CSCI262
System Security (**S8a**)

An introduction to Databases and SQL

Dilbert: 17-Nov-1995

Outline

- Information versus data.
- Security properties for databases.
- Relational databases.
- Structured Query Language (SQL).
- Database keys.
- Integrity rules.

Information versus data

- What is the difference between information and data?
- Informally not necessary anything …
- … but we are interested in distinguishing between:
  - The raw data that we might store, such as the ages of 50,000 students in our database.
  - … and the information, such as the average student age that a database allows us to extract.
- Typically we think of data as the collected input, while information is the output resulting from processes applied to data.
  - Information is knowledge.

- A database management system (DBMS) allows related data to be centrally stored and controlled.
  - Controlled includes standardising how data is entered, updated, retrieved and deleted.
  - A DBMS integrates the data and the management of the data.
- You might remember the concept of entropy, it can be used to measure the amount of information.
  - (Raw) data is almost certainly an inefficient representation of information.

Types of information

- There are various categories of information that could be extracted from a database.
- **Exact information**:
  - Alice is 25 years old.
- **Bounds**:
  - Alice is less than 70 years old.
- **Negative results**:
  - Alice is not a male.
  - There is nobody more than 95 years old.
- **Existence**:
  - There exists a person with red hair.

Security properties for databases…

- The primary database asset is the stored data, and the information that can be extracted from it.
- **C**onfidentiality…
  - – Assets cannot be accessed by persons not authorised to access that asset.
- **I**ntegrity…
  - – Assets cannot be modified by persons not authorised to modify that asset.
- **A**vailability…
  - – Assets should be accessible to those allowed access.
- It's still about access control and authentication.

Models of databases

- There are three fairly well known traditional model of databases:
  - **Hierarchial**: A tree structure.
  - **Network**: A graph structure.
  - **Relational**: Data is stored in pre-defined tables.
- The most widely used database model is the relational databases one, and we will restrict our attention to relational databases.
- Another popular database type, although it tends to be applied in limited areas, is the object oriented database (OODB).
  - While traditional databases deal with atomic data types like numbers and character strings, OODB's deal with BLOBs (binary large objects).
- Nowadays we also have NoSQL databases

The relational database model

- A relational database is a collection of relational tables, which are referred to as relations or entities.

Relation name

N-tuple (row)

Attribute name

Relational schema (header)

EMPLOYEE

E#	NAME	DOB	SALARY	CITY	STREET	BLDG#
10	John	12-DEC-67	60K	Sydney	Station	12
20	Mary	01-MAR-56	100K	Perth	Victoria	5
30	James	08-SEP-68	90K	W'Gong	Barham	13
40	Kate	17-OCT-80	50K	W'gong	Barham	13
…	…	…	…	…	…	…

Column

Attribute value

Relation (relational table)

- A **relation** (or table or sometimes file) can be written as

$$\mathbf{R \subseteq D_1 \times D_2 \times \ldots \times D_n}$$

  where $D_1, D_2, \ldots, D_n$ are the domains of the attributes $A_1, A_2, \ldots, A_n$.
- The attributes are the columns so the relation corresponds to the product space over the spaces of each attribute, and are sometimes called fields.
- The elements of the relation are called n-tuples and are simply rows in the table.
  - They are sometimes called records.
- An n-tuple is a sequence $<v_1, v_2, \ldots, v_n>$ where $v_i \in D_i$ for each $i = 1, 2, \ldots, n$.

Structured Query Language (SQL)

- SQL has a long history.
  - It was developed by IBM in the mid 1970's.
- SQL is a (sort-of) standard language designed to carry out interactions with relational databases.
  - This includes **inserts, updates, retrievals** and **deletions**.
  - The SQL commands fall into two categories: Data Manipulation Language (DML) or Data Definition Language (DDL).
- There seem to be several versions of the standard and various implementation differences but at the basic level SQL is quite similar.
- On the next couple of slides there are examples of different SQL statements…

An INSERT statement

```
INSERT INTO EMPLOYEE VALUES
(70,'James','12-08-78', 90000,'Sydney', Pitt,45);
```

Two UPDATE statements

```
UPDATE EMPLOYEE
SET DOB ='13-08-78'
WHERE E# = 70;

UPDATE EMPLOYEE
SET SALARY = SALARY + 0.1*SALARY;
```

EMPLOYEE

E#	NAME	DOB	SALARY	CITY	STREET	BLDG#
10	John	12-DEC-67	66K	Sydney	Station	12
20	Mary	01-MAR-56	110K	Perth	Victoria	5
30	James	08-SEP-68	99K	W'Gong	Barham	13
40	Kate	17-OCT-80	55K	W'gong	Barham	13
70	James	13-08-78	99K	Sydney	Pitt	45

A DELETE statement

```
DELETE FROM EMPLOYEE
WHERE DOB < '01-JAN-70';
```

SELECT retrieves data/information from a relation.

