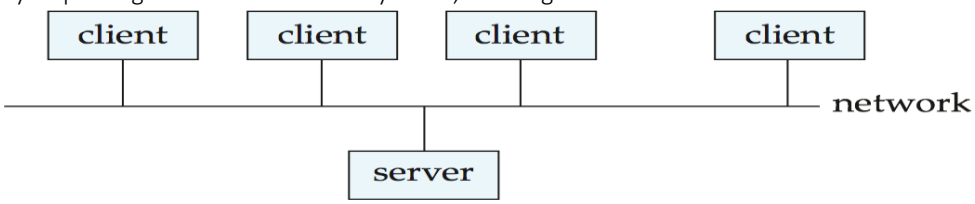
1.1 Database System Architectures
1.2 Classic Client/Server Architecture
1.3 Setting ODBC/JDBC for connecting database in MSSQL Server, Oracle
1.4 Developing Three-Tier Client/Server Architecture
1.5 Open Database Connectivity

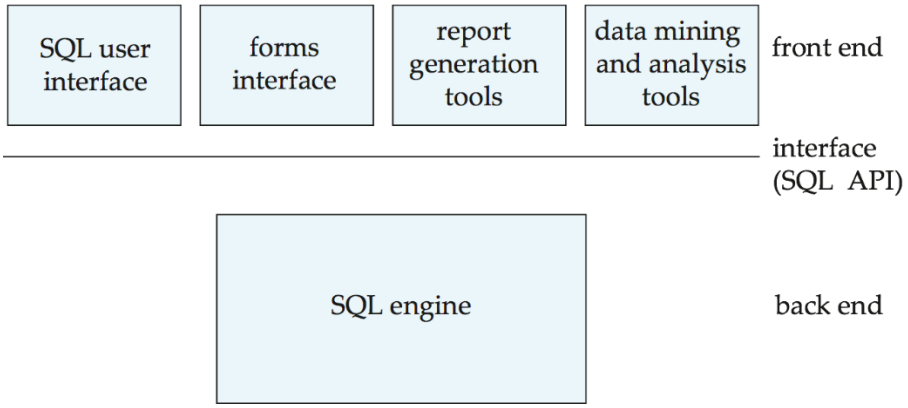
## 1.1 Database System Architectures

### I. Centralized and Client -Server Systems

- ✓ Run on a single computer system and do not interact with other computer systems.
- ✓ General-purpose computer system: one to a few CPUs and a number of device controllers that are connected through a common bus that provides access to shared memory.
- ✓ Single-user system (e.g., personal computer or workstation): desk-top unit, single user, usually has only one CPU and one or two hard disks; the OS may support only one user.
- ✓ Multi-user system: more disks, more memory, multiple CPUs, and a multi-user OS. Serve a large number of users who are connected to the system vie terminals. Often called *server* systems.
- ✓ Server systems satisfy requests generated at *m* client systems, whose general structure is shown below:




- ✓ Database functionality can be divided into:
  - o **Back-end**: manages access structures, query evaluation and optimization, concurrency control and recovery.
  - o **Front-end**: consists of tools such as *forms*, *report-writers*, and graphical user interface facilities.
- ✓ The interface between the front-end and the back-end is through SQL or through an application program interface.




- ✓ Advantages of replacing mainframes with networks of workstations or personal computers connected to back-end server machines:
  - o better functionality for the cost
  - o flexibility in locating resources and expanding facilities
  - o better user interfaces
  - o easier maintenance

### II. Server System Architectures

- ✓ Server systems can be broadly categorized into two kinds:
  - o **Transaction Servers** which are widely used in relational database systems, and
  - o **Data Servers**, used in object-oriented database systems

- **Transaction Servers**
  - ✓ Also known as middleware or *multi-tier* programming for the enterprise market
  - ✓ Also called **query server** systems or SQL *server* systems
    - o Clients send requests to the server
    - o Transactions are executed at the server
    - o Results are shipped back to the client.
  - ✓ Requests are specified in SQL, and communicated to the server through a *remote procedure call* (RPC) mechanism.

