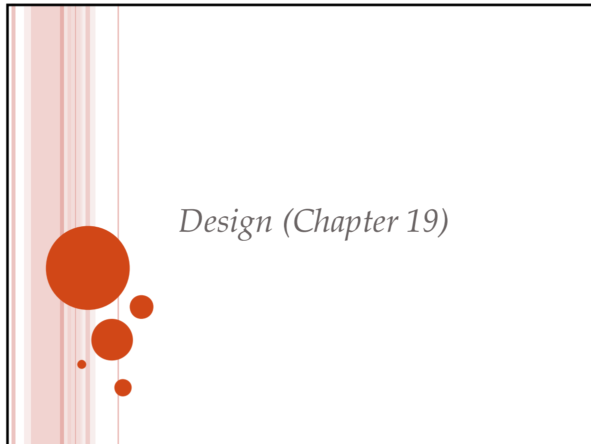


SFU SIMON FRASER UNIVERSITY

Introduction

- Typically, the first database design uses a high-level database model such as the ER model
  - This model is then translated into a relational schema
- Sometimes a relational database schema is developed directly without going through the high-level design
- Either way, the initial relational schema usually has room for improvement, in particular by *eliminating redundancy*
  - Redundancies lead to undesirable update and deletion *anomalies*
- Relational database design theory introduces
  - various *normal forms* avoid various types of redundancies
  - algorithms to convert a relational schema into these normal forms

SFU SIMON FRASER UNIVERSITY

Functional Dependency

- *Normal forms* are based on the concept of functional dependencies between sets of attributes
- A *functional dependency (FD)* $X \rightarrow Y$ is an assertion about a relation R that whenever two tuples of R agree on all the attributes of set X, then they must also agree on all attributes in set Y

Location	Store	Product	Price
Coquitlam	FutureShop	Phone	10$
Burnaby	FutureShop	Phone	10$
Vancouver	FutureShop	Phone	10$
Coquitlam	BestBuy	Phone	20$
Burnaby	BestBuy	Phone	20$
Vancouver	BestBuy	Phone	20$

Functional Dependency

- We say "$X \rightarrow Y$ holds in R."
- Convention:
  - X, Y, Z represent sets of attributes of relation R
  - A, B, C,… represent single attributes of R.
  - no parentheses to denote sets of attributes, just ABC, rather than $\{A,B,C\}$.
- A FD $X \rightarrow Y$ is called *trivial* if $Y \subseteq X$.

Location	Store	Product	Price
Coquitlam	FutureShop	Phone	10$
Burnaby	FutureShop	Phone	10$
Vancouver	FutureShop	Phone	10$
Coquitlam	BestBuy	Phone	20$
Burnaby	BestBuy	Phone	20$
Vancouver	BestBuy	Phone	20$

Splitting / Combining Rule

- $X \rightarrow A_1A_2 \ldots A_n$ holds for R if and only if each of $X \rightarrow A_1$, $X \rightarrow A_2$,…, $X \rightarrow A_n$ hold for R.
  - i.e.: knowing X, we know A_1 and A_2
  - Example: The FD $A \rightarrow BC$ is equivalent to the two FDs $A \rightarrow B$ and $A \rightarrow C$
- This rule can be used to
  - *split* a FD into multiple ones with singleton right sides
  - *combine* multiple singleton right side FDs into one FD
- There is no splitting /combining rule for left sides.
- We'll generally express FDs with *singleton right sides*.

Functional Dependency

- Consider the relation
  Movies1
  (title, year, length, genre, studioName, starName)
- title year → length genre studioName
  - Holds assuming that there are not two movies with the same title in the same year.
- title year → starName
  - does not hold, since a movie can have more than one star acting
- A FD makes an assertion about *all possible instances* of a relation, not only about one (such as the current) instance

KEYS

- Given a relation R with attributes $X = \{A_1, \ldots, A_n\}$.
  - is a *superkey* for relation R if K functionally determines X, i.e. $K \rightarrow X$
- K is a *key* for R
  - if K is a superkey, but no proper subset of K is a superkey
- Keys are a special case of a FD.
- Keys can be deduced systematically, if all FDs for relation R are given.

KEYS

- {title, year, starName} is a *superkey* of Movies1, since
  title → title,
  year → year,
  title year → length,
  title year → genre,
  title year → studioName,
  starName → starName.
- Remember that title year → starName does not hold.
- {title, year}, {year, starName} and {title, starName} are *not* superkeys.
- Thus, {title, year, starName} is a *key* of Movies1.

CLOSURE OF ATTRIBUTES

- Given a set of attributes $\{A_1, \ldots, A_n\}$ and a set S of FDs.
- The *closure* of $\{A_1, \ldots, A_n\}$ under S is the set of attributes X such that every relation that satisfies all the FDs in S also satisfies $\{A_1, \ldots, A_n\} \rightarrow X$, i.e. $\{A_1, \ldots, A_n\} \rightarrow X$ follows from the FDs in S.
- The closure of set Y is denoted by Y^+.
- Example
  attribute set {A, B, C}
  FDs {AB→D, D→E, BC→F, G→H}
  $\{A,B,C\}^+ = \{A, B, C, D, E, F\}$

Closure of Attributes

- Given a set of attributes $\{A_1, \ldots, A_n\}$ and a set S of FDs.
- If necessary, apply the splitting rule to the FDs in S.
- Initialize X to $\{A_1, \ldots, A_n\}$.
- Repeat
  search for some FD $B_1, \ldots, B_m \rightarrow C$ in S
  such that for all $i: B_i \in X$ and $C \notin X$
  and add C to the set X
  until no more attribute C can be added.
- Now $X = \{A_1, \ldots, A_n\}^+$, return X.

Closure of Attributes

- Given a set of attributes $\{A, B, C, D, E, F\}$
  and FDs {AB→ C, BC→ AD, D→ E, CF→B}.
- What is $\{A,B\}^+$?
- Apply the splitting rule: split BC→ AD into BC→ A and BC→ D.
- Initialize X = {A,B} .
- Iterations
  apply AB→ C, X = {A,B,C}
  apply BC→ D, X = {A,B,C,D}
  apply D→ E, X = {A,B,C,D,E}
- Return $\{A,B\}^+$ = {A,B,C,D,E}.

