Introduction to

Standard Query Language

Erik Zeitler
UDBL
erik.zeitler@it.uu.se
Why a query language?

Given some data, how should users and computer programs communicate with it?

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

<table>
<thead>
<tr>
<th>(column_1)</th>
<th>…</th>
<th>(column_n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;row 2&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(&lt;row n&gt;)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example database

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FNAME</td>
<td>MINIT</td>
<td>LNAME</td>
<td>SSN</td>
<td>BDATE</td>
<td>ADDRESS</td>
<td>SEX</td>
<td>SALARY</td>
<td>SUPERSSN</td>
<td>DNO</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DNAME</td>
<td>DNUMBER</td>
<td>MGRSSN</td>
<td>MGRSTARTDATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPT_LOCATIONS</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DNUMBER</td>
<td>DLOCATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROJECT</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PNAME</td>
<td>PNUMBER</td>
<td>LOCATION</td>
<td>DNUM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WORKS_ON</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSN</td>
<td>PNO</td>
<td>HOURS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPENDENT</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSN</td>
<td>DEPENDENT_NAME</td>
<td>SEX</td>
<td>BDATE</td>
<td>RELATIONSHIP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Schema diagram, database state (E/N ch 5, p 136-137)
(c) Addison Wesley Longman Inc
<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>FNAME</th>
<th>MINIT</th>
<th>LNAME</th>
<th>SSN</th>
<th>BDATE</th>
<th>ADDRESS</th>
<th>SEX</th>
<th>SALARY</th>
<th>SUPERSSN</th>
<th>DNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>1234</td>
<td>Jones</td>
<td>123456789</td>
<td>1985-01-09</td>
<td>231 Fondle, Houston, TX</td>
<td>M</td>
<td>35000</td>
<td>null</td>
<td>2</td>
</tr>
<tr>
<td>Franklin</td>
<td>Viking</td>
<td>3033</td>
<td>Jones</td>
<td>303456789</td>
<td>1985-12-08</td>
<td>630 Vose, Houston, TX</td>
<td>M</td>
<td>45000</td>
<td>303456789</td>
<td>5</td>
</tr>
<tr>
<td>Alicia</td>
<td>Zelma</td>
<td>9699</td>
<td>Smith</td>
<td>969987777</td>
<td>1989-09-09</td>
<td>332 Galle, Spring, TX</td>
<td>F</td>
<td>25000</td>
<td>969987777</td>
<td>4</td>
</tr>
<tr>
<td>Jennifer</td>
<td>Wallace</td>
<td>8676</td>
<td>Smith</td>
<td>8676564321</td>
<td>1941-05-20</td>
<td>291 Berry, Bellair, TX</td>
<td>F</td>
<td>43000</td>
<td>8676564321</td>
<td>4</td>
</tr>
<tr>
<td>Ramiro</td>
<td>Naniya</td>
<td>6666</td>
<td>Gonzalez</td>
<td>666698444</td>
<td>1962-09-15</td>
<td>575 Fire Oak, Humble, TX</td>
<td>M</td>
<td>30000</td>
<td>333456555</td>
<td>5</td>
</tr>
<tr>
<td>Joyce</td>
<td>English</td>
<td>4554</td>
<td>Garcia</td>
<td>454534545</td>
<td>1982-02-31</td>
<td>5631 Rice, Houston, TX</td>
<td>F</td>
<td>25000</td>
<td>455434545</td>
<td>5</td>
</tr>
<tr>
<td>Ahmad</td>
<td>Jabbar</td>
<td>9876</td>
<td>Rees</td>
<td>987698767</td>
<td>1989-01-29</td>
<td>980 Dallas, Houston, TX</td>
<td>M</td>
<td>25000</td>
<td>987698767</td>
<td>4</td>
</tr>
<tr>
<td>James</td>
<td>Borg</td>
<td>8666</td>
<td>Jones</td>
<td>866666666</td>
<td>1937-11-10</td>
<td>450 Stone, Houston, TX</td>
<td>M</td>
<td>55000</td>
<td>null</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th>DNAME</th>
<th>DNUMBER</th>
<th>MGRSSN</th>
<th>MGRSTARTDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>5</td>
<td>333456555</td>
<td>1988-05-25</td>
<td></td>
</tr>
<tr>
<td>Administration</td>
<td>4</td>
<td>9876564321</td>
<td>1985-01-01</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WORKS_ON</th>
<th>ESSN</th>
<th>PNO</th>
<th>HOURS</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789</td>
<td>123456789</td>
<td>1</td>
<td>32.5</td>
</tr>
<tr>
<td>123456789</td>
<td>123456789</td>
<td>2</td>
<td>7.5</td>
</tr>
<tr>
<td>666698444</td>
<td>666698444</td>
<td>3</td>
<td>40.0</td>
</tr>
<tr>
<td>454534545</td>
<td>454534545</td>
<td>1</td>
<td>20.0</td>
</tr>
<tr>
<td>454534545</td>
<td>454534545</td>
<td>2</td>
<td>20.0</td>
</tr>
<tr>
<td>333456555</td>
<td>333456555</td>
<td>2</td>
<td>10.0</td>
</tr>
<tr>
<td>333456555</td>
<td>333456555</td>
<td>3</td>
<td>10.0</td>
</tr>
<tr>
<td>333456555</td>
<td>333456555</td>
<td>10</td>
<td>10.0</td>
</tr>
<tr>
<td>333456555</td>
<td>333456555</td>
<td>20</td>
<td>10.0</td>
</tr>
<tr>
<td>9876564321</td>
<td>9876564321</td>
<td>10</td>
<td>10.0</td>
</tr>
<tr>
<td>9876564321</td>
<td>9876564321</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>9876564321</td>
<td>9876564321</td>
<td>30</td>
<td>5.0</td>
</tr>
<tr>
<td>9876564321</td>
<td>9876564321</td>
<td>30</td>
<td>20.0</td>
</tr>
<tr>
<td>866666666</td>
<td>866666666</td>
<td>20</td>
<td>15.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>PNAME</th>
<th>PNUMBER</th>
<th>LOCATION</th>
<th>DNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductX</td>
<td>1</td>
<td>1</td>
<td>Bellair</td>
<td>5</td>
</tr>
<tr>
<td>ProductY</td>
<td>2</td>
<td>2</td>
<td>Sugarland</td>
<td>5</td>
</tr>
<tr>
<td>ProductZ</td>
<td>3</td>
<td>3</td>
<td>Houston</td>
<td>5</td>
</tr>
<tr>
<td>Computerization</td>
<td>10</td>
<td>10</td>
<td>Stafford</td>
<td>4</td>
</tr>
<tr>
<td>Reorganization</td>
<td>20</td>
<td>20</td>
<td>Houston</td>
<td>1</td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
<td>30</td>
<td>Stafford</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPENDENT</th>
<th>ESSN</th>
<th>DEPENDENT_NAME</th>
<th>SEX</th>
<th>BDATE</th>
<th>RELATIONSHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>333456555</td>
<td>Alice</td>
<td>F</td>
<td>1986-01-05</td>
<td>DAUGHTER</td>
<td></td>
</tr>
<tr>
<td>333456555</td>
<td>Theodore</td>
<td>M</td>
<td>1983-10-25</td>
<td>SON</td>
<td></td>
</tr>
<tr>
<td>333456555</td>
<td>Joy</td>
<td>F</td>
<td>1958-05-03</td>
<td>SPOUSE</td>
<td></td>
</tr>
<tr>
<td>9876564321</td>
<td>Abner</td>
<td>M</td>
<td>1982-02-28</td>
<td>SPOUSE</td>
<td></td>
</tr>
<tr>
<td>123456789</td>
<td>Michael</td>
<td>M</td>
<td>1988-01-04</td>
<td>SON</td>
<td></td>
</tr>
<tr>
<td>123456789</td>
<td>Alice</td>
<td>F</td>
<td>1989-12-30</td>
<td>DAUGHTER</td>
<td></td>
</tr>
<tr>
<td>123456789</td>
<td>Elizabeth</td>
<td>F</td>
<td>1987-05-05</td>
<td>SPOUSE</td>
<td></td>
</tr>
</tbody>
</table>
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
- www.mimer.com
- Download & install on your PC
- Excellent reference manuals on the web sites
Basic query statement: `select – from – where`