* is a wildcard…

Some SELECT statements

```
SELECT *
FROM EMPLOYEE;

SELECT E#, NAME, DOB
FROM EMPLOYEE;

SELECT E#, NAME, SALARY
FROM EMPLOYEE
WHERE (DOB <= '01-JAN-70') AND (CITY = 'Sydney');
```

Relational Views

- A relational view is a named derived virtual relation defined in terms of the other named relations and/or other relational views.
- One use of views is for access control.
  - We allow access to a derived view rather a base relation.

Two CREATE VIEW statements

```
CREATE VIEW SYD_EMP AS(
        SELECT *
        FROM EMPLOYEE
        WHERE CITY = 'Sydney');

CREATE VIEW OLD_EMP AS(
        SELECT E#, NAME
        FROM EMPLOYEE
        WHERE DOB < '01-01-50');
```

Changing data through a view…

```
SELECT E#, NAME
FROM SYD_EMP
WHERE STREET = 'Broadway';

DELETE FROM SYD_EMP
WHERE NAME = 'Patrick';

UPDATE SYD_EMP
SET NAME = 'Maggie'
WHERE E# = 77;

INSERT INTO SYD_EMP VALUES(
(13,'Mike','29-FEB-72',60000,'Hobart',Victoria,5);
```

Database keys

- These are not cryptographic keys.
- There are a few terms we need to describe.
- *A* **minimal key for a relation** is an attribute that identifies all tuples of the relation in a unique way, or a set of attributes that does the same but for which no proper subset is a minimal key.

- For EMPLOYEE(E#,NAME,DOB,SALARY,CITY,STREET,BLDG#), (E#) is a minimal key.
- (E#,NAME) is NOT a minimal key because (E#) is a proper subset and a minimal key.

EMPLOYEE

E#	NAME	DOB	SALARY	CITY	STREET	BLDG#
10	John	12-DEC-67	60K	Sydney	Station	12
20	Mary	01-MAR-56	100K	Perth	Victoria	5
30	James	08-SEP-68	90K	W'Gong	Barham	13
40	Kate	17-OCT-80	50K	W'gong	Barham	13
…	…	…	…	…	…	…

More terminology

- The **Primary key** is the arbitrarily selected minimal key.
- A **Candidate key** is a minimal key which is not the primary key.
- A **Foreign key** is an attribute, or set of which together, the values of which are the same as the values of a primary or candidate key in another relation.
  - This defines a relationship between the two relations.

An example…

- Did and Eid are, respectively, the primary keys for the Department and Employee relations.
- Did is a foreign key in the Employee relation.

Department Table

Did	Dname	Dacctno
4	human resources	528221
8	education	202035
9	accounts	709257
13	public relations	755827
15	services	223945

Did: Primary key

Employee Table

Ename	Did	Salarycode	Eid	Ephone
Robin	15	23	2345	6127092485
Neil	13	12	5088	6127092246
Jasmine	4	26	7712	6127099348
Cody	15	22	9664	6127093148
Holly	8	23	3054	6127092729
Robin	8	24	2976	6127091945
Smith	9	21	4490	6127099380

Did: Foreign key; Eid: Primary key

(a) Two tables in a relational database

From page 147 of Stallings and Brown (Computer Security: Principles and Practice, Pearson Education 2008).

Creating a table…

- We haven't seen table creation, and it is probably a good idea to.
  - So here the instructions to create the two tables used in the example on the previous slide, again taken from Stallings & Brown.

```
CREATE TABLE department (
    Did INTEGER PRIMARY KEY,
    Dname CHAR (30),
    Dacctno CHAR (6) );

CREATE TABLE employee (
    Ename CHAR (30),
    Did INTEGER,
    SalaryCode INTEGER,
    Eid INTEGER PRIMARY KEY,
    Ephone CHAR (10),
    FOREIGN KEY (Did) REFERENCES department (Did));
```

Integrity rules

- These are a couple of meta-consistency rules that should exist independent of the application domain.
- **Entity integrity rule**: No component of the primary key of a relation is allowed to accept NULLs.
- In other words, when we are inserting a tuple, all values of primary key attributes must be provided, otherwise we might not be able to find a tuple
- **Referential integrity rule**: The database must not contain unmatched foreign key values.

CSCI262
System Security (**S8b**)

Secure Statistical Databases

Dilbert: 17-Apr-1996

Outline

- What is a statistical database?
- Why use one? →
  - What are we protecting?
- Types of disclosure.
- Direct attacks.
- Statistical queries.
- Protecting against inference.

What is a statistical database?

- A statistical database is primarily defined on the basis of the data or information that it provides… that is based on the nature of the query results.
- Rather than getting exact entries, or precise lists of entries, we get statistical information, such as an average or sum.
  - Direct access isn't allowed.
- A single physical database can have a statistical database interface and the typical type of interface we saw earlier with the standard select queries.
  - A pure statistical database stores only statistical data.