Relational Schema Design

- Goal of relational schema design is to avoid anomalies.
- Redundancies lead to certain forms of anomalies. and redundancy.
- *Update anomaly*
  - one occurrence of a fact is changed, but not all occurrences
- *Deletion anomaly*
  - valid fact is lost when a tuple is deleted
- In the following example, consider the relation

Movies1
(title, year, length, genre, studioName, starName).

SFU SIMON FRASER UNIVERSITY

Relational Schema Design

title	year	length	genre	studioName	starName
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone with the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

- Update anomaly: update the length to 125 (only) for the first Star Wars tuple
- Deletion anomaly: delete Vivien Leigh → we loose all information about *Gone with the Wind*

SFU SIMON FRASER UNIVERSITY

Example

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

- Problems due to R→W :
  - *Update anomaly*: Can we change W in just the 1st tuple of SNLRWH?
  - *Insertion anomaly*: What if we want to insert an employee and don't know the hourly wage for his rating?
  - *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

Hourly_Emps2

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Wages

R	W
8	10
5	7

Will 2 smaller tables be better?

Decomposing Relations

SFU SIMON FRASER UNIVERSITY

Decomposing Relations

- How to eliminate these anomalies? Decompose the relation into two or more relations that together have the same attributes.
  - Decomposition *MUST* be lossless.
- Given relation $R\ \{A_1, \ldots, A_n\}$. A *decomposition* of R consists of two relations $S\ \{B_1, \ldots, B_m\}$ and $T\ \{C_1, \ldots, C_k\}$ such that $\{A_1,\ldots,A_n\} = \{B_1,\ldots,B_m\} \cup \{C_1,\ldots,C_k\}$
- Decompose Movies1

  (title, year, length, genre, studioName, starName)

into Movies2 (title, year, length, genre, studioName) and Movies3 (title, year, starName).

SFU SIMON FRASER UNIVERSITY

Decomposing Relations

title	year	length	genre	studioName	starName
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone with the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Movies1

title	year	length	genre	studioName
Star Wars	1977	124	SciFi	Fox
Gone with the Wind	1939	231	drama	MGM
Wayne's World	1992	95	comedy	Paramount

Movies2

→ The update and deletion anomalies are gone!

title	year	starName
Star Wars	1977	Carrie Fisher
Star Wars	1977	Mark Hamill
Star Wars	1977	Harrison Ford
Gone with the Wind	1939	Vivien Leigh
Wayne's World	1992	Dana Carvey
Wayne's World	1992	Mike Meyers

Movies3

SFU SIMON FRASER UNIVERSITY

Normalization

- A logical design method which minimizes data redundancy and reduces design flaws.
- Consists of applying various "normal" forms to the database design.
- The normal forms break down large tables into smaller subsets.

First Normal Form (1NF)

- Each attribute must be atomic
  - No repeating columns within a row.
  - No multi-valued columns.
- 1NF simplifies attributes
  - Queries become easier.

1NF

Employee (unnormalized)

emp_no	name	dept_no	dept_name	skills
1	Kevin Jacobs	201	R&D	C, Perl, Java
2	Barbara Jones	224	IT	Linux, Mac
3	Jake Rivera	201	R&D	DB2, Oracle, Java

Employee (1NF)

emp_no	name	dept_no	dept_name	skills
1	Kevin Jacobs	201	R&D	C
1	Kevin Jacobs	201	R&D	Perl
1	Kevin Jacobs	201	R&D	Java
2	Barbara Jones	224	IT	Linux
2	Barbara Jones	224	IT	Mac
3	Jake Rivera	201	R&D	DB2
3	Jake Rivera	201	R&D	Oracle
3	Jake Rivera	201	R&D	Java

Functional Dependence

Employee (1NF)

emp_no	name	dept_no	dept_name	skills
1	Kevin Jacobs	201	R&D	C
1	Kevin Jacobs	201	R&D	Perl
1	Kevin Jacobs	201	R&D	Java
2	Barbara Jones	224	IT	Linux
2	Barbara Jones	224	IT	Mac
3	Jake Rivera	201	R&D	DB2
3	Jake Rivera	201	R&D	Oracle
3	Jake Rivera	201	R&D	Java

- Name, dept_no, and dept_name are functionally dependent on emp_no.
  - (emp_no -> name, dept_no, dept_name)
- Skills is not functionally dependent on emp_no since it is not unique to each emp_no.

Data Integrity

Employee (1NF)

emp_no	name	dept_no	dept_name	skills
1	Kevin Jacobs	201	R&D	C
1	Kevin Jacobs	201	R&D	Perl
1	Kevin Jacobs	201	R&D	Java
2	Barbara Jones	224	IT	Linux
2	Barbara Jones	224	IT	Mac
3	Jake Rivera	201	R&D	DB2
3	Jake Rivera	201	R&D	Oracle
3	Jake Rivera	201	R&D	Java

- Insert Anomaly - adding null values. eg, inserting a new department does not require the primary key of emp_no to be added.
- Update Anomaly - multiple updates for a single name change, causes performance degradation. eg, changing IT dept_name to IS
- Delete Anomaly - deleting wanted information. eg, deleting the IT department removes employee Barbara Jones from the database

Second Normal Form (2NF)

- Each attribute must be functionally dependent on the primary key.
  - Functional dependence - the property of one or more attributes that uniquely determines the value of other attributes.
  - Any non-dependent attributes are moved into a smaller (subset) table.
- 2NF improves data integrity.
  - Prevents update, insert, and delete anomalies.

2NF

Employee (1NF)

emp_no	name	dept_no	dept_name	skills
1	Kevin Jacobs	201	R&D	C
1	Kevin Jacobs	201	R&D	Perl
1	Kevin Jacobs	201	R&D	Java
2	Barbara Jones	224	IT	Linux
2	Barbara Jones	224	IT	Mac
3	Jake Rivera	201	R&D	DB2
3	Jake Rivera	201	R&D	Oracle
3	Jake Rivera	201	R&D	Java

Employee (2NF)

emp_no	name	dept_no	dept_name
1	Kevin Jacobs	201	R&D
2	Barbara Jones	224	IT
3	Jake Rivera	201	R&D

Skills (2NF)

emp_no	skills
1	C
1	Perl
1	Java
2	Linux
2	Mac
3	DB2
3	Oracle
3	Java

SFU SIMON FRASER UNIVERSITY

Transitive Dependence

Employee (2NF)

emp_no	name	dept_no	dept_name
1	Kevin Jacobs	201	R&D
2	Barbara Jones	224	IT
3	Jake Rivera	201	R&D

- Dept_no and dept_name are functionally dependent on emp_no however, department can be considered a separate entity.

