Databases

Week 1 – Introduction

Dr. Roger Young

Course Overview

- Relational databases – Structured Query Language (SQL)
- ERD (Entity Relationship Diagram)
- Joins
- Java Database Connectivity (JDBC)
- Stored Procedures, Views, Triggers
- Normalization
- ACID v BASE
- NoSQL

What is a Database?

- We know what data is. A piece of information!
- Information is not useful if not organized

- Database – a collection of data stored in an organized manner

Why not store info in a file system?

- Still widely used today (e.g. for backup) but have the following problems

- **Data Redundancy** (Duplication of data)
  - Wasteful
  - Inconsistent
  - Loss of metadata integrity
    - Same data has different names in different files, or same name may have data in different files.

Why not store info in a file system

- **Lengthy Development Times**
  - Little opportunity to re-use previous development efforts
- **Limited Data Sharing**
  - Users have little opportunity to share data outside their own
- **Excessive Program Maintenance**
  - Factors above combine to create heavy maintenance load

Advantages of a Database

- Minimal Data Redundancy (duplication)
- Improved Consistency
- Improved Data Sharing
- Increased Application Development Productivity
- Enforcement of Standards
- Better Data Accessibility/ Responsiveness
- Security, Backup/Recovery, Concurrency

Advantages of a DBMS

- So why not use them always?
  - Expensive/complicated to set up & maintain
  - This cost & complexity must be offset by need
  - General-purpose, not suited for special-purpose tasks (e.g. text search!)

Database management system

- Need for DBMS has exploded in the last years
  - **Corporate:** retail swipe/clickstreams, “customer relationship mgmt”, “supply chain mgmt”, “data warehouses”, etc.
  - **Scientific:** digital libraries, Human Genome project, NASA Missions, physical sensors, grid physics network

Databases and roles

- Database administrators (DBAs) role
  - Design logical/physical schemas
  - Handle security and authorization
  - Data availability, crash recovery
  - Database tuning as needs evolve

Database management system

- A **Database Management System (DBMS)** is a software system designed to store, manage, and facilitate access to databases.
- It is a data storage and retrieval system which permits data to be stored non-redundantly while making it appear to the user as if the data is well-integrated.

Relational Database

- A relational database is a collection of data items organized as a set of formally described tables from which data can be accessed easily.
- A relational database is created using the relational model.
- The software used in a relational database is called a relational database management system (RDBMS).
- The relational database was first defined in June 1970 by of IBM

Relational database management system

- A data model is a collection of concepts for describing data.
- RDBMS is based on the relational model of data
  - Each database has a set of named relations (tables)
  - Each relation has a set of named attributes (columns)
  - Each tuple (row) has a value for each attribute
  - Each attribute has a defined type

