simple example
- SQL and the relational data model
  - Example queries
- NULL values and 3-valued logic
  - Example queries

# Background

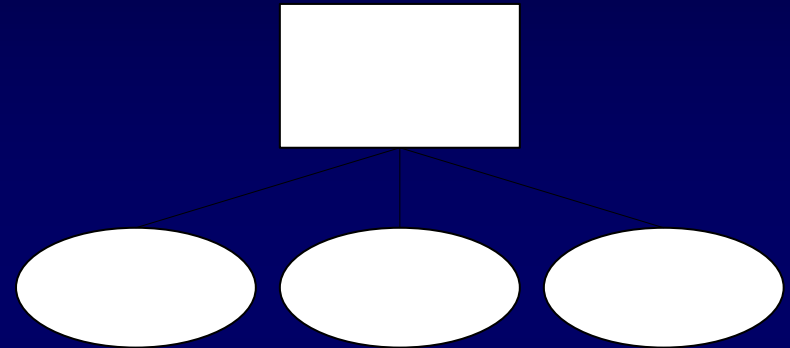
- History
  - SEQUEL (Structures English QUery Language) – early 70's, IBM Research
  - SQL (ANSI 1986), SQL1 or SQL86
  - SQL2 or SQL92
  - SQL3 or SQL99
    - Core specification and optional specialized packages
- SQL consists of ~20 basic commands
  - A lot of research money for each SQL command...
- Standard language for all commercial DBMS
  - Each DBMS has features outside standard

# Terminology

Theoretical foundation:

*The relational data  
model*

- relation – table
- tuple – row
- attribute – column



<i>column<sub>1</sub></i>	...	<i>column<sub>n</sub></i>
< <i>row 2</i> >		
...		
< <i>row n</i> >		

# Example database

**EMPLOYEE**

FNAME	MINIT	LNAME	<u>SSN</u>	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
-------	-------	-------	------------	-------	---------	-----	--------	----------	-----

**DEPARTMENT**

DNAME	<u>DNUMBER</u>	MGRSSN	MGRSTARTDATE
-------	----------------	--------	--------------

**DEPT\_LOCATIONS**

<u>DNUMBER</u>	<u>DLOCATION</u>
----------------	------------------

**PROJECT**

PNAME	<u>PNUMBER</u>	PLOCATION	DNUM
-------	----------------	-----------	------

**WORKS\_ON**

<u>ESSN</u>	<u>PNO</u>	HOURS
-------------	------------	-------

**DEPENDENT**

<u>ESSN</u>	<u>DEPENDENT_NAME</u>	SEX	BDATE	RELATIONSHIP
-------------	-----------------------	-----	-------	--------------

Schema diagram, database state (E/N ch 5, p 136-137)

(c) Addison Wesley Longman Inc

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John		Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin		Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia		Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer		Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh		Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce		English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad		Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James		Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE	DEPT_LOCATIONS	DNUMBER	DLOCATION
							Houston
							Stafford
							Bellaire
							Sugarland
	Research	5	333445555	1988-05-22			
	Administration	4	987654321	1995-01-01			
	Headquarters	1	888665555	1981-06-19			