Why use one? → What are we protecting?

- Data inherently sensitive:
  - Information about the locations of WMD (Weapons of Mass Destruction).
- Data from a sensitive source:
  - Information that compromises the identity of the informer.
- Data declared as sensitive:
  - Classified military data.
- A sensitive component of a tuple (row):
  - The values of attribute salary.
- Data sensitive in relation to earlier disclosed information:
  - X invested money in company Y.
  - The owner of Y and X belong to the same golf club and frequently play together.

Types of disclosure

- Exact data:
  - Analysis of the results of correlated complex queries provides the exact value of data item.
- Bounds:
  - Analysis of the results of correlated complex queries provides the lower and upper bounds for a value of item.
- Negative results:
  - Analysis of the results of correlated complex queries provides information that v is NOT a value of item x.
- Existence:
  - Analysis of the results of correlated complex queries provides information that a data item exists.

Types of information

- There are various categories of information that could be extracted a database.
  - We mentioned this earlier but this is specifically relevant to inference so we will repeat these categories and examples.
- **Exact information**:
  - Alice is 25 years old.
- **Bounds**:
  - Alice is less than 70 years old.
- **Negative results**:
  - Alice is not a male.
  - There is nobody more than 95 years old.
- **Existence**:
  - There exists a person with red hair.

- **Associations or relations among pieces of information**:
  - Alice is younger than Bob.
  - Alice has spent more time as a student than Bob.
- We might want to control access to some of those.
- And it isn't simply a matter of not letting people get access to certain entries in the database!
- The application may insist we cannot get the address of a specific person.
  - But we can find out there is a 25 year old female student living in Crypto Lane.
  - A separate query may reveal that Alice is the only 25 year old female.
  - We can infer that Alice lives in Crypto Lane.
  - Think sets and Venn diagrams, if that helps. ☺

Dilbert: 17-Apr-1996

The follow up ...

Dilbert: 18-Apr-1996

Attacks…an example…

- Before describing some of the formalisms for statistical databases we will consider an example of a non-statistical inference attack.
- We will try and construct queries to precisely match one data item.

```
SELECT NAME
FROM EMPLOYEE
WHERE SALARY BETWEEN 50000 AND 100000;
```

- If 3 names are found then we could try:

```
SELECT NAME
FROM EMPLOYEE
WHERE SALARY BETWEEN 75000 AND 100000;
```

- If no names are found then we try:

```
SELECT NAME
FROM EMPLOYEE
WHERE SALARY BETWEEN 62500 AND 75000;
```

- And so on …

- The database system might enforce a consistency constraint:
  - *If manager then salary between 100,000 and 200,000.*
- Then a sequence of INSERT statements may disclose the constraint, on the basis of their INSERT's being denied, …
  - And the constraint may itself be confidential.
- A subsequent query …

```
SELECT COUNT(*)
FROM EMPLOYEE
WHERE POSITION = 'MANAGER'
```

- … would provide an indication of how much money the company spends on management salaries.

COUNT?

- COUNT is an example of an aggregate function.
- SUM, AVG, MIN, MAX are the other basic ones.
- These provide aggregate information on a column of a relation.
- Statistical databases provide information by means of statistical (aggregate) queries on an attribute (column) of a relational table.
- There are a lot of other aggregate functions, and it is possible to define new ones for use by the user.

AVG	COLLECT	CORR
CORR_*	COUNT	COVAR_POP
COVAR_SAMP	CUME_DIST	DENSE_RANK
FIRST	GROUP_ID	GROUPING
GROUPING_ID	LAST	MAX
MEDIAN	MIN	PERCENTILE_CONT
PERCENTILE_DISC	PERCENT_RANK	RANK
REGR		
STATS_BINOMIAL_TEST		
STATS_CROSSTAB		
STATS_F_TEST	STATS_KS_TEST	STATS_MODE
STATS_MW_TEST	STATS_ONE_WAY_ANOVA	
STATS_T_TEST_*		
STATS_WSR_TEST		STDDEV
STDDEV_POP	STDDEV_SAMP	SUM
SYS_XMLAGG	VAR_POP	VAR_SAMP
VARIANCE	XMLAGG	

Statistical queries …

SELECT *aggregate-function*
FROM *relational-table*
WHERE *query-predicate*
[GROUP BY group-by-*attribute-list*];

- We have already seen the *aggregate functions*.
- The *query-predicates* are predicates that determine the tuples (rows) that are used to compute the *aggregate-function*.
  – These are conditions we need to match.
  – The values in the columns used are not necessarily directly accessible.
- The tuples (rows) matching the query-predicate form the *query-set*.