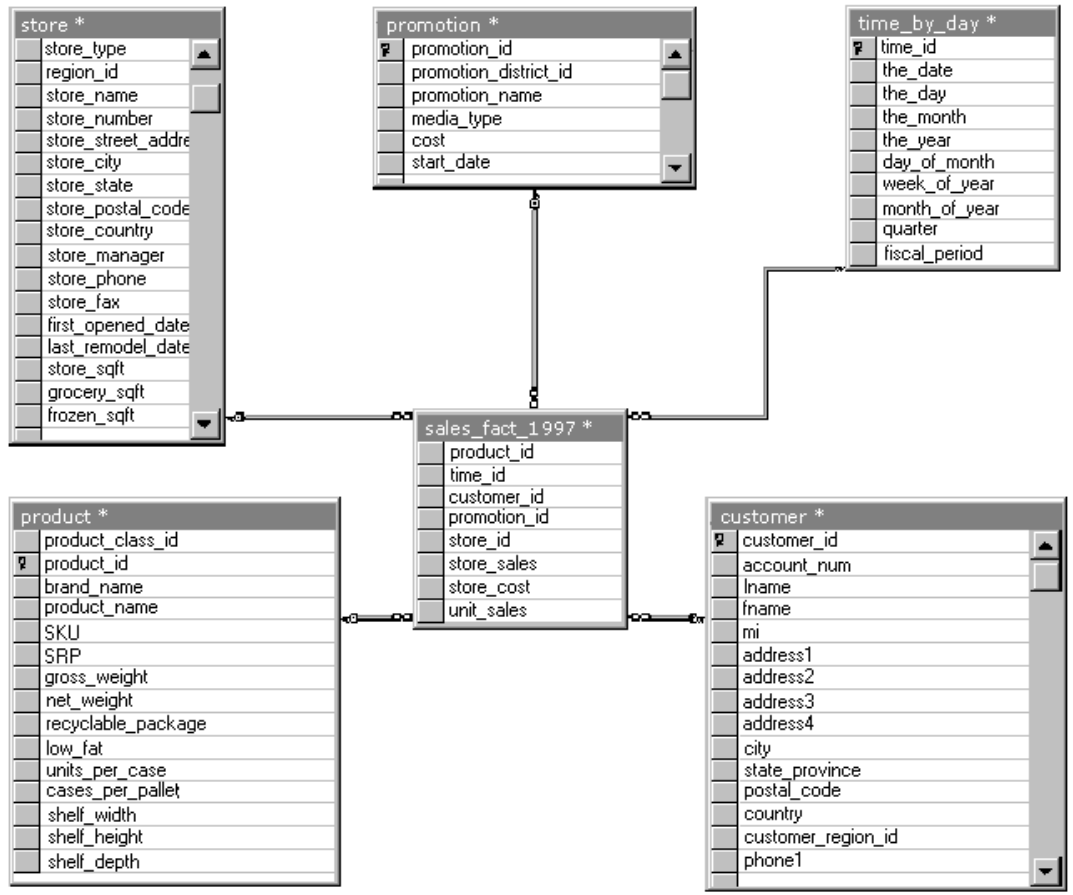
Relational database management system

- A ***schema*** is a description of a particular collection of data, using a given data model. It describes the structure of the database.

- **Schema of RDBMS database**
  - A structural description of the relations (tables) in the database

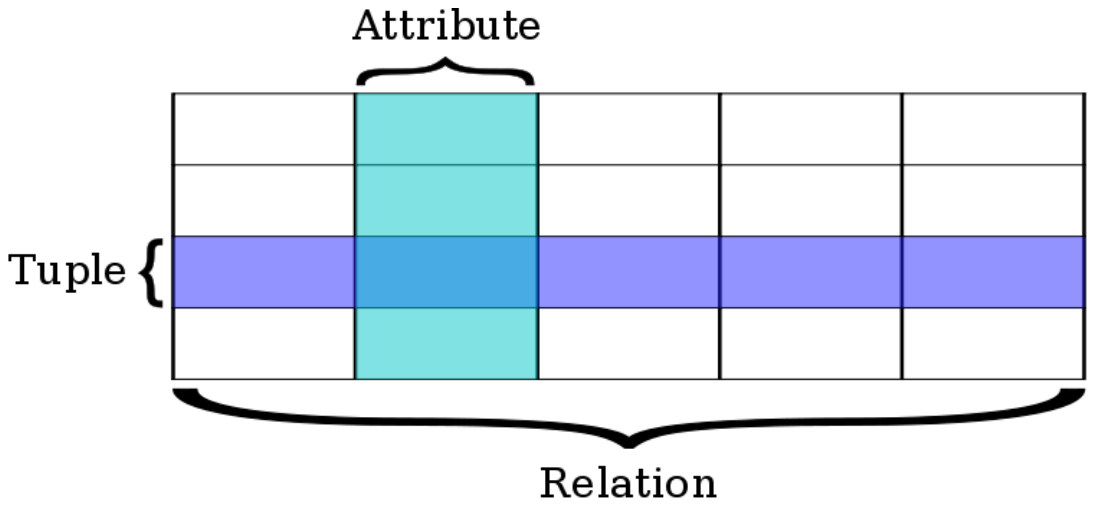
Relational database management system

- **Schema Example**



Relational Databases

- A relation is defined as a set of rows that have the same attributes.
- A row usually represents an object and information about that object.
- Objects are typically physical objects or concepts.
- A relation is usually described as a table, which is organized into rows and columns.



Definitions of Terminology

Formal relational term	Informal equivalents
relation	table
tuple	row or record
cardinality	number of rows
attribute	column or field
degree	number of columns
(unique) identifier	Primary key

Overview of Relational Databases

- Entity
  - Object about which you want to store data
  - Different tables store data about each different entity
- Relationships
  - Links that show how different records are related

Overview of Relational Databases

- Key fields
  - Establish relationships among records in different tables
  - Main types of key fields
    - Primary
    - Candidate
    - Foreign
    - Composite