SFU SIMON FRASER UNIVERSITY

Third Normal Form (3NF)

- Remove transitive dependencies.
  - Transitive dependence - two separate entities exist within one table.
  - Any transitive dependencies are moved into a smaller (subset) table.
- 3NF further improves data integrity.
  - Prevents update, insert, and delete anomalies.

SFU SIMON FRASER UNIVERSITY

3NF

Employee (2NF)

emp_no	name	dept_no	dept_name
1	Kevin Jacobs	201	R&D
2	Barbara Jones	224	IT
3	Jake Rivera	201	R&D

Employee (3NF)

emp_no	name	dept_no
1	Kevin Jacobs	201
2	Barbara Jones	224
3	Jake Rivera	201

Department (3NF)

dept_no	dept_name
201	R&D
224	IT

Other Normal Forms

- Boyce-Codd Normal Form (BCNF)
  - Strengthens 3NF by requiring the keys in the functional dependencies to be superkeys (a column or columns that uniquely identify a row)
- Fourth Normal Form (4NF)
  - Eliminate trivial multivalued dependencies.
- Fifth Normal Form (5NF)
  - Eliminate dependencies not determined by keys.

Normalizing our team (1NF)

games

game_id	date	opponent	result
34	6/3/05	Chicago	W
35	6/8/05	Seattle	W
40	6/15/05	Phoenix	L
42	6/20/05	LA	W

sales

sales_id	game_id	merch	tickets
120	34	5000	25000
122	35	4500	30000
125	40	2500	15000
126	42	6500	40000

players

player_id	game_id	name	start_date	end_date	aces	blocks	spikes	digs
45	34	Mike Speedy	1/1/00		12	3	20	5
45	35	Mike Speedy	1/1/00		10	2	15	4
45	40	Mike Speedy	1/1/00		7	2	10	3
78	42	Frank Newmon	5/1/05					
102	34	Joe Powers	1/1/02	7/1/05	8	6	18	10
102	35	Joe Powers	1/1/02	7/1/05	10	8	24	12
103	42	Tony Tough	1/1/05		15	10	20	14

Normalizing our team (2NF & 3NF)

games

game_id	date	opponent	result
34	6/3/05	Chicago	W
35	6/8/05	Seattle	W
40	6/15/05	Phoenix	L
42	6/20/05	LA	W

sales

sales_id	game_id	merch	tickets
120	34	5000	25000
122	35	4500	30000
125	40	2500	15000
126	42	6500	40000

players

player_id	name	start_date	end_date
45	Mike Speedy	1/1/00	
78	Frank Newmon	5/1/05	
102	Joe Powers	1/1/02	7/1/05
103	Tony Tough	1/1/05	

player_stats

player_id	game_id	aces	blocks	spikes	digs
45	34	12	3	20	5
45	35	10	2	15	4
45	40	7	2	10	3
102	34	8	6	18	10
102	35	10	8	24	12
103	42	15	10	20	14

Benefits of Database Normalization

- Decreased storage requirements!

	1 VARCHAR(20)
converted to	1 TINYINT UNSIGNED
in a table of	1 million rows
is a savings of	~20 MB

- Faster search performance!
  - Smaller file for table scans.
  - More directed searching.
- Improved data integrity!

Boyce-Codd Normal Form

- There are many ways of decomposing a relation.
- Which decompositions leads to an anomaly-free relation?
- Boyce-Codd normal form defines a condition under which the anomalies discussed so far cannot exist.
- A relation R is in *Boyce-Codd normal form* (BCNF), if and only if the following condition holds:
  - for every non-trivial FD $A_1 \dots A_n \rightarrow B_1 \dots B_m$ for R $\{A_1, \dots, A_n\}$ is a superkey for R.
    - I.e., the left side of a FD needs to contain a key.

Boyce-Codd Normal Form

- Consider
  Movies1
  (title, year, length, genre, studioName, starName).
- We have the FD
  - title year → length genre studioName
  - The left side of this FD is not a superkey, i.e. Movies1 is not in BCNF

title year is not unique for all tuples

title	year	length	genre	studioName	starName
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone with the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Boyce-Codd Normal Form

- Consider

  Movies2

  (title, year, length, genre, studioName).

title	year	length	genre	studioName
Star Wars	1977	124	SciFi	Fox
Gone with the Wind	1939	231	drama	MGM
Wayne's World	1992	95	comedy	Paramount

- {title, year } is its only key, since neither

  title $\rightarrow$ length genre studioName nor

  year $\rightarrow$ length genre studioName hold.

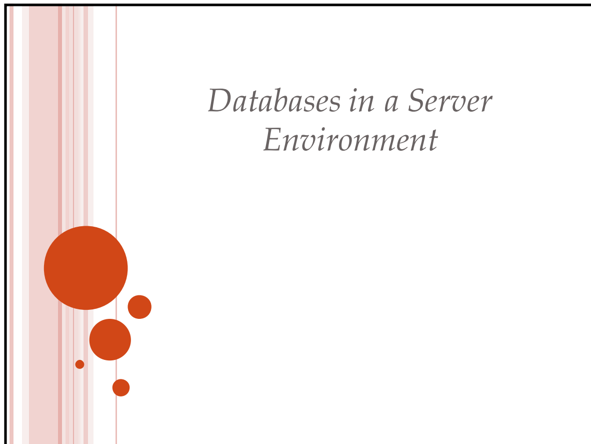
- All non-trivial FDs must have at least title and year on the left side. Thus, Movies2 is in BCNF.

Problems with Decompositions

- There are three potential problems to consider:
  - Some queries become more expensive.
  - Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
  - Checking some dependencies may require joining the instances of the decomposed relations.
- *Tradeoff*: Must consider these issues vs. redundancy.

Summary

- A functional dependency (FD) states that two tuples that agree on some set of attributes also agree on another attribute set.
- Keys are special cases of functional dependencies.
- Redundancies in a relational table lead to anomalies such as update and deletion anomalies.
- A relation is in Boyce-Codd normal form (BCNF), if the left sides of all non-trivial FDs contain a key.
- A schema in BCNF avoids the above anomalies.
- A given schema can be decomposed into subsets of attributes such that the resulting tables are all in BCNF and the join of these tables recovers the original table.