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

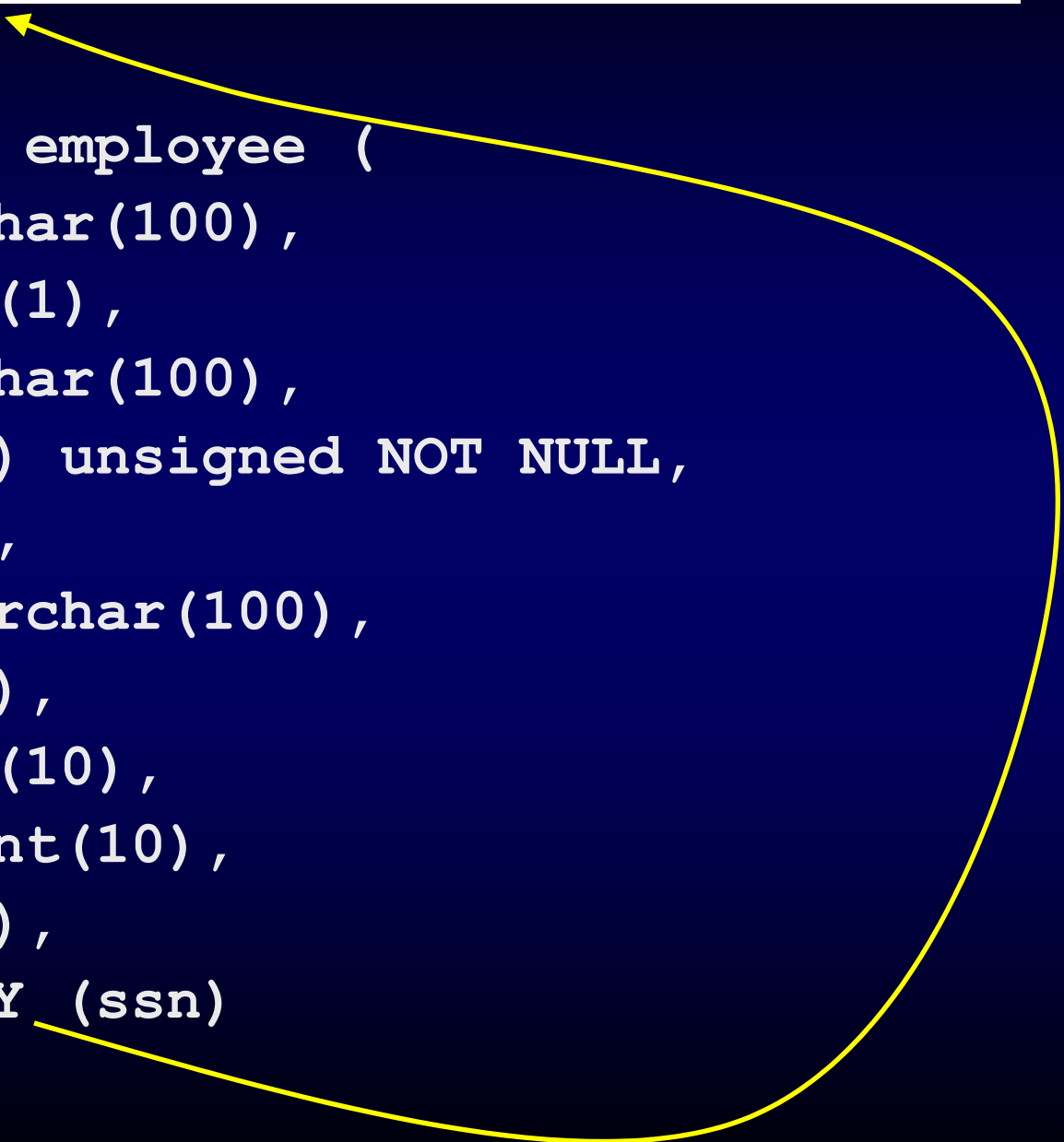
DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1987-05-05	SPOUSE



## EMPLOYEE

FNAME	MINIT	LNAME	<u>SSN</u>	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
-------	-------	-------	------------	-------	---------	-----	--------	----------	-----

```
CREATE TABLE employee (  
    fname varchar(100),  
    minit char(1),  
    lname varchar(100),  
    ssn int(10) unsigned NOT NULL,  
    bdate date,  
    address varchar(100),  
    sex char(1),  
    salary int(10),  
    superssn int(10),  
    dno int(10),  
    PRIMARY KEY (ssn)  
);
```



```
unix$ mysql -u root -p
> CREATE DATABASE comp;
> CONNECT comp;

> CREATE TABLE emp (
    fname varchar(100) ,
    lname varchar(100) ,
    ssn bigint unsigned NOT NULL
    PRIMARY KEY (ssn)
);

> INSERT INTO emp VALUES (
    'Erik' , 'Zeitler' , 197510061111
);

> SELECT * FROM emp;
> SELECT fname FROM emp;
```

# Recommendation

- [www.mysql.com](http://www.mysql.com)
- [www.mimer.com](http://www.mimer.com)
- Download & install on your PC
- Excellent reference manuals on the web sites

# Basic query statement: select – from – where

**SELECT**  $A_1, A_2, \dots, A_n$

**FROM**  $r_1, r_2, \dots, r_m$

**WHERE**  $P$ ;

- $A_1, A_2, \dots, A_n$  – list of attribute names to be retrieved
- $r_1, r_2, \dots, r_m$  – List of tables required to process the query
- $P$  – Conditional expression identifying the tuples to be retrieved
  - AND, OR, NOT,  $<$ ,  $<=$ ,  $=$ ,  $>=$ ,  $>$
- Result of the query is a table