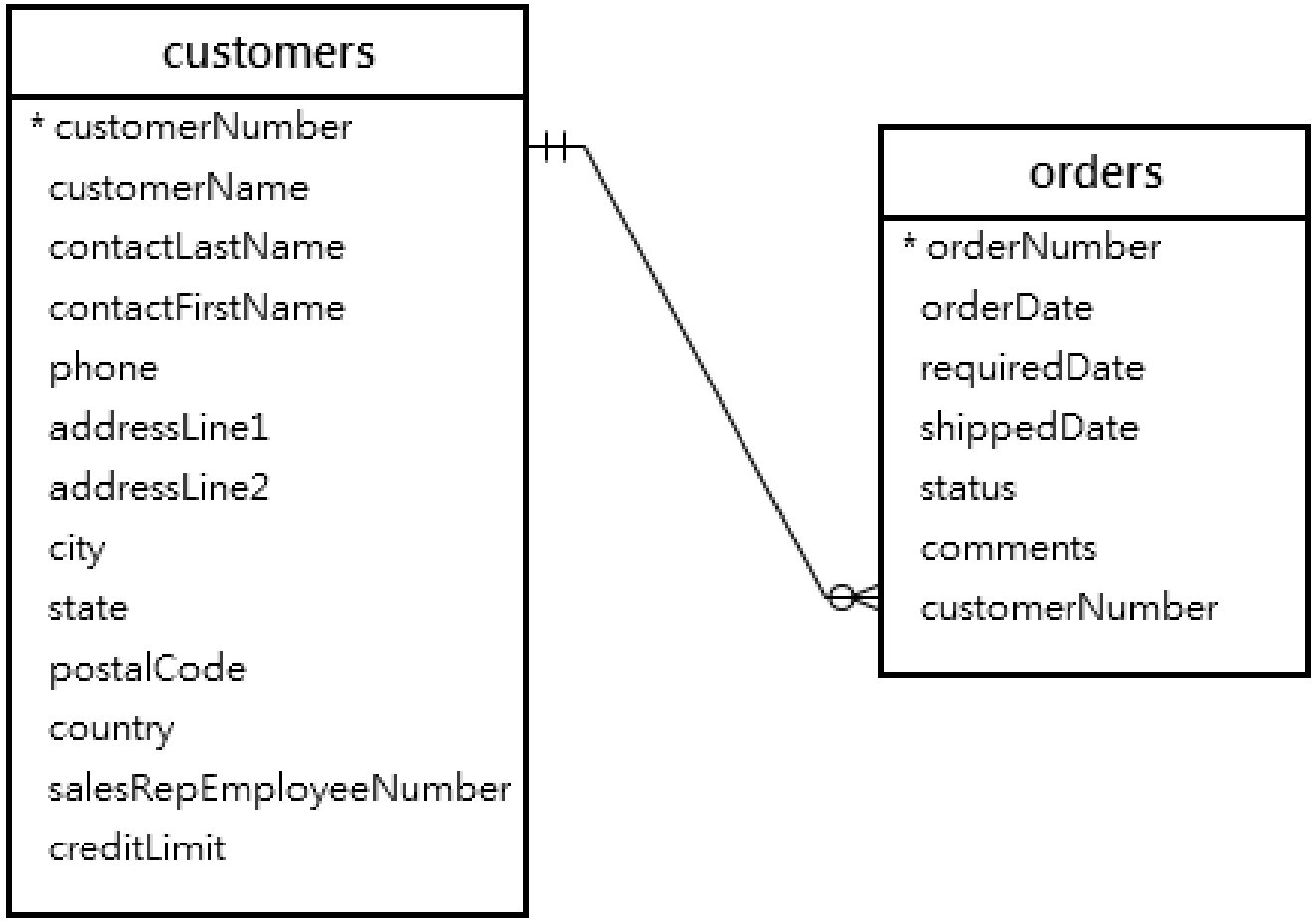
Primary Keys

- Column in relational database table whose value must be unique for each row
- Serves to identify individual occurrence of entity
- Every row must have a primary key
- Cannot be NULL
- NULL
  - Value is absent or unknown
  - No entry is made for that data element

Foreign Keys

- Column in table that is a primary key in another table
- Creates relationship between two tables
- Value must exist in table where it is the primary key