Introduction

- So far:
  - interactive SQL interface,
  - pure "SQL programs".
- In practice often:
  - queries are not ad-hoc, but programmed once and executed repeatedly,
  - need the greater flexibility of a general-purpose programming language, especially for complex calculations (e.g. recursive functions) and graphic user interfaces.
    - SQL statements part of a larger software system

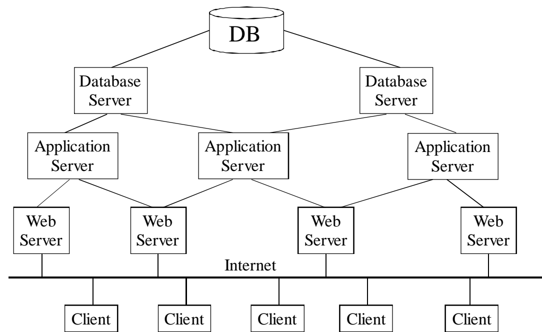
The Three-Tier Architecture

- The following *three-tier architecture* is common for database installations:
  - *Web servers* connect clients to the DBS, typically over the Internet (*web-server tier*).
  - *Applications servers* perform the "business logic" requested by the webserves, supported by the database servers (*application tier*).
  - *Database servers* execute queries and modifications of the database for the application servers (*database tier*).

The Three-Tier Architecture

- Multiple processes can be run on the same processor. E.g., web server, application server and database server all on the same processor.
- This is common in small systems.
- In large-scale systems, however, there are usually many processors running processes corresponding to the same "server", i.e. the same program.
- For example, many processors running application server processes and other processors running web server processes.

The Three-Tier Architecture



The Web-Server Tier

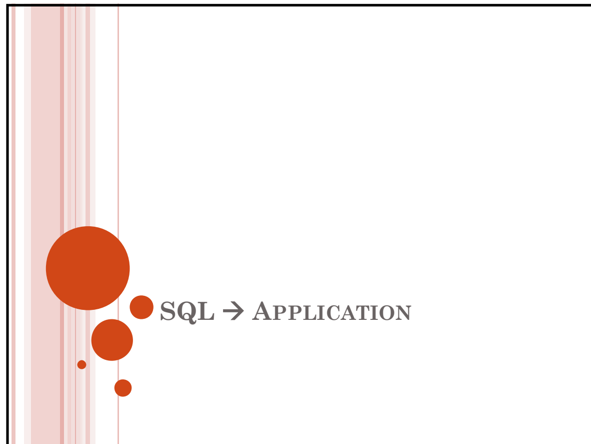
- When a user makes a request on the Internet, a web server responds.
- The user becomes a *client* of that server.
- Example Amazon.com
  - User enters www.amazon.com in browser.
  - Web server presents Amazon homepage.
  - User enters book title and starts search.
  - The web server responds to the user request, using the services of an application server.

The Application Tier

- The application tier receives requests from the web-server tier and turns data from the database tier into answers to web server requests.
- Example Amazon.com
  - Web server requests book with given title from application server.
  - Application server sends corresponding SQL query to database server.
  - Database server returns a (set of) tuple(s).
  - Application server assembles the resulting tuple(s) into an HTML page and sends it to the web server.

The Database Tier

- The database tier execute queries issued by the application tier and returns the corresponding results to the application tier.
- Example Amazon.com
  - Application server sends SQL query searching for a book to database server.
  - Database server executes SQL query and returns a (set of) tuple(s).
  - Application server imports a batch of new books over night and database server performs the corresponding SQL modifications.



SFU SIMON FRASER UNIVERSITY

IMPEDANCE MISMATCH PROBLEM

- We want to exchange data in both directions between a DBS and an application program.
- SQL relations are (multi-) sets of records, with no *a priori* bound on the number of records
  - Usually the application doesn't know the number of records returned
- No data structure traditionally exist in procedural programming languages to work with sets
  - Programming languages are record-oriented
- Programming languages have data types that are not available in SQL
  - Can cast SQL variables to other types

46

SFU SIMON FRASER UNIVERSITY

IMPEDANCE MISMATCH PROBLEM

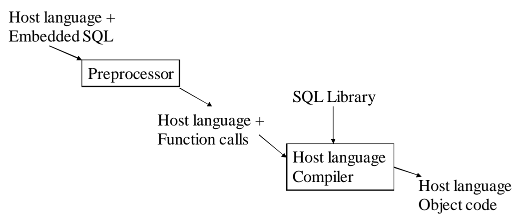
- There are three alternative approaches to integrate DBS and application program.
  - Embed SQL in the host programming language
    Embedded SQL, Dynamic SQL.
  -
  - Store program code in DBS
    Stored procedures.
  - Create special API to call SQL commands
    JDBC for Java.

47

SFU SIMON FRASER UNIVERSITY

EMBEDDED SQL

- Approach: Embed SQL in the host language.
  - A *preprocessor* converts the SQL statements into special API calls.
  - Then a regular compiler is used to compile the code.



48

Embedded SQL

- Embedded SQL constructs:
  - Connecting to a database:
    EXEC SQL CONNECT
  - Declaring shared variables:
    EXEC SQL BEGIN (END) DECLARE SECTION
  - SQL Statements:
    EXEC SQL Statement;
    all statements except queries can be directly embedded
  - Declaring and manipulating cursors
    for embedding SQL queries

Shared Variables

- Definition of shared variables (e.g., host language C)

```
EXEC SQL BEGIN DECLARE SECTION
  char c_sname[20];
  long c_sid;
  short c_rating;
  float c_age;
EXEC SQL END DECLARE SECTION
```

- Two special "error" variables:
  - SQLCODE (long, is negative if an error has occurred)
  - SQLSTATE (char[6], predefined codes for common errors, e.g. '02000' = no tuple found)

Shared Variables

- A shared variable can be used in an SQL statement instead of some constant.
- When using a shared variable, its name must be preceeded by a colon (:).

```
EXEC SQL
INSERT INTO Sailors (sid, sname, rating, age)
VALUES (: c_sid, :c_sname, : c_rating, : c_age);
```

Cursors

- Programs work on *single instances* data
- SQL returns *sets* of data
- What do you use to point to something specific?
  - CURSOR!
- Use cursor to point at specific tuple in a set
- What was there before cursors were implemented?

Cursors

- Can declare a cursor on any query statement.
- Can *open* a cursor, and repeatedly *fetch* next tuple, until all tuples have been retrieved.
- Can also *modify / delete* tuple pointed to by a cursor.
- Can *close* cursor so that it is no longer accessible.