# SQL and the relational data model

- Projection
- Cartesian product
- Selection
- Set operations
  - Union
  - Difference
  - Intersection
- Assignment operator
  - Rename relations
- Join
  - $\theta$  join
  - Equijoin
  - Natural join

# Relation algebra *projection*

- Projection is done in the SELECT clause:



The star (\*) denotes  
"all attributes"

Ex 1, Look at interesting fields

```
> select * from employee;  
> select fname from employee;  
> select fname, bdate from employee;
```

Ex 2, projection!

```
> select x,y,z from vectors;  
> select x,y from vectors;
```

# The SQL SELECT clause

- Projection
- Remove duplicates: **distinct**
  - > **select plocation from project;**
  - > **select distinct plocation from project;**
- Arithmetic expressions
  - > **select x/10, (y\*z)/2, z+3 from vectors;**
  - > **select ssn, salary, salary\*.327 from employee;** #

# Relational algebra *selection*

**SELECT**  $A_1, A_2, \dots, A_n$

**FROM**  $r_1, r_2, \dots, r_m$

**WHERE**  $P$ ;

- $P$  is the selection predicate
  - operates on attributes in relations  $r_1, r_2, \dots, r_m$
  - Selects tuples to be returned
- selection  $\approx$  filtering

Selection in SQL: The WHERE clause  $\rightarrow \rightarrow$



# The SQL WHERE clause

- Ex 1, Look for employee info

```
> select * from employee  
where fname='John' ;
```

- Ex 2, Look for employee info

```
> select * from employee  
where bdate > '1955-01-01'  
and salary between 30000 and 50000 ;
```

- Ex 3, vector length!

```
> select x,y,z from vectors  
where x > 10 and x*x+y*y+z*z < 200 ;
```

# Rel. algebra *Cartesian product*

Similar to Cartesian product of two vectors

$$(v_1 \quad v_2 \quad \dots \quad v_n) \times (w_1 \quad w_2 \quad \dots \quad w_n) = \begin{pmatrix} v_1 w_1 & & v_1 w_n \\ \vdots & \ddots & \\ v_n w_1 & & v_n w_n \end{pmatrix}$$

The Cartesian product forms  
all possible pairs  
of the elements  
of the operands

# The SQL FROM clause

Similarly, given two database tables

```
select *  
from persons, cars;
```

persons
Alex
John
Mike

X

cars
Audi
BMW
Mercedes

=

Alex	Audi
John	Audi
Mike	Audi
Alex	BMW
John	BMW
Mike	BMW
Alex	Mercedes
John	Mercedes
Mike	Mercedes

, this SQL query generates all possible **persons-cars** combinations.