Primary / Foreign Key Examples



SQL Data Types

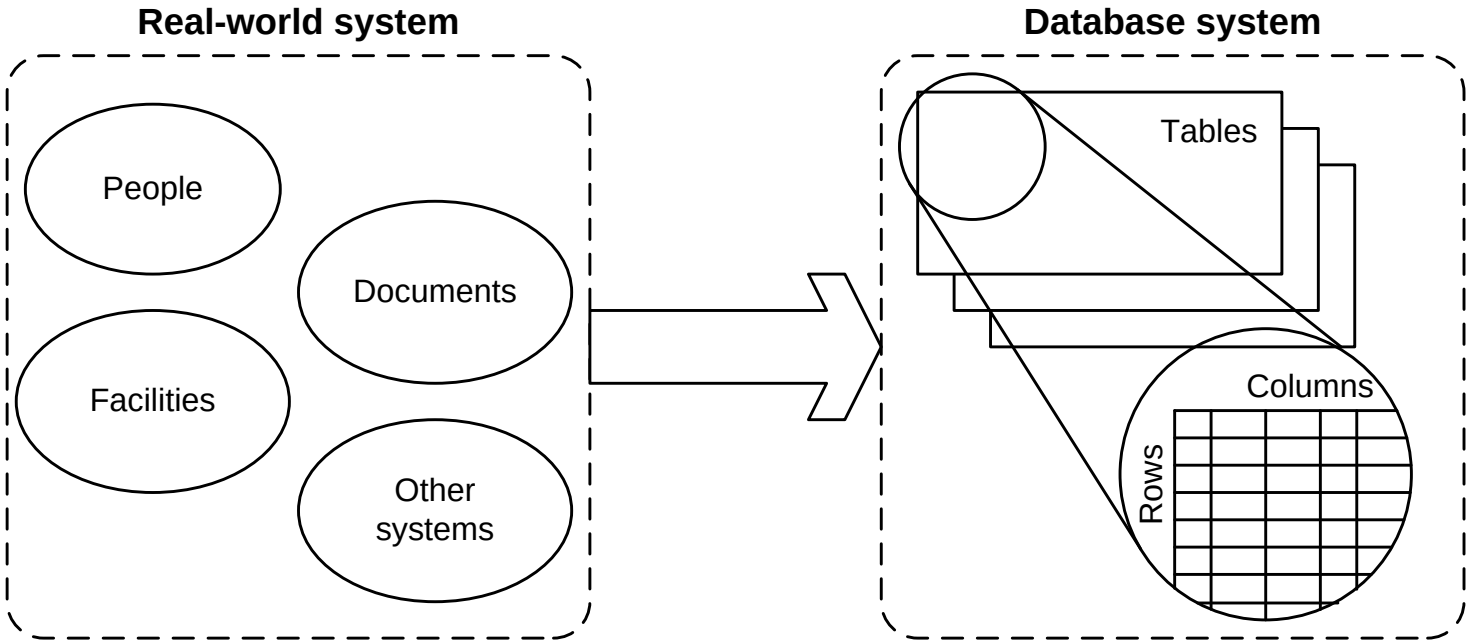
- Each Database has its own range of data types.

- In MySQL there are three main types
  - Text
  - Number
  - Date/Time

Some MySQL Data Types

Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. Note: If you put a greater value than 255 it will be converted to a TEXT type
TEXT	Holds a string with a maximum length of 65,535 characters
INT	-2147483648 to 2147483647 normal. The maximum number of digits may be specified in parenthesis
DATE()	A date. Format: YYYY-MM-DD
DATETIME()	A date and time combination. Format: YYYY-MM-DD HH:MM:SS

A Database is modelled on a real-world system



Reference: MYSQL, Murach publishing

Database: designing the data structure

Database design	SQL statements for data definition
Identify the tables, table attributes (columns) and how tables relate to each other	CREATE DATABASE (LIBRARY)
Identify the primary key and foreign keys, the link between the tables	CREATE TABLE For: BOOK, BORROWER, LIBRARIAN, AUTHOR, lending
Determine the data type of the attribute	
	ALTER DATABASE
	ALTER TABLE
	DROP TABLE

SQL: data definition

SHOW DATABASES;	Lists all defined databases
SHOW DATABASES LIKE db_name;	List all defined databases that match criteria
CREATE DATABASE[if not exists] db_name;	Creates database;
USE db_name;	All further statements are towards database
CREATE TABLE table_name(data_field 1 datatype, data_field2 datatype, PRIMARY KEY (dataname));	Creates table in current database
SHOW TABLES in db-name;	Lists the tables

SQL – (Structured Query Language)

- Data Definition Language (DDL)
  - Create/alter/delete tables and their attributes

- Data Manipulation Language (DML)
  - Query one or more tables
  - Insert/delete/modify tuples in tables

Data Definition Language (DDL)

• Data Definition Language (DDL) is a vocabulary used to define data structures (database schema) in SQL. These statements are used to create, alter, or drop data structures in SQL.

1. *CREATE*
2. *DROP*
3. *ALTER*

CREATE TABLE

- Specifies a new table by giving it a name, and specifying each of its attributes and their data types (INTEGER, FLOAT, DECIMAL(i,j), CHAR(n), VARCHAR(n))
- A constraint NOT NULL may be specified on an attribute

```
CREATE TABLE   DEPARTMENT
        (          DNAME  VARCHAR(10)      NOT NULL,
                   DNUMBER          INTEGER NOT NULL,
                   MANAGER          CHAR(9),
                   MGRSTARTDATE  CHAR(9)  );
```

SQL DDL Creation Syntax

CREATE TABLE *table-name*
(*attribute-name domain,*
attribute-name domain);

SQL DDL Creation Example

CREATE TABLE *branch*
(name varchar*(10),*
city varchar*(20),*
director varchar*(20),*
assets integer);

branch

name	city	director	assets

DROP TABLE

- Used to remove a relation (table) *and its definition*
- The relation can no longer be used in queries, updates, or any other commands since its description no longer exists
- <u>Example:</u>

DROP TABLE BRANCH;

SQL DDL Deletion Syntax

DROP TABLE *table_name*

Examples:

DROP TABLE *branch;*
DROP TABLE *FoodCart*;

ALTER TABLE

- Used to add an attribute to one of the tables
- The new attribute will have NULLs in all the tuples of the relation right after the command is executed; hence, the NOT NULL constraint is *not allowed* for such an attribute
- Example:

ALTER TABLE EMPLOYEE **ADD** JOB VARCHAR(12);

- The database users must still enter a value for the new attribute JOB for each EMPLOYEE tuple. This can be done using the UPDATE command.