Some examples…

- Consider that we have a relationship described by the following schema:

```
EMPLOYEE(E#,NAME,GENDER,SUBURB,STREET,SALARY,POSITION,DEPTNAME)
```

- Then one statistical query would be …

```
SELECT AVG(SALARY)
FROM EMPLOYEE
WHERE DEPTNAME = 'SALES' AND GENDER = 'F';
```

- Another would be …

```
SELECT COUNT(*)
FROM EMPLOYEE
WHERE SALARY > 100000
GROUP BY DEPTNAME;
```

Sensitivity levels relationships: Aggregation…

- The sensitivity level, or classification, of an aggregate computed over a group of values usually differs from the sensitivity levels of the individual values.
- For example:
  - The sensitivity level of an average salary in a department is lower than sensitivity level of the salaries of individual employees.
- Should the MIN and MAX have lower classifications?
  - After all, they are the salaries of individuals.
  - But they aren't tied to identities.

… and inference

- Inference means the derivation of sensitive information from non-sensitive (typically aggregated) data.
- For example:
  - An average salary of all employees older than 60 discloses an exact value of salary if exactly one employee older than 60 is employed.

Attacks...

- We can have attacks where the aggregates are over small enough samples that information about individual elements of data is leaked.
  - The results from several aggregate queries can be used also.
- These are often referred to as **direct attacks** and differ from **indirect attacks**, where information from external sources is combined with the results of aggregate queries.
- For example, we might know, independent of the database, who lives in which suburb and who is a member of which department.
- The query …

```
SELECT SUM(SALARY), COUNT(*)
FROM EMPLOYEE
WHERE GROUP BY DEPTNAME, SUBURB;
```

- … may then disclose the salaries of the employees who are the only people employed in a department and living in a particular suburb.

- Another example:
  - We know from external sources that Mary is the only female employee in Security department.
  - So we query:

```
SELECT SUM(SALARY)
FROM EMPLOYEE
WHERE DEPTNAME = 'SECURITY' AND GENDER = 'F'
```

- This discloses Mary's salary, since the summation is over a query set consisting of one row.
- What if the system refused to reveal the results if the summation is performed over a small number of rows, in other words with a small query set?
  - We can still construct an attack, but now from a series of acceptable aggregate queries.

- So, we know from the external sources that Mary is employed in Security department.
- We can perform a series of queries to infer Mary's salary.

```
SELECT COUNT(*)
FROM EMPLOYEE
WHERE DEPTNAME = 'SECURITY';
```

⇒ 5 (total number of employees in Security)

```
SELECT COUNT(*)
FROM EMPLOYEE
WHERE DEPTNAME = 'SECURITY' AND GENDER = 'M';
```

⇒ 4 (there is only one female employed in Security)

```
SELECT AVG(SALARY))
FROM EMPLOYEE
WHERE DEPTNAME = 'SECURITY';
```

⇒ 40,000 (total salaries in Security = 200,000)

```
SELECT AVG(SALARY)
FROM EMPLOYEE
WHERE DEPTNAME = 'SECURITY' AND GENDER = 'M';
```

⇒ 38,000 (total salaries of male employees in Security = 152,000)

- And now we can now infer her salary…

⇒ 200,000 - 152,000 = 48,000 (a salary of the only female in Security; Mary is a female

Tracker attacks…

- At some point the sophistication of the combined attacks produces something called a Tracker.
  - This is a sequence of queries that allow us to isolate characteristics of an individual.
  - By analysing the tracker results we can sometimes produce a series of algebraic relationships, containing variables corresponding to particular query results.
    - The characteristics can then be expressed as functions of the queries results.

A tracker example ...

- In a yet another example if we know that a triple [NAME, SUBURB, STREET] usually uniquely identifies the employees, then we can find a salary of any employee in the following way:

```
SELECT SUM(SALARY)
FROM EMPLOYEE
WHERE (NAME = 'JOHN' AND
      SUBURB = 'LIVERPOOL' AND
      STREET = 'STATION') OR
        POSITION ='MANAGER';
```

$\Rightarrow$ ΣM = 5*10^5 (assume that John is a manager)

```
SELECT SUM(SALARY)
FROM EMPLOYEE
WHERE (NAME = 'JOHN' AND
      SUBURB = 'LIVERPOOL' AND
      STREET = 'STATION') OR
        NOT (POSITION ='MANAGER');
```

$\Rightarrow$ ΣnonM + John = 7*10^5

```
SELECT SUM(SALARY)
FROM EMPLOYEE;
```

$\Rightarrow$ ΣM + ΣnonM = 11*10^5

- Continuing, we get a system of 3 linear equations with 3 unknown variables, ΣM, $\Sigma nonM$, and `John`:

(1) $\Sigma M = 5*10^5$

(2) $\Sigma nonM + John = 7*10^5$

(3) $\Sigma M + \Sigma nonM = 11*10^5$

- From(1)+(2)-(3) we obtain: $John = 1*10^5$

- What if John is not a manager ? Then the system of equations is as follows:

(1) $\Sigma M + John = 5*10^5$

(2) $\Sigma nonM = 7*10^5$

(3) $\Sigma M + \Sigma nonM = 11*10^5$

- Why can we not simply say:

```
SELECT SUM(SALARY)
FROM EMPLOYEE
WHERE (NAME = 'JOHN' AND SUBURB = 'LIVERPOOL' AND
    STREET = 'STATION');
```

?