More... <#>

# Select ... from ... where revisited

Basic SQL query: three *clauses*

**select** <projection-predicate>

**from** <table list>

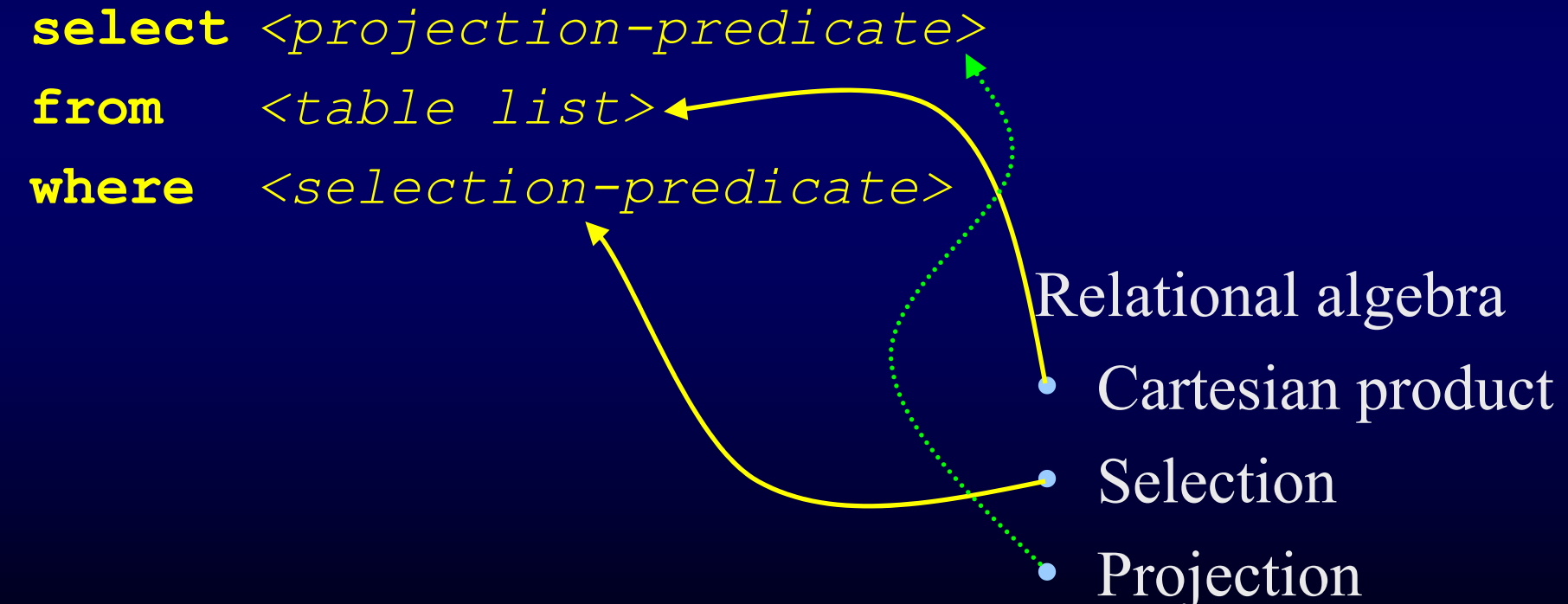
**where** <selection-predicate>

Relational algebra

• Cartesian product

• Selection

• Projection



# Select – from – where

**Ex 1:** Find all employees working at research dept

```
SELECT      EMPLOYEE.LNAME, ADDRESS  
FROM        EMPLOYEE, DEPARTMENT  
WHERE       DEPARTMENT.NAME='Research'  
            AND DNUMBER=DNO;
```

**Ex 2:** All employees and their managers

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME  
FROM        EMPLOYEE E, EMPLOYEE S  
WHERE       E.SUPERSSN=S.SSN;
```

# SQL and the relational data model

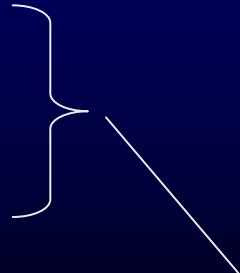
SELECT ... FROM ... WHERE ...



projection,  
cartesian product,  
selection

- Set operations

- Union
- Difference
- Intersection



*Operands must be union  
compatible*

- Assignment operator
  - Rename relations
- Join
  - $\theta$  join
  - Equijoin
  - Natural join

# Examples of set operations

- Retrieve all first names in the database

```
> select fname from employee
```

```
union
```

```
select dependent_name from dependent;
```

- Are there any projects in a town without departments?

```
> select plocation FROM project p
```

```
except
```

```
select dlocation FROM dept_locations;
```

# SQL and the relational data model

SELECT ... FROM ... WHERE ...



projection,  
cartesian product,  
selection

- Set operations
  - Union – **union**
  - Difference – **except**
  - Intersection – **intersect**

- **Assignment operator**
  - **Rename relations**
- Join
  - $\theta$  join
  - Equijoin
  - Natural join



# Rename, assignment

- Rename: **as**
  - > **select distinct superssn  
as 'manager social security number'  
from employee;**
- Assignment: **create table ... as select ...**
  - > **create table names as  
select fname from employee  
union  
select dependent\_name from dependent;**

# SQL and the relational data model

SELECT ... FROM ... WHERE ...

$\Leftrightarrow$

projection,  
cartesian product,  
selection

- Set operations
  - Union – **union**
  - Difference – **except**
  - Intersection – **intersect**

- Assignment operator
  - Rename relations
- **Join**
  - $\theta$  join
  - Equijoin
  - Natural join

# Join

- Relational algebra notation:  $R \bowtie_C S$
- $C$  – join condition
  - $C$  is on the form  $A_R \theta A_S$   
 $\theta$  is one of  $\{=, <, >, \leq, \geq, \neq\}$
  - Several terms can be connected as  $C_1 C_2 \dots C_K$ .
- Special cases
  - Equijoin:  $\theta$  is  $=$
  - Natural join: All identically named attributes in relations  $R$  and  $S$  have matching values

# SQL join

- Recall this query