SQL DDL Alteration Syntax

To add an attribute:

ALTER TABLE *table_name*
ADD *Att Domain;*

To remove an attribute:

ALTER TABLE *table_name*
DROP *Att;*

SQL DDL Alteration Example

ALTER TABLE *branch* ADD *zip* INTEGER;

branch

name	city	director	assets

becomes

branch

name	city	director	assets	zip

SQL DDL Alteration Example

ALTER TABLE *branch* DROP *zip;*

branch	name	city	director	assets	zip

becomes

branch	name	city	director	assets

Create your own Database & Table

- Open Workbench and create a students table with 5 columns of your choice

- To create a new database

- DROP DATABASE IF EXISTS lab1; #this will delete an existing db called test
- CREATE DATABASE IF NOT EXISTS lab1; # this will create a new db
- USE lab1; #you could have many dbs, this tells the system which one you want to work on

Data Manipulation Language (DML)

• Data manipulation language comprises the SQL data change statements, which modify stored data but not the schema or database objects.

INSERT INTO ... VALUES ...

SELECT….FROM….WHERE…

UPDATE ... SET ... WHERE ...

DELETE FROM ... WHERE ...

INSERT INTO (DML)

- Adds data to a table
- Syntax:
  INSERT INTO *table_name* (*column*, …, *column*)
    VALUES (*value*, …, *value*);

- The *column*s are the names of columns you are putting data into, and the *value*s are that data
- String data must be enclosed in single quotes
- Numbers are not quoted
- You can omit the column names if you supply a value for *every* column

INSERT (cont.)

- In its simplest form, it is used to add one or more tuples to a relation
- Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command

INSERT INTO (Cont.)

- Inserting into a table
  - **Insert into employee (emp_Name, Dept_no, gender, salary)**
    Values (‘Sara johns’, 1, ‘F’, 1440);
- Inserting a record that has some null attributes requires identifying the fields that actually get data
- When you insert a record and you have values for all attributes, there is no need to specify the attributes names.
  - **Insert into employee**
    Values (‘Suzy Alan’, 10, ‘F’, 1200);
- Inserting from another table
  - **INSERT INTO emp_senior**
    select * from employee where age > 60;

The main condition in this case, that both tables has the same attributes and ordered in the same order

Insert Example

- To insert a row into a table, it is necessary to have a value for each attribute, and order matters.

Example: INSERT into FoodCart
VALUES ('02/26/08', 'pizza', 70);

FoodCart

date	food	sold
02/25/08	pizza	350
02/26/08	hotdog	500

date	food	sold
02/25/08	pizza	350
02/26/08	hotdog	500
02/26/08	pizza	70

Revisiting Relational Terminology

Table name

Attribute names

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Tuples or rows

Revisiting Relational Terminology

- The *schema* of a table is the table name and its attributes:

Product(PName, Price, Category, Manfacturer)

- A *key* is an attribute whose values are unique;
  we underline a key

Product(PName, Price, Category, Manfacturer)

SQL Query

Basic form:

```
SELECT  <attributes>
FROM    <one or more relations>
WHERE   <conditions>
```

Insert Values into your student table

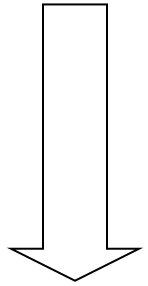
- Insert information on 5 students into your previously created student table

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  *
FROM    Product
WHERE   category='Gadgets'
```



"selection"

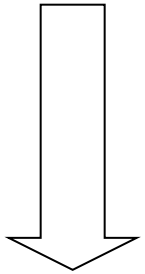
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT   PName, Price, Manufacturer
FROM     Product
WHERE    Price > 100
```



"selection"

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

Notation

Input Schema

Product(PName, Price, Category, Manfacturer)

```
SELECT   PName, Price, Manufacturer
FROM     Product
WHERE    Price > 100
```

Answer(PName, Price, Manfacturer)

Output Schema

Deletion Syntax

To delete rows from the table:

DELETE FROM <table name>
WHERE <condition>;

Deletion Example

```
DELETE FROM FoodCart
    WHERE food = 'hotdog';
```

FoodCart

date	food	sold
02/25/08	pizza	349
02/26/08	hotdog	500
02/26/08	pizza	70

Becomes

date	food	sold
02/25/08	pizza	349
02/26/08	pizza	70

Note: If the WHERE clause is omitted all rows of data are deleted from the table.