- Because the system refuses to reveal the results when the summation is performed over 1 row !

Tracker attacks

- A query predicate T that allows to track down information about a single tuple (row) is called an individual tracker for that tuple (row).
- For example a triple [NAME,SUBURB,STREET] uniquely identifies John in the previous example.
- A general tracker is a predicate that can be used to find the answer to any inadmissible query.
- For example the predicates:

(NAME = 'JOHN' AND SUBURB = 'LIVERPOOL' AND STREET = 'STATION') OR POSITION ='MANAGER'

(NAME = 'JOHN' AND SUBURN = 'LIVERPOOL' AND STREET = 'STATION') OR NOT (POSITION ='MANAGER')

- … are general trackers.

- Let us lot at a more general tracker mechanism.
- Assume that the predicate ϕ_t is an individual tracker.
- Then the system of general trackers can be constructed in the following way:

$Q_1 = \phi_t$ or ϕ_1 or ϕ_2

$Q_2 = \phi_t$ or ϕ_1

$Q_3 = \phi_t$ or ϕ_3

$Q_4 = \phi_2$ or ϕ_3

- … where the results of queries $q:\phi_1$, $q':\phi_2$, $q'':\phi_3$ are mutually disjoint. This means any two pairs have no overlap.
- Using the results of queries v_1, v_2, v_3, v_4, we obtain a system of linear equations:

(1) $f_t + f_1 + f_2 = v_1$

(2) $f_t + f_1 = v_2$

(3) $f_t + f_3 = v_3$

(4) $f_2 + f_3 = v_4$

- The solution is:

(((1)-(2))+(3))-(4): $f_t = (((v_1 - v_2) + v_3) - v_4$

Protecting against inference

- There are two classes of solution.
  – The first is try and design the database in such a way that inference is reduced.
  – The second is to attempt to detect and reject specific queries, or sequences of queries, that would result in an inference channel.
- An inference channel is a path by which unauthorized data is obtained, from authorized data.

In database design…

- We can alter the database structure.
  – For example, splitting a table.
- Or we can change the access control structure.
  – Giving a finer tuned model.
- These approaches are likely to reduce availability. ☹

A splitting problem ...

- Stallings and Brown, 2nd edition, provides an example.
- Consider the table Inventory (Figure 5.6(a))...

Item	Availability	Cost ($)	Department
Shelf support	In-store/online	7.99	Hardware
Lid support	Online only	5.49	Hardware
Decorative chain	In-store/online	104.99	Hardware
Cake pan	Online only	12.99	Housewares
Shower/tub cleaner	In-store/online	11.99	Housewares
Rolling pin	In-store/online	10.99	Housewares

- We want to stop Item and Cost being viewed together, so we create some views.

```
CREATE VIEW V1 AS(
  SELECT Availability, Cost
  FROM Inventory
  WHERE Department = 'Hardware');
```

```
CREATE VIEW V2 AS(
  SELECT Item, Department
  FROM Inventory
  WHERE Department = 'Hardware');
```

Stallings and Brown, 2nd edition, Figure 5.6(b)

- There is no functional dependence between Item and Cost, so giving both of those views to someone who isn't allowed to see the relationship between Item and Cost, won't have that relationship.
- But, the database itself maintains row order on the views, so someone knowing the structure of Inventory table can determine the relation between Item and Cost, and recover the first three rows of the Inventory table.
  - That's inference ☹

Protection at the query level...