- ✓ A transaction server **works** when an application or application server requests for a specific data object residing on a database or database server on the network or Internet. The transaction server acts as an intermediary server that can ensure that the application or user receives the requested data from the database or the completion of that transaction. E.g. Microsoft Server (Viper) or Microsoft transaction server (MTS),.

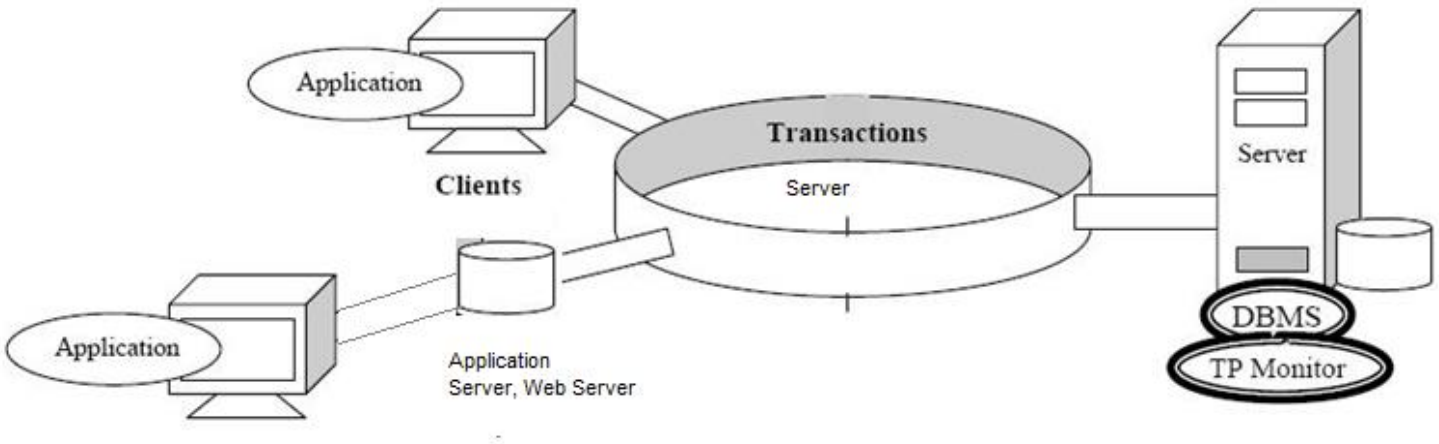
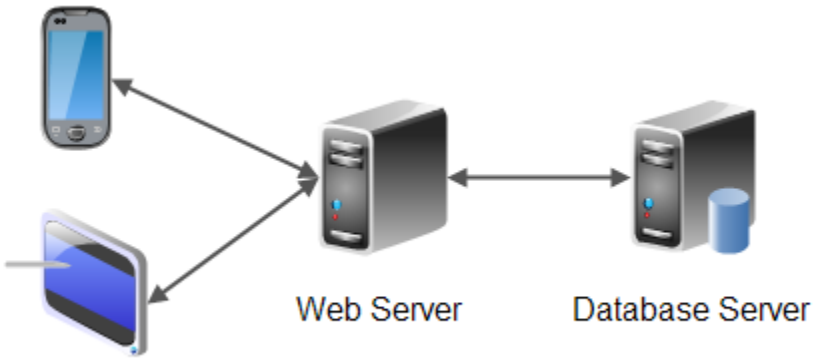