Another Delete Example

```
DELETE FROM Student
WHERE sNumber=6;
```

Update Syntax

To update the content of the table:

UPDATE <table name>

SET <attr> = <value>

WHERE <selection condition>;

Update Example

UPDATE FoodCart SET sold = 349

WHERE date = ’02/25/08’ AND food = ‘pizza’;

FoodCart

date	food	sold
02/25/08	pizza	350
02/26/08	hotdog	500
02/26/08	pizza	70

Becomes

date	food	sold
02/25/08	pizza	349
02/26/08	hotdog	500
02/26/08	pizza	70

Constraints

- Constraints are used to enforce the integrity of the data in a table by defining rules about values that can be stored in the columns of the table.
- Types of constraints include:
  1. NOT NULL – prevents null values from being stored in a column
  2. PRIMARY KEY
  3. FOREIGN KEY CONSTRAINT

SQL NULL Values

- NULL values represent missing unknown data.
- By default, a table column can hold NULL values.
- If a column in a table is optional, we can insert a new record or update an existing record without adding a value to this column. This means that the field will be saved with a NULL value.

SQL NULL Values

"Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola		Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari		Stavanger

- Suppose that the "Address" column in the "Persons" table is optional. This means that if we insert a record with no value for the "Address" column, the "Address" column will be saved with a NULL value.
- How can we test for NULL values?
- It is not possible to test for NULL values with comparison operators,
- such as =, <, or <>.
- We will have to use the **IS NULL** and **IS NOT NULL** operators instead.

SQL IS NULL

- How do we select only the records with NULL values in the "Address" column?
- We will have to use the IS NULL operator:

```
SELECT LastName,FirstName,Address FROM Persons
WHERE Address IS NULL;
```

The result-set will look like this:

LastName	FirstName	Address
Hansen	Ola	
Pettersen	Kari	

PRIMARY KEY

- There are 2 ways to define a PK
  1. Column-level constraint

  Put the PRIMARY KEY keywords after the data type for the column

  2. Table-level constraint

  You can also define a constraint at the table level you can provide a name for the constraint

  See examples on next slide

Column-level Constraint

```
CREATE TABLE students
{
    student_id INT        PRIMARY KEY,
    student-name        VARCHAR(30)  NOT NULL
 };
```

Table-level constraint

```
CREATE TABLE students
{
    student_id              INT,
    student-name            VARCHAR(30)      NOT NULL,
    CONSTRAINT student_pk PRIMARK KEY (student_id)
 }
```

You can code PK constraint either way, both have the same effect

FOREIGN KEY CONSTRAINT

- A *foreign key constraint* (reference constraint) requires values in one table to match values in another table. This defines the relationship between two tables and enforces referential integrity.

A table with a column-level foreign key constraint

- To create a fk constraint at the column level, you code the REFERENCES keyword followed by the name of the related table and the name of the related column in parentheses.

```
CREATE TABLE invoices
(
  invoice_id      INT    PRIMARY KEY,
  vendor_id       INT    REFERENCES vendors (vendor_id),
  invoice_number  VARCHAR(50)    NOT NULL    UNIQUE
);
```

A table with a table-level foreign key constraint

```
CREATE TABLE invoices
(
  invoice_id       INT          PRIMARY KEY,
  vendor_id        INT          NOT NULL,
  invoice_number   VARCHAR(50)  NOT NULL    UNIQUE,
  CONSTRAINT invoices_fk_vendors
    FOREIGN KEY (vendor_id)
       REFERENCES vendors (vendor_id)
) ;
```

Note if you try to Insert a row into invoices with a vendor_id value that isn't matched by the vendor_id column in the vendors table, you will get an error message

Referential Integrity

- The system will display an error message if a constraint was violated
- If you try to delete rows that have related rows in another table , you will also see an error message
- In some cases you may want to automatically delete the related rows - use ON DELETE CASCADE

Referential Integrity ctd

- Then when you delete a row from the pk table, the delete is cascaded to the related rows in the foreign table

```
CONSTRAINT invoices_fk_vendors
  FOREIGN KEY (vendor_id) REFERENCES vendors (vendor_id)
  ON DELETE CASCADE
```

- Use cascading deletes with caution …

Keys and Foreign Keys

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Key

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Foreign key

Altering the constraints of a table

- You may need to change the constraints of a table after you create it
- Use the ALTER TABLE statement

```
ALTER TABLE vendors
ADD PRIMARY KEY (vendor_id)

ALTER TABLE vendors
DROP PRIMARY KEY
```

SQL Queries:
The LIKE operator

```
SELECT   *
FROM     Products
WHERE    PName LIKE '%gizmo%'
```

- **LIKE** : pattern matching on strings
- may contain two special symbols:
  - % = any sequence of characters
  - _ = any single character