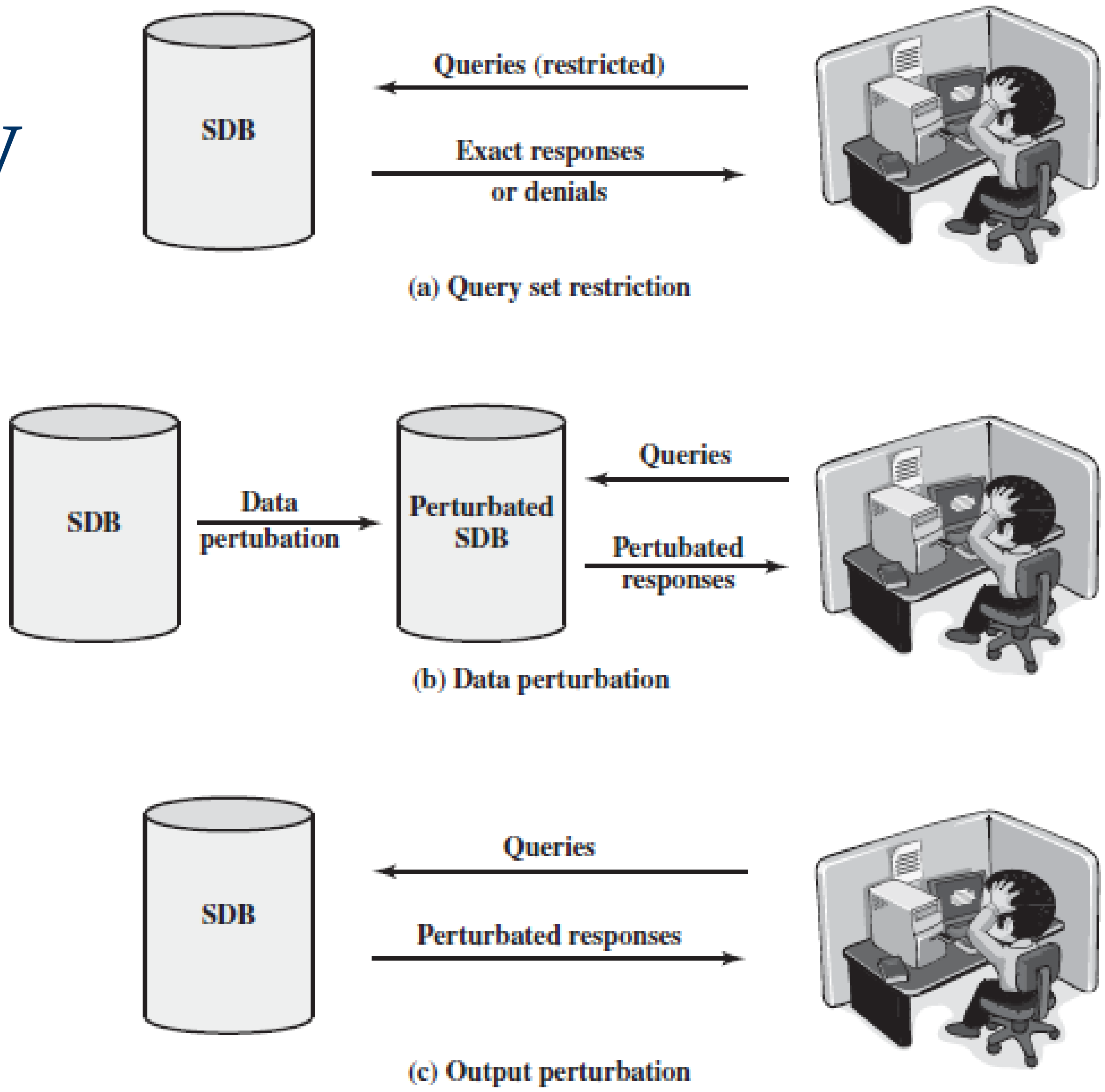


Figure 5.8 Approaches to Statistical Database Security

Query set restriction…

- Suppress sensitive information.
  - Do not disclose an answer when a query set is too small, or too large.
- We have seen already that this doesn't necessarily work.
- We can still sometimes, actually most of the time, construct trackers.

- Consider a database with N rows or records.
- A query q(C) is permitted only if the number of records matching C satisfies…

$$k \leq |X(C)| \leq N - k$$

- … where k is a fixed integer greater than 1.

Data perturbation

- The data in the database is changed, in such a way that the statistics that are generated are still accurate, but inferential information about characteristics on individual rows is inaccurate.
- How can you calculate the average mark in a class if you don’t trust anybody enough to give them your mark?

- How can we perturb data?
- One method is data-swapping.
- Another is to analyse the confidential data and construct a distribution which seems to represent it.
- Then sample from the distribution to construct fake data, for use in a modified database, which is statistically consistent with the original.
  - It will be pretty easy to generate global statistics that are consistent but for subsets of the data we are likely to obtain inaccurate results.

Output perturbation

- This is similar to data perturbation but here we distort the statistical output.
- We want the results to be fairly accurate, but inference to leak little information about the individual data.
- One technique is the Random-sample query method.
  - Here an appropriate subset of the query set that the statistic would be calculated on is determined, and the statistic is calculated on that.
- Alternatively the statistical result on the real query set can itself be changed, likely in a randomized way.

CSCI262
System Security (**S8c**)

SQL injection attacks

SQL Injection

- Part of #1: Injection on the OWASP Top Ten.
  - Open Web Application Security Project.
- There are different types of SQL injection attacks, and the classification vary a fair bit.
  - The classifications are on different characteristics, how/where …
  - There are distinctions between integer and string based injections, and between where the information goes, and …
- However, the underlying principles are the same:
  - Something is entered somewhere, by the client, in the context of SQL, causing something not nice to happen, such as meaning some confidential information to be able to be extracted.