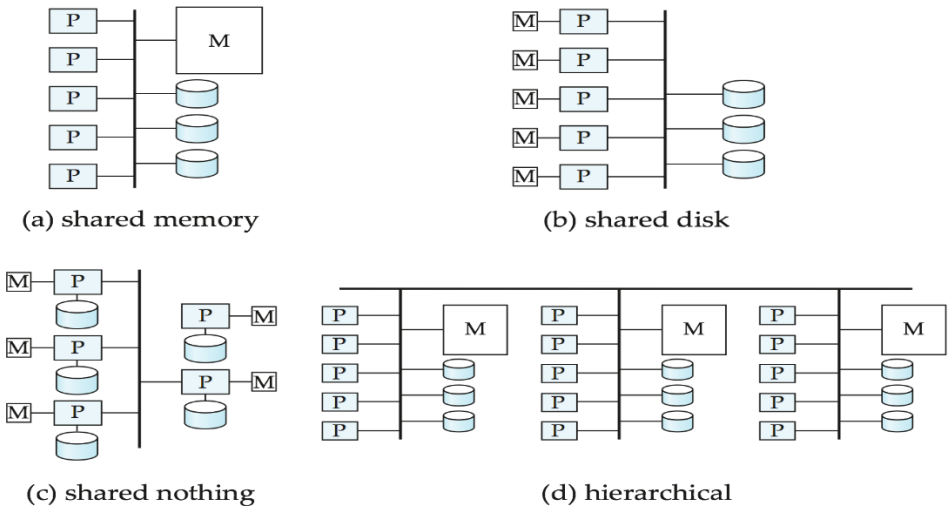

Fig. Transaction Server

- **Data Servers or Database Servers**
  - ✓ Used in high-speed LANs, in cases where
    - The clients are comparable in processing power to the server
    - The tasks to be executed are compute intensive.
  - ✓ Data are shipped to clients where processing is performed, and then shipped results back to the server. This architecture requires full back-end functionality at the clients.
  - ✓ Used in many object-oriented database systems




## III. Parallel Systems

- ✓ Parallel database systems consist of multiple processors and multiple disks connected by a fast interconnection network.
- ✓ A coarse-grain parallel machine consists of a small number of powerful processors
- ✓ A massively parallel or fine grain parallel machine utilizes thousands of smaller processors.
- ✓ Two main performance measures:
  - o throughput --- the number of tasks that can be completed in a given time interval
  - o response time --- the amount of time it takes to complete a single task from the time it is submitted
- ✓ **Shared memory** -- processors share a common memory
- ✓ **Shared disk** -- processors share a common disk
- ✓ **Shared nothing** -- processors share neither a common memory nor common disk
- ✓ **Hierarchical** -- hybrid of the above architectures




**Shared Memory**

- ✓ Processors and disks have access to a common memory, typically via a bus or through an interconnection network.
- ✓ Extremely efficient communication between processors — data in shared memory can be accessed by any processor without having to move it using software.

**Shared Disk**

- ✓ All processors can directly access all disks via an interconnection network, but the processors have private memories.
  - The memory bus is not a bottleneck
  - Architecture provides a degree of **fault-tolerance** — if a processor fails, the other processors can take over its tasks since the database is resident on disks that are accessible from all processors.
- ✓ Shared-disk systems can scale to a somewhat larger number of processors, but communication between processors is slower.

**Shared Nothing**

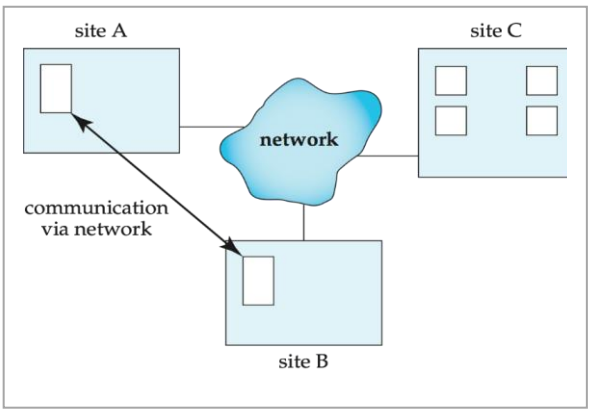
- ✓ Node consists of a processor, memory, and one or more disks. Processors at one node communicate with another processor at another node using an interconnection network. A node functions as the server for the data on the disk or disks the node owns.
- ✓ Examples: Teradata, Tandem, Oracle-n CUBE
- ✓ Data accessed from local disks (and local memory accesses) do not pass through interconnection network, thereby minimizing the interference of resource sharing.
- ✓ Shared-nothing multiprocessors can be scaled up to thousands of processors without interference.
- ✓ Main drawback: cost of communication and non-local disk access; sending data involves software interaction at both ends.