Cursors

- *Find names of sailors who've reserved a red boat, in alphabetical order.*

```
EXEC SQL DECLARE sinfo CURSOR FOR
   SELECT  S.sname
   FROM  Sailors S, Boats B, Reserves R
   WHERE  S.sid=R.sid AND R.bid=B.bid
               AND B.color='red'
   ORDER BY  S.sname;
```

CURSORS

- Default cursors start from the first tuple and fetch all tuples in order.
- *Scrollable cursors* provide much more flexibility:
  - FIRST / LAST: direct access of first / last tuple
  - PRIOR: scroll backward
  - RELATIVE c: scroll c tuples forward / backward
  - ABSOLUTE c: random access of the c-th tuple.
- Declared by DECLARE . . . SCROLL CURSOR.

CURSORS

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
   char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
   c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
   SELECT S.sname, S.age
   FROM Sailors S
   WHERE S.rating > :c_minrating
   ORDER BY S.sname;
do {
   EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
   printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```


DYNAMIC SQL

- Often, the concrete SQL statement is known not at compile time, but only at runtime.
  - Example 1: a program prompts user for parameters of SQL query, reads the parameters and executes query.
  - Example 2: a program prompts user for an SQL query, reads and executes it.
- Construction of SQL statements on-the-fly:
  PREPARE: parse and compile SQL command.
  EXECUTE: execute command.

Dynamic SQL

- PREPARE parses string, converts it into SQL statement and generates query plan.
- Query plan is returned as result of the PREPARE statement.
- Same query plan can be executed multiple times.

```
EXEC SQL BEGIN DECLARE SECTION;
   char *query;
EXEC SQL END DECLARE SECTION;
   /* prompt user for a query and let :query point to it
EXEC SQL PREPARE SQLquery FROM :query;
while (. . .) {
   EXEC SQL EXECUTE SQLquery;}
```


Stored Procedures

- A *stored procedure* is a function / procedure written in a general-purpose programming language that is executed within the DBS.
- Allows to perform computations that cannot be expressed in SQL.
- Procedure executed through a single SQL statement.
- Executed in the process space of the DB server.
- SQL standard: *PSM* (Persistent Stored Modules). Extends SQL by basic concepts of a general-purpose programming language.

Stored Procedures

- Advantages:
  - Can encapsulate application logic while staying “close” to the data.
  - Reuse of application logic by different users / application programs.
  - Avoid (possibly inefficient) tuple-at-a-time return of query results through cursors.

Programming Stored Procedures

- Format of a *procedure* declaration:

```
CREATE PROCEDURE <name> (<parameters>)
        <local declarations>
        <procedure body>;
```

- Format of a *function* declaration:

```
CREATE FUNCTION <name> (<parameters>) RETURNS
<type>          <local declarations>
        <procedure body>;
```

- *Parameters* can have three different *modes*: IN, OUT, or INOUT.

Programming Stored Procedures

- Examples:

```
CREATE PROCEDURE ShowNumReservations
  SELECT S.sid, S.sname, COUNT(*)
  FROM Sailors S, Reserves R
  WHERE S.sid = R.sid
  GROUP BY S.sid, S.sname;

CREATE PROCEDURE IncreaseRating(
  IN sailor_sid INTEGER, IN increase INTEGER)
  UPDATE Sailors
  SET rating = rating + increase
  WHERE sid = sailor_sid;
```


Programming Stored Procedures

- Declaration of local variables

```
DECLARE <name> <type>
```

- Assignment statements

```
SET <variable> = <expression>
```

- Conditional statement

```
IF <condition> THEN <statement list>
ELSEIF <condition> THEN <statement list> . . .
ELSE <statement list> END IF;
```

- Statement groups

```
BEGIN <statement 1>; <statement 2>; . . .;
END
```

Programming Stored Procedures

- Loop statements

```
LOOP <statement list>
END LOOP;
```

- Statements can be labeled
  - prefix the statement by <label name> :
- Breaking out of a loop

```
LEAVE <loop label>;
```

- Return conditions of queries can be named

```
DECLARE <name> CONDITION FOR SQLSTATE
   <value>;
```


Example of a Stored Procedure

```
CREATE PROCEDURE UpdateEveryOtherSailor
   DECLARE NotFound CONDITION FOR SQLSTATE '02000';
   DECLARE SailorCursor CURSOR FOR SELECT * FROM Sailors;
   DECLARE number INTEGER;
   BEGIN
        SET number = 0;
        OPEN SailorCursor;
        sailorLooP: LOOP
              FETCH SailorCursor INTO . . .;
              IF NotFound THEN LEAVE sailorLooP  END IF;
              IF number = 0 THEN BEGIN
                        UPDATE Sailor
                        SET rating = 2 * rating
                        WHERE CURRENT OF SailorCursor;
                        number = 1;
                   END
                   ELSE SET number = 0;
              END IF;
        END LOOP;
        CLOSE SailorCursor;
   END;
```


Programming Stored Procedures

- Stored procedures can also be written directly in a general-purpose programming language.
- Example

```
CREATE PROCEDURE TopSailors(IN num
   INTEGER)
LANGUAGE JAVA
EXTERNAL NAME "file:///c:/storedProcs/rank.jar"
```

Calling Stored Procedures

- A stored procedure / function can be called from
  - an Embedded SQL program,
  - a stored procedure / function (possibly recursive call),
  - interactive SQL statements.
- Example

  Definition

```
CREATE FUNCTION MinRating RETURNS INTEGER
SELECT MIN(rating)
FROM Sailors S;
```

  Call

```
INSERT  INTO SailorStatistics(minRating)
VALUES (MinRating);
```


DB Call Level Interfaces

- Rather than modify compiler, add library with database calls (API) and call their methods from program.
- Special standardized interface: procedures/objects.
- Pass SQL strings from programming language, present result sets in a language-friendly way.
- Examples: ODBC, JDBC.

DB Call Level Interfaces

- Approach similar to Embedded SQL, but not so DBMS-dependent.
- In Embedded SQL, the preprocessor and SQL library are DBMS specific, which makes the resulting object code not portable to other DBMS.
- When using API, create DBMS-independent code that can be executed on any DBMS.
- Main idea: a “driver” traps the calls and translates them into DBMS-specific code.