LIKE Examples

- Select firstnames where second letter = U
- Select * from table where firstName like ‘_u%’;

- Select lastnames that end in S
- Select * from table where lastName like ‘%s’;

Eliminating Duplicates

```
SELECT   DISTINCT category
FROM     Product
```

Category
Gadgets
Photography
Household

Compared to:

```
SELECT   category
FROM     Product
```

Category
Gadgets
Gadgets
Photography
Household

SELECT **DISTINCT** Example

"Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

```
SELECT DISTINCT City
    FROM Persons
```

City
Sandnes
Stavanger

Another **DISTINCT** Example

name	city	director	assets
Branch_one	Jakarta	Bo Lee	80000
Clementi	Singapore	Ng Wee Hiong	3000000
F_branch	Johor Barhu	John	1500000
KL_branch	Kuala Lumpur	Yu Fei	1000000
Monas	Jakarta	Agus Arianto	4000000
S_branch	Johor Barhu	George	1200000

SELECT COUNT(DISTINCT city) as NumCities
FROM branch

NumCities
4

SQL **ORDER BY** Keyword

The ORDER BY keyword is used to sort the result-set by a specified column.

The ORDER BY keyword sort the records in ascending order by default.

If you want to sort the records in a descending order, you can use the DESC keyword.

SQL ORDER BY Syntax:

```
SELECT column_name(s)
        FROM table_name
        ORDER BY column_name(s) ASC|DESC
```

ORDER BY (Default ASC) Example

"Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

SELECT * FROM Persons
ORDER BY LastName

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
4	Nilsen	Tom	Vingvn 23	Stavanger
3	Pettersen	Kari	Storgt 20	Stavanger
2	Svendson	Tove	Borgvn 23	Sandnes

ORDER BY (DESC) Example

"Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

```
SELECT * FROM Persons
        ORDER BY LastName DESC
```

_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger
1	Hansen	Ola	Timoteivn 10	Sandnes

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT   DISTINCT category
FROM     Product
ORDER BY category
```

Category
Gadgets
Household
Photography

```
SELECT   Category
FROM     Product
ORDER BY PName
```

Category
Gadgets
Household
Gadgets
Photography

```
SELECT   DISTINCT category
FROM     Product
ORDER BY PName
```

Category
Gadgets
Household
Photography

SQL Alias

With SQL, an alias name can be given to a table.

SQL Alias Syntax:

```
SELECT column_name(s)
        FROM table_name
        AS alias_name;
```

SQL Scalar functions

- SQL scalar functions return a single value, based on the input value.
- Useful scalar functions:
  - UCASE() - Converts a field to upper case
  - LCASE() - Converts a field to lower case
  - LENGTH() - Returns the length of a text field
  - ROUND() - Rounds a numeric field to the number of decimals specified
  - NOW() - Returns the current system date and time

SQL Aggregate Functions

- SQL has many built-in functions for performing calculations on data.

- SQL Aggregate Functions return a single value, calculated from values in a column. Useful aggregate functions:
  - AVG() - Returns the average value
  - COUNT() - Returns the number of rows
  - SUM() - Returns the sum

Aggregation

```
SELECT  avg(price)
FROM    Product
WHERE   maker="Toyota"
```

```
SELECT  count(*)
FROM    Product
WHERE   year > 1995
```

SQL supports several aggregation operations:

sum, count, min, max, avg

Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

```
SELECT  Count(category)
FROM    Product
WHERE   year > 1995
```

same as Count(*)

We probably want:

```
SELECT  Count(DISTINCT category)
FROM    Product
WHERE   year > 1995
```

Simple Aggregations

Purchase

Product	Date	Price	Quantity
Bagel	10/21	1	20
Banana	10/3	0.5	10
Banana	10/10	1	10
Bagel	10/25	1.50	20

```
SELECT  Sum(price * quantity)
FROM      Purchase
WHERE   product = 'bagel'
```

50 (= 20+30)

SQL COUNT(column_name) Example

"Orders" table:

_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

```
SELECT COUNT(Customer) AS CustomerNilsen
FROM Orders
WHERE Customer='Nilsen'
```

CustomerNilsen
2

SQL COUNT(*) Example

"Orders" table:

_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

```
SELECT COUNT(*) AS NumberOfOrders
FROM Orders
```

NumberOfOrders
6

SQL COUNT(DISTINCT column_name) Example

"Orders" table:

_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

```
SELECT COUNT(DISTINCT Customer)
AS NumberOfCustomers
FROM Orders
```

NumberOfCustomers
3

SQL AVG() Function

The AVG() function returns the average value of a numeric column.

SQL AVG() Syntax:

```
SELECT AVG(column_name)
FROM table_name
```

SQL AVG() Example

"Orders" table:

_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

SELECT AVG(OrderPrice) AS OrderAverage
FROM Orders

OrderAverage
950

SQL Cont'd

- Select ***(Group By, Having Clause)***
- Now(), AND, OR
- Nested Subqueries
- More DML
  - Insert
  - Delete
  - Update
- Constraints

How to group and summarize data

- The GROUP BY clause groups the rows of a result set based on one or more columns or expressions.
- Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;
```

Example using details table

id	firstName	lastName	age	gender	position	department	rate	hours
1	Joe	Mullins	64	M	Lecturer	Engineering	63.08	12
2	Joan	Macgill	27	F	Researcher	Science	38.00	35
3	Jim	Mitchell	51	M	Researcher	Business	38.00	25
4	John	Magner	47	M	Lecturer	Humanities	63.08	16
5	Jean	Madden	45	F	Professor	Design	76.45	14
6	Jack	Minogue	61	M	Administrator	Hospitality	45.57	37
7	Josephine	Mahony	33	F	Head	Nursing	98.56	40
8	Juan	Mosley	56	M	Professor	Engineering	76.45	11
9	Jamie	Mulllen	45	M	Researcher	Science	38.00	37

```
select department, count(*) AS "Number of Emp "
from detailslab2.details
group by department;
```

department	Number of Emp
Business	4
Design	2
Engineering	4
Hospitality	3
Humanities	1
Nursing	2
Science	4

Group By with 2 columns

- If you include 2 or more columns or expressions in the GROUP BY clause, they will form a hierarchy where each column or expression is subordinate to the previous one.

```
select department,gender, count(*) AS "Number of Emp "
from detailslab2.details
group by department, gender;
```

department	gender	Number of Emp
Business	F	2
Business	M	2
Design	F	2
Engineering	F	1
Engineering	M	3
Hospitality	M	3
Humanities	M	1
Nursing	F	2
Science	F	2
Science	M	2

Group By and Having Clause

- The GROUP BY clause determines how the selected rows are grouped, and the HAVING clause determines which groups are included in the final results

Group By & Having Clause example

```
select department,gender, count(*) AS "Number of Emp "
from detailslab2.details
group by department, gender;
```

department	gender	Number of Emp
Business	F	2
Business	M	2
Design	F	2
Engineering	F	1
Engineering	M	3
Hospitality	M	3
Humanities	M	1
Nursing	F	2
Science	F	2
Science	M	2

```
select department,gender, count(*) AS "Number of Emp"
from detailslab2.details
group by department, gender
having count(*)  >2;
```

department	gender	Number of Emp
Engineering	M	3
Hospitality	M	3

Group By & WITH ROLLUP

- You can use the WITH ROLLUP operator in the GROUP BY clause to add summary rows to the final result set.

WITH ROLLUP example

```
select department, gender, count(*) AS "Number of Emp"
from details
group by department, gender  WITH ROLLUP;
```

department	gender	Number of Emp
Business	F	2
Business	M	2
Business	NULL	4
Design	F	2
Design	NULL	2
Engineering	F	1
Engineering	M	3
Engineering	NULL	4
Hospitality	M	3
Hospitality	NULL	3
Humanities	M	1
Humanities	NULL	1
Nursing	F	2
Nursing	NULL	2
Science	F	2
Science	M	2
Science	NULL	4
NULL	NULL	20

SQL NOW() Function

The NOW() function returns the current system date and time.

SQL NOW() Syntax:

```
SELECT NOW()
        FROM table_name
```

SQL NOW() Example

"Products" table:

Prod_Id	ProductName	Unit	UnitPrice
1	Jarlsberg	1000 g	10.45
2	Mascarpone	1000 g	32.56
3	Gorgonzola	1000 g	15.67

SELECT ProductName, UnitPrice, Now() as PerDate FROM Products

ProductName	UnitPrice	PerDate
Jarlsberg	10.45	10/7/2008 11:25:02 AM
Mascarpone	32.56	10/7/2008 11:25:02 AM
Gorgonzola	15.67	10/7/2008 11:25:02 AM

SQL AND Operator

The AND operator is used to filter records based on more than one condition.

The AND operator displays a record if both the first condition and the second condition is true.

AND Example

"Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

```
SELECT * FROM Persons
WHERE FirstName='Tove'
AND LastName='Svendson'
```

_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes

SQL OR Operator

The OR operator is used to filter records based on more than one condition.

The OR operator displays a record if either the first condition or the second condition is true

OR Example

"Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

```
SELECT * FROM Persons
WHERE FirstName='Tove'
OR FirstName='Ola'
```

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Nested Queries

- There may be scenarios where you need the output of one query to be used in another query. This is possible through nested queries.

- For example. In the details lab, you may want to view staff who are younger than the average age.
- Instead of using two separate queries as follows:
- Select avg(age) from details; “which returns 41”
- Followed by
- Select * from details where age < 41

you can do the following:

- Select * from details where age <(select avg(age) from details);