**Hierarchical**

- ✓ Combines characteristics of shared-memory, shared-disk, and shared-nothing architectures.
- ✓ Top level is a shared-nothing architecture – nodes connected by an interconnection network, and do not share disks or memory with each other.
- ✓ Each node of the system could be a shared-memory system with a few processors.
- ✓ Alternatively, each node could be a shared-disk system, and each of the systems sharing a set of disks could be a shared-memory system.
- ✓ Reduce the complexity of programming such systems by **distributed virtual-memory** architectures
  - Also called **non-uniform memory architecture (NUMA)**

## IV. Distributed Systems

- ✓ Data spread over multiple machines (also referred to as sites or nodes).
- ✓ Network interconnects the machines and data shared by users on multiple machines




i. Homogeneous distributed databases
   - Same software/schema on all sites, data may be partitioned among sites
   - Conditions must be satisfied as follows :
     - The operating system used at each location must be same or compatible.
     - The data structures used at each location must be same or compatible.
     - The database application (or DBMS) used at each location must be same or compatible.
   - Goal: provide a view of a single database, hiding details of distribution

ii. Heterogeneous distributed databases

- Heterogeneous systems are usually used when individual sites use their own hardware and software.
- Different software/schema on different sites

e.g. For example, one location may have the latest relational database management technology, while another location may store data using conventional files or old version of database management system. Similarly, one location may have the Windows operating system, while another may have UNIX.

- Goal: integrate existing databases to provide useful functionality
- Translations are required to allow communication between different sites (or DBMS)
- The heterogeneous system is often not technically or economically feasible. In this system, a user at one location may be able to read but not update the data at another location.

## V. Network Types

✓ Local-area networks (LANs) – composed of processors that are distributed over small geographical areas, such as a single building or a few adjacent buildings.

✓ Wide-area networks (WANs) – composed of processors distributed over a large geographical area.

## 1.2 Classic Client/Server Architecture

**Client/Server Architecture** The client/server architecture significantly decreased network traffic by providing a query response rather than total file transfer. It allows multi-user updating through a GUI front end to a shared database. Remote Procedure Calls (RPCs) or standard query language (SQL) statements are typically used to communicate between the client and server.

**Major Quality Attributes on Tier Architecture**

- **Reliability** : New methods are needed
- **Usability** : Separation makes usability easier to achieve
- **Security** : Tiers provide for security "walls"
- **Availability** : Tiers enhance redundancy
- **Scalability** : Fairly easy to expand services
- **Maintainability** : Good design – maintenance is easy … bad design – maintenance is hard E

**The following are the examples of client/server architectures.**

1) Two tier architectures A two-tier architecture is where a client talks directly to a server, with no intervening server. It is typically used in small environments (less than 50 users).

In two tier client/server architectures, the user interface is placed at user's desktop environment and the database management system services are usually in a server that is a more powerful machine that provides services to the many clients. Information processing is split between the user system interface environment and the database management server environment.

2) Three tier architectures the three-tier architecture is introduced to overcome the drawbacks of the two-tier architecture. In the three-tier architecture, a middleware is used between the user system interface client environment and the database management server environment.