Java Database Connectivity

- Java Database Connectivity (JDBC)
- Integration of DBS with Java programs
- Object-oriented nature

JDBC Architecture

- Four architectural components:
  - Application
    initiates and terminates connections, submits SQL statements
  - Driver manager
    loads JDBC drivers
  - Driver
    connects to data source, transmits requests and returns/translates results and error codes
  - Data source (DBS)
    processes SQL statements

JDBC Architecture

- Four types of drivers
  - Bridge Translates SQL commands into non-native API. Example: JDBC-ODBC bridge. Code for ODBC and JDBC driver needs to be available on each client.
  - Direct translation to native API, non-Java driver
    Translates SQL commands to native API of data source. Need OS-specific binary on each client.
  - Network bridge
    Send commands over the network to a middleware server that talks to the data source. Needs only small JDBC driver at each client.
  - Direction translation to native API via Java driver
    Converts JDBC calls directly to network protocol used by DBMS. Needs DBMS-specific Java driver at each client.

JDBC Classes and Interfaces

Steps to process a database query:

1. Load the JDBC driver
2. Connect to the data source
3. Execute SQL statements

JDBC Driver Management

- All drivers are managed by the *DriverManager* class.
- Loading a JDBC driver:
  - In the Java code:
    Class.forName("oracle/jdbc.driver.Oracledriver");
  - When starting the Java application:
    -Djdbc.drivers=oracle/jdbc.driver

Connections in JDBC

- We interact with a data source through sessions. Each connection identifies a logical session.
- JDBC URL:
  jdbc:<subprotocol>:<otherParameters>
- Example:

```
String url="jdbc:oracle:www.bookstore.com:3083";
Connection con;
try{
    con =
    DriverManager.getConnection(url,usedId,password);
} catch SQLException excpt { …}
```

Executing SQL Statements

- Three different classes of SQL statements:
  - Statement
    both static and dynamic SQL statements
  - PreparedStatement
    semi-static SQL statements
  - CallableStatment
    stored procedures
- PreparedStatement class:
  Precompiled, parametrized SQL statements:
  - Structure is fixed,
  - Values of parameters are determined at run-time.

76

Executing SQL Statements

- Example of prepared statements

```
String sql="INSERT INTO Sailors VALUES(?,?,?,?)";
PreparedStatment pstmt=con.prepareStatement(sql);
pstmt.clearParameters();
pstmt.setInt(1,sid);          // sid is a Java variable
pstmt.setString(2,sname);     // sname is a Java variable
pstmt.setInt(3, rating);      // rating is a Java variable
pstmt.setFloat(4,age);        // age is a Java variable

// we know that no tuples are returned, thus we use
      executeUpdate()
int numTuples = pstmt.executeUpdate();
```

77

Executing SQL Statements

- PreparedStatement.executeUpdate only returns the number of affected records.
- PreparedStatement.executeQuery returns data, encapsulated in a ResultSet object (a cursor).

```
ResultSet rs=pstmt.executeQuery(sql);
// rs is now a cursor
While (rs.next()) {
  // process the data
}
```

78

Executing SQL Statements

- A *ResultSet* is a very powerful cursor:
  - previous(): moves one tuple back
  - absolute(int num): moves to the tuple with the specified number
  - relative (int num): moves forward or backward *num* tuples
  - first() and last(): positions on first / last tuple.
- Use the type-specific *get*-methods to access the attribute values of the current cursor tuple:

  e.g. rs.getString("name");
  rs.getFloat("rating");

Executing SQL Statements

SQL Type	Java class	ResultSet get method
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

Matching Java and SQL data types

Executing SQL Statements

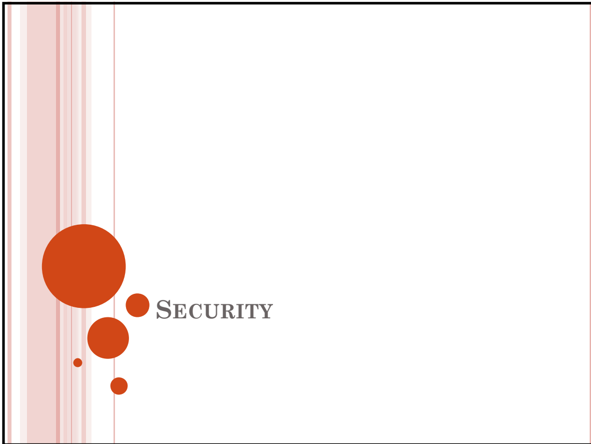
- CallableStatment
  stored procedure
- Calling a stored procedure
  - CallableStatement cstmt= con.prepareCall("{call ShowSailors});
  - ResultSet rs = cstmt.executeQuery();
  - while (rs.next()) {
  - …
  - }

Executing SQL Statements

- JDBC exploits Java's capabilities for dealing with exceptions and warnings.
- JDBC defines SQL-specific subclasses: SQLException and SQLWarning.
- Most methods in package java.sql can throw an SQLException if an error occurs.
- These exceptions need to be handled, unlike the values of the SQLSTATE variable in Embedded SQL that can be ignored.
- SQLWarning is a subclass of SQLException; not as severe (they are not thrown and their existence has to be explicitly tested).

A "Complete" JDBC Example

```
Connection con = // connect
   DriverManager.getConnection(url, "login", "pass");
Statement stmt = con.createStatement(); // set up stmt
String query = "SELECT name, rating FROM Sailors";
ResultSet rs = stmt.executeQuery(query);
try { // handle exceptions
    // loop through result tuples
    while (rs.next()) {
        String s = rs.getString("name");
        Int n = rs.getFloat("rating");
        System.out.println(s + "   " + n);
    }
} catch(SQLException ex) {
    System.out.println(ex.getMessage ()
        + ex.getSQLState () + ex.getErrorCode ());
}
```

SFU SIMON FRASER UNIVERSITY

Security and User Authorization

- Data stored in a DBMS is often vital to the enterprise and needs to be protected from unauthorized access.
  - E.g., banking or health insurance DB
- The *access control* component of the DBMS ensures the security of the DB.
- In SQL, users or user groups are associated with *authorization Ids*. A user must specify an authorization ID and corresponding *authentication information* (e.g., password) before the DBMS accepts his commands.
- A *privilege* grants a user the right to perform certain SQL operations.

SFU SIMON FRASER UNIVERSITY

Introduction to DB Security

- Secrecy: Users should not be able to see things they are not supposed to.
  - E.g., A student can't see other students' grades.
- Integrity: Users should not be able to modify things they are not supposed to.
  - E.g., Only instructors can assign grades.
- Availability: Users should be able to see and modify things they are allowed to.

