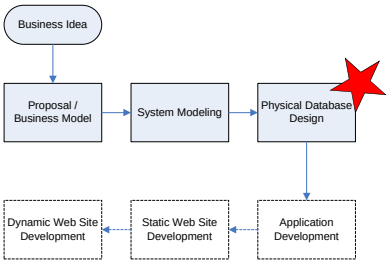
Assignment #5
(1/3) – Due 11/30/2011

- We're almost ready to build our application. Before that, however, we need to have our database ready.



1

Assignment #5
(2/3)

- Based on your data model (assignment #4), you're required to build a physical database, normalize it (to the 3rd normal form at least), and populate it with data.
  - The data will be provided in the database "data" (after the coming Sunday) that can be imported into your SQL server.
  - You'll find the database given to be quite different from your physical data model, contains error, and not normalized. This is ***normal.***
  - You will have to populate your database with the given data using SQL queries. You can either use pure SQL or with the assistance of the "query builder" to build these queries in order to populate your physical database.
  - The process is known as ETL (Extract/Transform/Load).

2

Assignment #5
(3/3)

- Submit through the blackboard system
  1. An E-R diagram showing your normalized physical database for your business.
  2. A word document contains a step-by-step SQL statements that you used to ETL the given data into your database.
     - Should also state the purpose of the statement
     - Note ETL also need to clean up the errors in data, so you should have some statements to find wrong data and get rid of them.
       - There are several errors introduced (on purpose) in the data that you will receive.

3

Mid Term

- Lecture 1 to the end of next week's lecture on ETL.
- Open book, open note, but no computer!
- Paper-based
- Date?
  - 11/25 (Friday) night?
  - 12/2 (Friday) night?
  - Other?

4

Lecture 9

SQL (II)

5

Review

- SQL – Structured Query Language
  - Database Creation & Connection
  - Table & Index Creation
  - Changing / Deleting Objects
  - Basic Data Manipulation
    - Data insertion
    - Data retrieval
    - Data deletion and updating

6

SQL syntax

- Free form, case insensitive

```
Create database dbname;
Create table newTable (
   f1 int primary key,
   f2 int
);
alter table newTable add f3 int;
alter table newTable drop column f2;
drop table newTable;
drop database dbname;
```

7

Reviews

```
insert into newTable values (1, 3);
insert into newTable values (2, 4);
insert into newTable(f1) values (3);
insert into newTable (f1, f3) values (4, 3);
insert into newTable (f3, f1) values (3, 4);
insert into newTable (f3) values (5);
```

8

Today's Topic

- SQL
  - DELETE & UPDATE
  - Advanced Select
  - GROUP BY and HAVING clauses of SELECT
  - Joins
  - Sub-queries
  - Views

9

Data Deletion and Updating

- Delete data
  - DELETE FROM `tableName` WHERE `[criteria]`
  - `DELETE FROM newTable WHERE f1 < 3;`
- Updating data
  - UPDATE `tableName` SET `fieldName=newValue [, fieldName1=newValue1, …]`;
    (NOTE! This will update ALL records in the table, use WHERE to limit records to be updated.)
  - UPDATE `tableName` SET […] WHERE `[criteria]`
  - UPDATE newTable SET f3 = 5;
  - UPDATE newTable SET f3 = f3 * 2 WHERE f1 < 3;

10

Examples

- Find the average sale
  - SELECT AVG(`Amt`) FROM `Orders`;
- Find the average sale for a customer
  - SELECT AVG(`Amt`) FROM `Orders` WHERE `Cust = 211`;
- Add an office
  - INSERT INTO `Offices (OfficeNbr, City, Region, Target, Sales)` VALUES `('55', 'Dallas', 'West', 200000, 0)`;
- Delete a customer
  - DELETE FROM `Customers` WHERE `Company = 'Connor Co'`;
- Raise a credit limit
  - UPDATE `CustomersSET CreditLimit = 75000` WHERE `Company = 'AmaratungaEnterprise'`;

11

Advanced SELECT

- SELECT is the real heart of SQL.[1]
- SELECT [ALL | DISTINCT] `selection_list`
  FROM `{tableName | viewName} [, …]`
  [WHERE `criteria`]
  [GROUP BY `columnName [, …]`]
  [HAVING `criteria`]
  [ORDER BY `{columnName | select_list_number}`]
  [DESC]
- SQL is a free-form language. However, the order of these clauses has to be in the order shown above to be syntax correct.