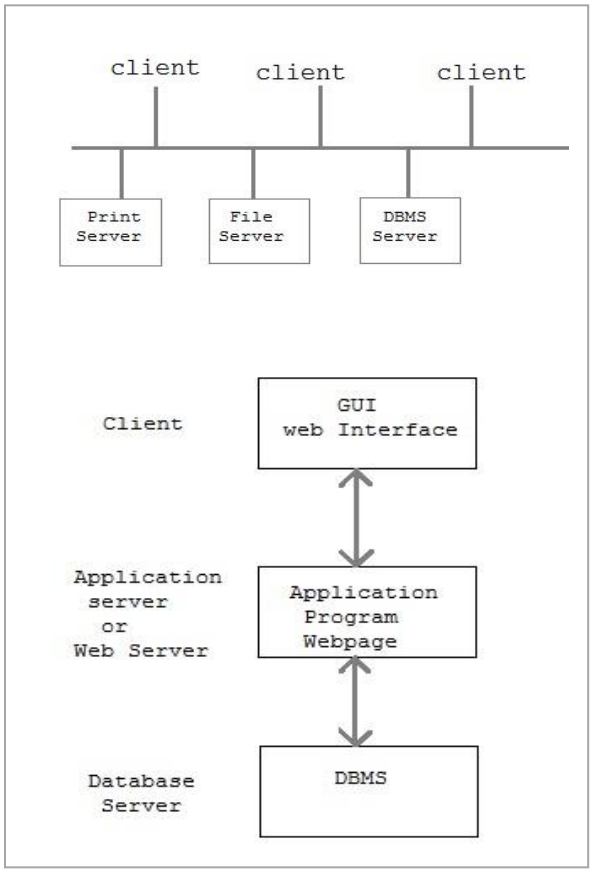
This middleware is implemented in a variety of ways such as transaction processing monitors, message servers or application servers. The middleware performs the function of queuing, application execution and database staging. In addition, the middleware adds scheduling and prioritization for work in progress.

The three-tier client/server architecture is used to improve performance for large number of users and also improves flexibility when compared to the two-tier approach.

The drawback of three tier architectures is that the development environment is more difficult to use than the development of two tier applications. The widespread use of the term 3-tier architecture also denotes the following architectures:

- Application sharing between a client, middleware and enterprise server
- Application sharing between a client, application server and enterprise database server.

i) **Three tier with message server**. In this architecture, messages are processed and prioritized asynchronously. Messages have headers that include priority information, address and identification number. The message server links to the relational DBMS and other data sources. Messaging systems are alternative for wireless infrastructures.



ii) **Three tier with an application server** This architecture allows the main body of an application to run on a shared host rather than in the user system interface client environment. The application server shares business logic, computations and a data retrieval engine. In this architecture applications are more scalable and installation costs are less on a single server than maintaining each on a desktop client. 3-tier architecture provides :

- A greater degree of flexibility
- Increased security, as security can be defined for each service, and at each level
- Increased performance, as tasks are shared between servers

### Comparison of Architectures

| Architecture | Pros | Cons |
|---|---|---|
| One tier | Simple Very high performance Self-contained | No networking – can't access remote services Potential for spaghetti code |
| Two tiers | Clean, modular design Less network traffic Secure algorithms Can separate UI from business logic | Must design/implement protocol Must design/implement reliable data storage |
| Three tiers | Can separate UI, logic, and storage Reliable, replicable data Concurrent data access via transactions Efficient data access | Need to buy DB product Need to hire DBA Need to learn SQL Object-relational mapping is difficult |
| N tiers | Support multiple applications more easily Common protocol/API | Less inefficient Must learn API (CORBA, RMI, etc.) Expensive products More complex, more faults Load balancing is hard |

**The basic characteristics of client/server architectures are:**

1) Combination of a client or **front-end portion** that interacts with the user, and a server or **back-end portion** that interacts with the shared resource. The client process contains **solution-specific logic** and provides the interface between the user and the rest of the application system. The server process acts as a **software engine** that manages shared resources such as databases, printers, modems, or high powered processors.

2) The front-end task and back-end task have fundamentally different requirements for computing resources such as processor speeds, memory, disk speeds and capacities, and input/output devices.

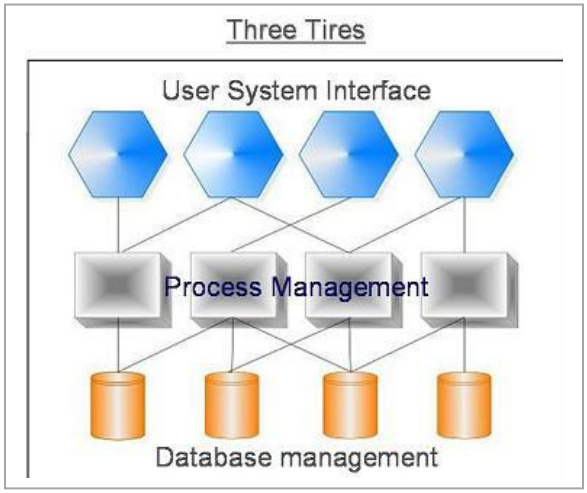
3) The environment is typically **heterogeneous** and multivendor. The hardware platform and operating system of client and server are not usually the same.Client and server processes communicate through a well-defined set of standard application program interfaces (API's) and RPC's.