SFU SIMON FRASER UNIVERSITY

Access Controls

- A security policy specifies who is authorized to do what.
- A security mechanism allows us to enforce a chosen security policy.
- Two main mechanisms at the DBMS level:
  - Discretionary access control
  - Mandatory access control

SFU SIMON FRASER UNIVERSITY

Discretionary Access Control

- Based on the concept of access rights or privileges for objects (tables and views), and mechanisms for giving users privileges (and revoking privileges).
- Creator of a table or a view automatically gets all privileges on it.
  - DMBS keeps track of who subsequently gains and loses privileges, and ensures that only requests from users who have the necessary privileges (at the time the request is issued) are allowed.

SFU SIMON FRASER UNIVERSITY

Privileges

- There are also some other privileges.
- There is no privilege for executing schema manipulation statements (CREATE, ALTER, DROP). They can only be executed by the *schema owner*.
- When creating a DB schema, a user obtains all corresponding privileges.
- Privileges can be granted by a privilege owner (user) to other users and can also be revoked.
- Normally, different users (user groups) have different privileges.

SFU SIMON FRASER UNIVERSITY

Privilege Checking

- Each schema, module (application program) and session has an *associated user* (authorization ID).
- An SQL operation consists of two parts:
  - the database elements accessed,
  - the agent performing the operation.
- The privileges of an agent are those corresponding to the current authorization ID. The *current authorization ID* is either the module authorization ID (if existing) or the session authorization ID.
- An SQL operation may be executed only if the current authorization ID possesses all required privileges.

SFU SIMON FRASER UNIVERSITY

Granting Privileges

- Format of a *grant statement*:

  GRANT <privilege list> ON <database element>
  TO <user list> [WITH GRANT OPTION]
- Privilege list is a list of the above SQL privileges or ALL PRIVILEGES.
- A database element is normally a table, but can also be a domain or other element.
- User list is a list of authorization IDs.
- The *grant option* gives the receiving user the right to grant the privilege further to another user.

SFU SIMON FRASER UNIVERSITY

Privileges

- SQL distinguishes the following privileges:

- SELECT ON <table>: the right to *read* all attributes of <table> and to *add* further attributes.
- INSERT ON <table>: the right to *insert* tuples into <table>.
- DELETE ON <table>: the right to *delete* tuples from <table>.
- UPDATE ON <table>: the right to *update* tuples of <table>.
- REFERENCES (<attribute>) ON <table>: the right to *refer to* <attribute> of <table> in an integrity constraint.
- TRIGGER ON <table>: the right to *define triggers* on <table>.

SFU SIMON FRASER UNIVERSITY

GRANT and REVOKE of Privileges

- GRANT INSERT, SELECT ON Sailors TO Horatio
  - Horatio can query Sailors or insert tuples into it.
- GRANT DELETE ON Sailors TO Yuppy WITH GRANT OPTION
  - Yuppy can delete tuples, and also authorize others to do so.
- GRANT UPDATE (rating) ON Sailors TO Dustin
  - Dustin can update (only) the rating field of Sailors tuples.
- GRANT SELECT ON ActiveSailors TO Guppy, Yuppy
  - This does NOT allow the 'uppies to query Sailors directly!
- REVOKE: When a privilege is revoked from X, it is also revoked from all users who got it solely from X.

SFU SIMON FRASER UNIVERSITY

Example of Privileges

GRANT SELECT ON Sailors TO Michael WITH GRANT OPTION;

- Michael can create the following view:

```
CREATE VIEW YoungSailors (sid, age, rating)
    AS SELECT S.sid, S.age, S.rating
     FROM Sailors S
     WHERE S.age < 18;
```

- Since Michael holds the grant option on the underlying table, he can grant privileges on the view YoungSailors to other users, e.g.
  GRANT SELECT ON YoungSailors TO Eric, Simon;
- Eric and Simon can execute SELECT queries on the view YoungSailors, but not the underlying Sailors table.
- Thus, views are another important way of managing access control.

SFU SIMON FRASER UNIVERSITY

Example of Privileges

- Michael can define another table Sneaky with a table constraint :

```
CREATE TABLE Sneaky (maxrating INTEGER,
        CHECK (maxrating >=
                (SELECT MAX (S.rating)
                FROM Sailors S)))
```

- By repeatedly inserting tuples with increasing maxrating values, Michael can determine the maximum rating value of Sailors.
- In order to avoid such undesired side-effects, SQL requires that a CHECK constraint references only a table for which the user holds a SELECT privilege.

SFU SIMON FRASER UNIVERSITY

GRANT/REVOKE on Views

- If the creator of a view loses the SELECT privilege on an underlying table, the view is dropped!
- If the creator of a view loses a privilege held with the grant option on an underlying table, (s)he loses the privilege on the view as well; so do users who were granted that privilege on the view!

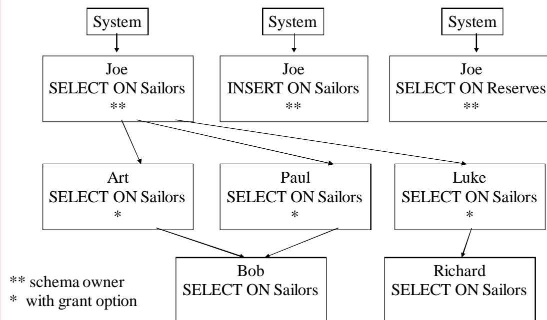
Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given ActiveSailors, but not Sailors or Reserves, we can find sailors who have a reservation, but not the *bid*'s of boats that have been reserved.
- Creator of view has a privilege on the view if (s)he has the privilege on all underlying tables.
- Together with GRANT/REVOKE commands, views are a very powerful access control tool.

Grant Diagrams

- A *grant diagram* (authorization graph) records the privileges of all users and their relationships.
- Nodes represent user privileges. An edge from user1/privilege1 to user2/privilege2 represents the fact that privilege 2 of user 2 was granted by user1 based on his privilege 1.
- It is also recorded whether the user is the schema owner or whether he holds the grant option.
- *Privilege descriptor*: *grantor* (for schema owner: system), *grantee*, *granted privilege* and whether the *grant option* is given.