Some simple SQL attacks…

- The most common type of SQL Injection attacks are manipulation attacks where existing SQL statements are modified:
  - Adding elements to the WHERE clause, or
  - Extending the SQL statement
  - Or …

Username	Bob

At the front

Behind the scenes

```
SELECT * FROM users
WHERE username = <The entered username>
```

Username	Bob’ OR ‘a’=‘a

```
SELECT * FROM users
WHERE username = ‘Bob’ OR ‘a’=‘a’
```

process_login.asp

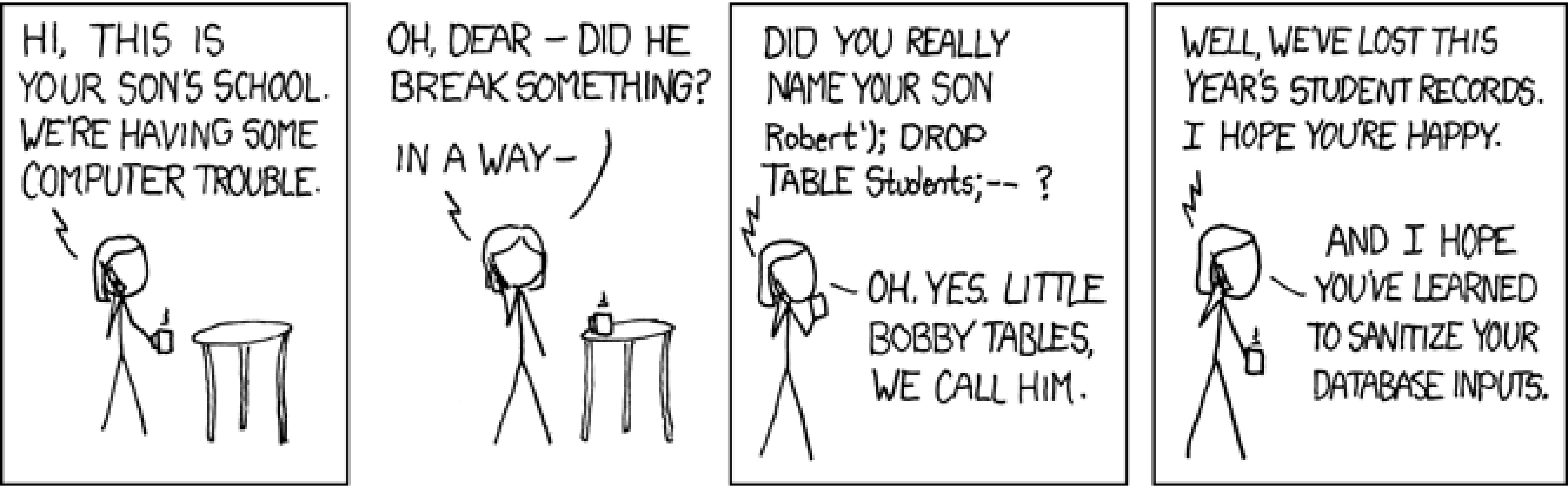
Application Service provider:
An online interface to an application.

```
var sql = "select * from users where
  username = '" + username + "' and
  password = '" + password + "'";
```

If the user specifies the following:

`Username: '; drop table users;--`
`Password:`

- Add additional SQL statements or commands to the existing SQL statement (code injection)
- The -- means the rest is treated as a comment and ignored.
- ..the 'users' table will be deleted, denying access to the application for all users.



http://xkcd.com

- This closes the name selection again but runs another command …

`DROP Table Students`

Function Call Injection

- Function call injection is the insertion of Oracle database functions or custom functions into a vulnerable SQL statement.
- Oracle, for example, has a lot of functions supplied in standard database packages.
- These function calls can be used to make operating system calls or manipulate data in the database.
- Any custom function or function residing in a custom package can also be executed in a SQL statement.

```
SELECT TRANSLATE('user input',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ', '0123456789')
FROM dual;
```

```
SELECT TRANSLATE('' ||
myappadmin.adduser('admin', 'newpass') || '',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ', '0123456789')
FROM dual;
```

Protection against SQL injection

- Protection can be provided by disciplined programming, protecting every dynamic statement.
- Input validation is pretty important too

```
$name = $_REQUEST['name'];
$query = "SELECT * FROM suppliers WHERE name = '" . $name . "';";
$result = mysql_query($query);
```

(a) Vulnerable PHP code

```
$name = $_REQUEST['name'];
$query = "SELECT * FROM suppliers WHERE name = '" .
mysql_real_escape_string($name) . "';";
$result = mysql_query($query);
```

(b) Safer PHP code

Figure 11.3 **SQL Injection Example**

Bind variables

- Rather than using literal values in SQL statements we use a placeholder, a bind variable, and replace that with the actual values using a separate API call.
  - Bind variables are of the form ?, :name, or @name, which depends on the SQL database server used.