4) An important characteristic of client-server systems is scalability. They can be scaled horizontally or vertically. Horizontal scaling means adding or removing client workstations with only a slight performance impact. Vertical scaling means migrating to a larger and faster server machine or multiservers.

## 1.4 Developing Three-Tier Client/Server Architecture

In the field of Web development, Three Tier is often employed in reference to web sites. In particular, Electronic commerce web sites are used in this system. This type of web site is usually built utilizing the following Three Tiers:

1. A **front end** web server, which serves static content
2. The **middle level** is typically an Application server. It might use, for example, a Java EE platform.
3. A **back end Database**, which will contain both the Database management system and the Data sets. This manages the data and provides access to it.




Three Tier Architecture contains the following tiers or levels:

1. Presentation Tier : User System Interface. This user interface is often a graphical one accessible through a web browser or web-based application and which displays content and information useful to an end user. This tier is often built on web technologies such as HTML5, JavaScript, CSS.

2. Application/Logic/Business Logic/Transaction Tier : Process Management. The application tier contains the functional business logic which drives an application's core capabilities. It's often written in Java, .NET, C#.

3. Data Tier: Database Management. The data tier comprises of the database/data storage system and data access layer. Examples of such systems are MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MongoDB, etc. Data is accessed by the application layer via API calls.

- Third tier( Database Management): contains database management functions. Its purpose is to optimize data and file services without having to result to the usage of proprietary database management system languages. This component makes sure that the data is consistent throughout the

environment. In order to do so, it utilizes such features as data locking, replication, and consistency. The connectivity among tiers can be changed dynamically, but of course this depends on the user's request for services and data.

- The middle tier on the above model provides process management services which will be shared by multiple applications. These services may include process performing, process resourcing, process development, and process monitoring. It is also called the application server. It improves scalability, reusability, flexibility, and maintainability via the centralization of process logic. This centralization makes change management and administration a lot simpler by localizing the functionality of the system so that changes only have to be written one time. They are then placed on the central tier and made available throughout the systems. With other architectural designs, it would be necessary to write the change in to each and every application.

### What Are the Benefits of Using a 3-Layer Architecture?

- **Speed of development**: by allowing teams to focus on their core competencies by specialize in front- end, server back-end, and data back-end development, by modularizing
- **Scalability, Performance**, : By separating out the different layers you can scale each independently depending on the need at any given time. For example, if you are receiving many web requests but not many requests which affect your application layer, you can scale your web servers without touching your application servers.
- **Availability**: By having disparate layers you can also increase reliability and availability by hosting different parts of your application on different servers and utilizing cached results.

## 5 Open Database Connectivity

**Open Database Connectivity** (**ODBC**) is a standard application programming interface (API) for accessing database management systems (DBMS). The designers of ODBC aimed to make it independent of database systems and operating systems. An application written using ODBC can be ported to other platforms, both on the client and server side, with few changes to the data access code.
ODBC accomplishes DBMS independence by using an *ODBC driver* as a translation layer between the application and the DBMS. The application uses ODBC functions through an *ODBC driver manager* with which it is linked, and the driver passes the query to the DBMS. An ODBC driver can be thought of as analogous to a printer driver or other driver, providing a standard set of functions for the application to use, and implementing DBMS-specific functionality

- An **ODBC**-enabled "front-end" or "client" desktop application, also known as an "ODBC Client." This is the application that the computer-user sees on the computer screen.
  -or-
- An **ODBC Driver** for a "back-end" or "server" DBMS (Database Management System). This is the DBMS application that resides on a computer that is used to store data for access by several users.