```
SELECT      EMPLOYEE.LNAME, ADDRESS
FROM        EMPLOYEE, DEPARTMENT
WHERE       DEPARTMENT.NAME='Research'
AND DNUMBER=DNO;
```

- Equijoin
  - of **employee** and **department** tables
  - w.r.t. employee.dnumber and department.dno.
- Joins are cartesian products with some selection criteria

# SQL join

- Another way:
  - `alter table project change pnumber pno int(10);`

# One more example

- Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise

```
SELECT      FNAME, LNAME,  
            1.1*SALARY AS INC_SAL  
FROM        EMPLOYEE, WORKS_ON, PROJECT  
WHERE       SSN=ESSN  
            AND PNO=PNUMBER  
            AND PNAME='ProductX' ;
```

# Special comparison

- Matching string patterns

- Use LIKE
- % for any number of arbitrary symbol
- \_ for any symbol

```
select * from employee
where address like '%Houston%';
```

- Approx math equality

- Use  $\text{abs}(x - x_1) < \varepsilon$ :

```
select * from employee
where abs(salary-30000) < 8000;
```

- Use BETWEEN:

```
select * from employee
where salary between 22000 and 38000;
```

# NULL values

- Sometimes an attribute is
  - Unknown (date of birth unknown)
  - Unavailable/withheld (refuses to list home phone #)
  - Not applicable (last college degree)
- Need to represent these cases in a DB!
- Solution: NULL.
  - What about logical operations involving NULL?  
⇒ Need to extend logic...



# 3-valued logic

<b>AND</b>	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

<b>OR</b>	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

<b>NOT</b>	TRUE	FALSE	UNKNOWN
	FALSE	TRUE	UNKNOWN

# Comparison of NULL values

- **=, ≠, >, <, LIKE, ...**
  - won't work. NULL is UNDEFINED!
- SQL check for NULL
  - **IS NULL**
  - **IS NOT NULL**
- JOIN operations
  - Tuples with NULL values in the join columns  
⇒ Not included in result
  - Exception: OUTER JOIN (E/N 8.5.6)

# NULL

- Find out who is The Big Boss

```
select fname, lname  
from employee  
where superssn is NULL;
```

# Aggregate functions

- Avg — average value
- Min — minimum value
- Max — maximum value
- Sum — sum of values
- Count — number of values

# Aggregate functions – group by

- Average salary

```
select avg(salary)
from employee;
```

- Average salary at each department

```
select dname, avg(salary)
from employee, department
where dno=dnumber group by dno;
```

# Aggregate functions – HAVING

- Find the projects that more than two employees are assigned to:
  - retrieve the project number,
  - its name,
  - and the number of its employees

```
SELECT project.pnumber, pname , count(*)  
FROM project, works_on  
WHERE project.pnumber = works_on.pno  
GROUP BY project.pnumber, pname  
HAVING count(*)>2;
```

# Summary

- Clauses:

<b>SELECT</b>	<attribute list>
<b>FROM</b>	<table list>
[ <b>WHERE</b>	<condition>]
[ <b>GROUP BY</b>	<grouping attributes>
[ <b>HAVING</b>	<group condition>]
[ <b>ORDER BY</b>	<attribute list>]

- More Than One Way To Do It™...

# Views

- Frequently posed queries should be expressed as views.

```
> create view tax_view as  
  select ssn, salary, salary*.327  
  from employee;
```

```
> select * from tax_view;
```



# Views

- Creating a view will not result in a new table. Views are not tables themselves
  - they are *views* of the underlying tables.
- A view query will return the state of the underlying tables.
- Consequence:
  - underlying tables are changed
  - $\Rightarrow$
  - the view will change

# Views

- Ex 1:

```
> update table employee  
  set salary = 1000000  
  where ssn = 123456;
```

```
> select * from tax_view;
```

- Ex 2:

We are removing one column!

```
> alter table employee drop salary;
```

The view will not work any more

```
> select * from tax_view;
```