Grant Diagrams



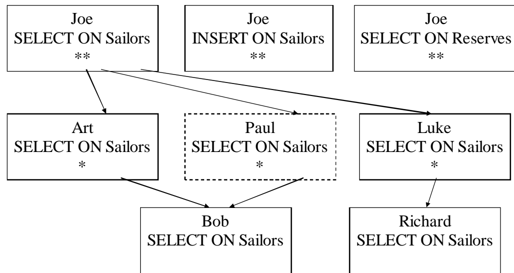
Revoking Privileges

- A granted privilege can be revoked at any time.
- Format of a *revoke statement*:

  REVOKE <privilege list> ON <database element>
  FROM <user list> (CASCADE | RESTRICT)
- CASCADE specifies that privileges that were *only* based on the revoked privilege are also revoked.
- RESTRICT means that the revoke statement is not executed if it would lead to a cascading revoke of other privileges.

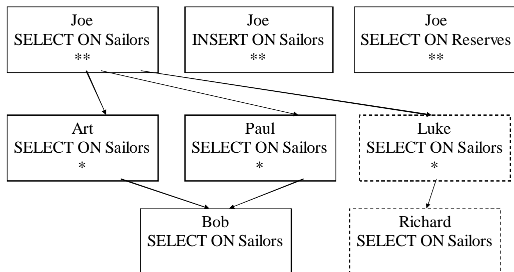
Revoking Privileges

REVOKE SELECT ON Sailors FROM Paul CASCADE



Revoking Privileges

REVOKE SELECT ON Sailors FROM Luke CASCADE



SFU SIMON FRASER UNIVERSITY

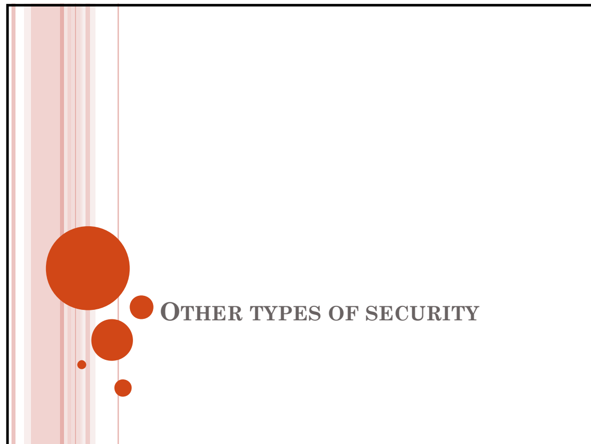
Role-Based Authorization

- In SQL-92, privileges are actually assigned to authorization ids, which can denote a single user or a group of users.
- In SQL:1999 (and in many current systems), privileges are assigned to roles.
  - Roles can then be granted to users and to other roles.
  - Reflects how real organizations work.
  - Illustrates how standards often catch up with "de facto" standards embodied in popular systems.

SFU SIMON FRASER UNIVERSITY

Security to the Level of a Field!

- Can create a view that only returns one field of one tuple. (How?)
- Then grant access to that view accordingly.
- Allows for *arbitrary* granularity of control, *but*:
  - Clumsy to specify, though this can be hidden under a good UI
  - Performance is unacceptable if we need to define field-granularity access frequently. (Too many view creations and look-ups.)



Other types of security

INTERNET-ORIENTED SECURITY

- Key Issues: User authentication and trust.
  - When DB must be accessed from a secure location, password-based schemes are usually adequate.
- *Encryption* is a technique used to address these issues.
- Masks" data for secure transmission or storage
  - *Encrypt(data, encryption key)* = encrypted data
  - *Decrypt(encrypted data, decryption key)* = original data
  - Without decryption key, the encrypted data is meaningless gibberish
- Can encrypt
  - Attribute
  - Table
  - Database

MANDATORY ACCESS CONTROL

- Based on system-wide policies that cannot be changed by individual users.
  - Each DB object is assigned a security class.
  - Each subject (user or user program) is assigned a clearance for a security class.
  - Rules based on security classes and clearances govern who can read/write which objects.
- Most commercial systems do not support mandatory access control. Versions of some DBMSs do support it; used for specialized (e.g., military) applications.

BELL-LAPADULA MODEL

- Objects (e.g., tables, views, tuples)
- Subjects (e.g., users, user programs)
- Security classes:
  - Top secret (TS), secret (S), confidential (C), unclassified (U): TS > S> C > U
- Each object and subject is assigned a class.
  - Subject S can read object O only if class(S) >= class(O) (Simple Security Property)
  - Subject S can write object O only if class(S) <= class(O) (*-Property)
- Idea is to ensure that information can never flow from a higher to a lower security level.

Multilevel Relations

bid	bname	color	class
101	Salsa	Red	S
102	Pinto	Brown	C

- Users with S and TS clearance will see both rows; a user with C will only see the 2nd row; a user with U will see no rows.
- If user with C tries to insert <101,Pasta,Blue,C>:
  - Allowing insertion violates key constraint
  - Disallowing insertion tells user that there is another object with key 101 that has a class > C!
  - Problem resolved by treating class field as part of key.

Statistical DB Security

- Statistical DB: Contains information about individuals, but allows only aggregate queries (e.g., average age, rather than Joe's age).
- New problem: It may be possible to infer some secret information!
  - E.g., If I know Joe is the oldest sailor, I can ask "How many sailors are older than X?" for different values of X until I get the answer 1; this allows me to infer Joe's age.
- Idea: Insist that each query must involve at least N rows, for some N. Will this work? (No!)

Why Minimum N is Not Enough

- By asking "How many sailors older than X?" until the system rejects the query, can identify a set of N sailors, including Joe, that are older than X; let X=55 at this point.
- Next, ask "What is the sum of ages of sailors older than X?" Let result be S1.
- Next, ask "What is sum of ages of sailors other than Joe who are older than X, plus my age?" Let result be S2.
- S1-S2 is Joe's age!

Case Study - AOL

- AOL released the log of 3 month's worth of searches by 650,000 users.
- *Problem?*

http://www.zoliblog.com/2006/08/06/aol-just-did-the-unthinkable---boycott-aol/

Case Study - NetFlix Challenge

- In order to get a better movie recommendation algorithm, the online DVD rental company gave more than 50,000 Netflix Prize contestants two massive datasets. The first included 100 million movie ratings, along with the date of the rating, a unique ID number for the subscriber, and the movie info. Based on this data from 480,000 customers, contestants had to come up with a recommendation algorithm that could predict 10 percent better than Netflix how those same subscribers rated other movies.
- *Problem?*

http://www.wired.com/threatlevel/2009/12/netflix-privacy-lawsuit/