Listing 19. Prepared statements with bind variables

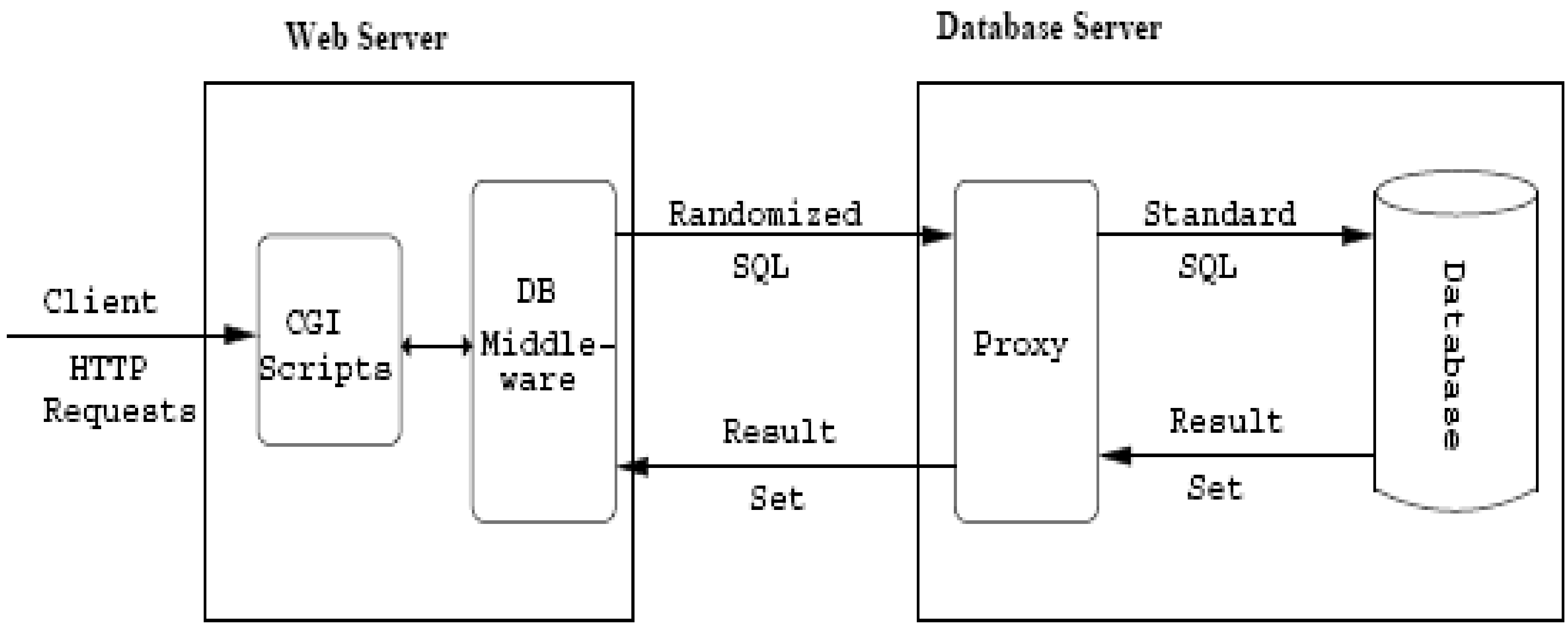
```
String selectStatement = "SELECT * FROM User WHERE userId = ? ";
PreparedStatement prepStmt = con.prepareStatement(selectStatement);
prepStmt.setString(1, userId);
ResultSet rs = prepStmt.executeQuery();
```

http://www.ibm.com/developerworks/library/se-bindvariables/

  - When a bind variable is passed as an argument to an SQL prepared statement, it is automatically escaped by the database driver. The resulting escaped strings treat the variable as user data and cannot be interpreted by the SQL database server as an SQL statement.

SQL Rand

- Invented by Boyd and Keromytis (2004) as a method of protecting against SQL injection attacks.
- It involves instruction set randomization, wherein a random key is added to SQL keywords (internally).
- The externally generated attack queries won't have the random key so the queries won't be carried out.
- Before the keywords are actually sent to the database the random key is removed.



Web Server
Database Server
Client
HTTP
Requests
CGI
Scripts
DB
Middle-
ware
Randomized
SQL
Result
Set
Proxy
Standard
SQL
Result
Set
Database

```
select gender, avg(age)
   from cs101.students
      where dept = %d
   group by gender
```

Randomizing

```
select123 gender, avg123 (age)
   from123 cs101.students
      where123 dept = %d
   group123 by123 gender
```