**SELECT** \( A_1, A_2, \ldots, A_n \)

**FROM** \( r_1, r_2, \ldots, r_m \)

**WHERE** \( P; \)

- \( A_1, A_2, \ldots, A_n \) – list of attribute names to be retrieved
- \( r_1, r_2, \ldots, r_m \) – 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`
  
  \[
  \text{> select } \text{plocation from project;}
  \]
  
  \[
  \text{> select distinct plocation from project;}
  \]

• Arithmetic expressions

  \[
  \text{> select x/10, (y*z)/2, z+3 from vectors;}
  \]
  
  \[
  \text{> select ssn, salary, salary*.327 from employee;}
  \]
Relational algebra selection

SELECT $A_1, A_2, \ldots, A_n$
FROM $r_1, r_2, \ldots, r_m$
WHERE $P$;

• $P$ is the selection predicate
  • operates on attributes in relations $r_1, r_2, \ldots, r_m$
  • 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

\[(v_1 \ v_2 \ \ldots \ v_n) \times (w_1 \ w_2 \ \ldots \ w_n) = \begin{pmatrix} v_1 w_1 & v_1 w_n \\ \vdots & \vdots \\ v_n w_1 & v_n w_n \end{pmatrix}\]

The Cartesian product forms all possible pairs of the elements of the operands.
The SQL FROM clause

Similarly, given two database tables

<table>
<thead>
<tr>
<th>persons</th>
<th>cars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>Audi</td>
</tr>
<tr>
<td>John</td>
<td>BMW</td>
</tr>
<tr>
<td>Mike</td>
<td>Mercedes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>select * from persons, cars;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
</tr>
<tr>
<td>John</td>
</tr>
<tr>
<td>Mike</td>
</tr>
<tr>
<td>Alex</td>
</tr>
<tr>
<td>John</td>
</tr>
<tr>
<td>Mike</td>
</tr>
<tr>
<td>Alex</td>
</tr>
<tr>
<td>John</td>
</tr>
<tr>
<td>Mike</td>
</tr>
</tbody>
</table>

, this SQL query generates all possible persons-cars combinations.
Basic SQL query: three *clauses*

```sql
select <projection-predicate>
from <table list>
where <selection-predicate>
```
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

• Assignment operator
  • Rename relations

• Join
  • θ join
  • Equijoin
  • Natural join

Operands must be union compatible
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 …
⇔
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 $C_1 C_2 \ldots C_K$.

• Special cases
  • Equijoin: $\theta$ is $=$
  • Natural join: All identically named attributes in relations $R$ and $S$ have matching values
SQL join

- Recall this query
  
  ```sql
  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

<table>
<thead>
<tr>
<th><strong>AND</strong></th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>OR</strong></th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>TRUE</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>NOT</strong></th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>
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)
Find out who is The Big Boss

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

- Average salary at each department
  ```sql
  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

```sql
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:
  SELECT <attribute list>
  FROM <table list>
  [WHERE <condition>]
  [GROUP BY <grouping attributes>]
  [HAVING <group condition>]
  [ORDER BY <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;