12

[1] The Practical SQL Handbook, Judith et al. Addison-Wesley, ISBN 0201447878

1. selection_list

- SELECT [ALL | DISTINCT] selection_list
  FROM {tableName | viewName} [, …]
  - selection_list contains column names to be picked. The order of column names can be arranged in any order as you pleased.
  - selection_list can also be used to specify new table display labels
  - selection_list can do computations: (),*/,+-, functions
  - selection_list can also be specified by using tableName.columnName;

13

2. FROM clause

- SELECT [ALL | DISTINCT] selection_list
  FROM {tableName | viewName} [, …]
  - tableName can be given an alias, which is important when we're dealing with joins (to be discussed)

14

3. WHERE clause

- SELECT [ALL | DISTINCT] selection_list
  FROM {tableName | viewName} [, …]
  [WHERE criteria]
  - Criteria can have
    - Comparison operators (=, <, >, <=, >=, <>)
    - Combinations or logical negations (AND, OR, NOT)
    - Lists (IN, NOT IN)
    - Ranges (BETWEEN and NOT BETWEEN)
    - Unknown values (IS NULL and IS NOT NULL)
    - Character matches (LIKE and NOT LIKE)
      - %: matching any length of characters
      - _: matching one character

15

4. ORDER BY clause

- SELECT [ALL | DISTINCT] selection_list
  FROM {tableName | viewName} [, …]
  [WHERE criteria]
  [GROUP BY columnName [, …]]
  [HAVING critera]
  [ORDER BY {columnName | select_list_number}]
  [DESC]
  - Column name or select_list_number
  - DESC suggested the sorted list should be in descending order
  - The list can have multiple column names

16

5. GROUP BY clause

- SELECT [ALL | DISTINCT] selection_list
  FROM {tableName | viewName} [, …]
  [GROUP BY columnName [, …]]
  [HAVING criteria]
  - Aggregate using the entire table
    - SUM(exp), AVG(exp), COUNT(exp), COUNT(*), MAX(exp), MIN(exp)
    - SUM(DISTINCT exp), AVG(DISTINCT exp), COUNT(DISTINCT exp)
  - Aggregate with GROUP BY
  - WHERE criteria that is a pre-filtering criteria to select records before aggregation; while HAVING criteria is post-filtering criteria to select aggregated records after aggregation.

17

Examples
Aggregation of the entire table

- Summation of sales amount
  - SELECT SUM(amt) AS total FROM orders;
- Count the number of orders
  - SELECT COUNT(cust) FROM orders;
- NULLs are not counted!
  - SELECT COUNT(qty), COUNT(*) FROM orders;
- Various aggregated functions applied
  - SELECT MAX(amt), MIN(amt), SUM(amt), AVG(amt) FROM orders WHERE prod LIKE '%CRANE%';
- Find distinct customers
  - SELECT COUNT(DISTINCT cust), COUNT(cust) FROM orders;

18

Example
Grouping

- Find the total amount of product purchased by each customer
  - SELECT cust, SUM(amt), COUNT(cust), SUM(qty) FROM orders GROUP BY cust;
- Find the number of distinct products purchased by each customer
  - SELECT cust, COUNT(DISTINCT prod) FROM orders GROUP BY cust;
- Find the distinct products purchased by each customer
  - SELECT cust, prod FROM orders GROUP BY cust, prod;
- Find distinct customers of each product purchased
  - SELECT prod, cust FROM orders GROUP BY prod, cust;

19

Example
Having

- Find total amount of purchase of each customer
  - SELECT cust, SUM(amt) AS total FROM orders GROUP BY cust;
- Find out who is our VIP
  - SELECT cust, SUM(amt) AS total FROM orders GROUP BY cust HAVING total > 2000000; (Invalid in T-SQL)
  - SELECT cust, SUM(amt) AS total FROM orders GROUP BY cust HAVING SUM(amt) > 2000000;
- Compare the following SQL statement
  - SELECT cust, SUM(amt) AS total FROM orders WHERE amt > 2000000 GROUP BY cust;

20