To use ODBC, the following three components are required:
***ODBC CLIENT*** - an ODBC-enabled front-end (also called ODBC client) - Examples: Microsoft Access, an application created with Access, an application created with Microsoft Visual Basic, an application created with C+Win SDK+ODBC SDK, or ODBC-enabled applications from other vendors (such as Lotus).
***ODBC DRIVE***R - an ODBC Driver for the ODBC Server. The ODBC Driver Catalog contains an extensive listing of ODBC Drivers. For example, the Microsoft ODBC Driver Pack is a collection of seven ODBC Drivers ready to be used or bundled with ODBC clients. A SQL Server ODBC Driver is included with Access, and Informix is working on an ODBC driver for Informix. To obtain an ODBC Driver Catalog, call the Microsoft Order Desk at (800) 360-7561. If you are outside the United States, contact your local subsidiary. To locate your subsidiary, see the Microsoft World Wide Offices Web site at:
http://www.microsoft.com/worldwide/
Any ODBC client can access any DBMS for which there is an ODBC Driver. DBMS SERVER is a back-end or server DBMS, for example SQL Server, Oracle, AS/400, Foxpro, Microsoft Access, or any DBMS for which an ODBC driver exists.

### How do these three components interact?

The ODBC client uses a language or vocabulary of commands (which is referred to as "ODBC") to request data from, or to send data to, the back- end or server DBMS. However, the DBMS doesn't understand the ODBC client request until the command passes through the ODBC Driver for that specific DBMS. This ODBC driver is software that resides on the front-end. The ODBC driver translates the command into a format that the ODBC Server can understand. The ODBC Server sends the answer back to the ODBC Driver, which translates the answer into a format that the ODBC Client can understand.


### Examples of ODBC in Use

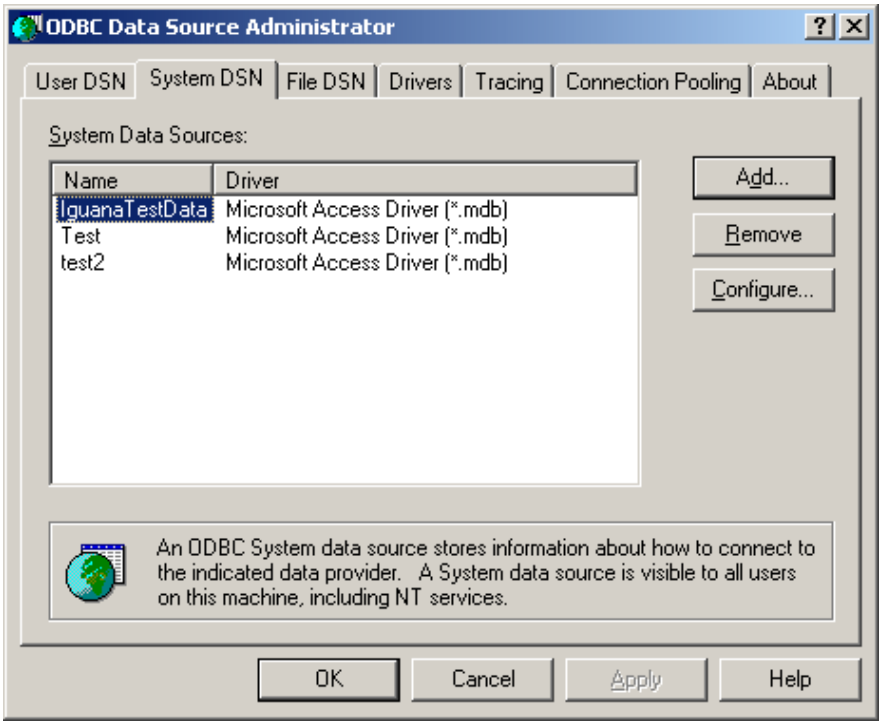
- Access front-end accessing data from a Oracle back-end using the Oracle ODBC Driver, which ships with Access 1.1.

- Visual Basic front-end accessing data from a dBASE back-end using the dBASE ODBC Driver, which is part of the MS ODBC Database Drivers Pack.

## 1.3 Setting ODBC/ JDBC for connecting database in MSSQL Server, Oracle

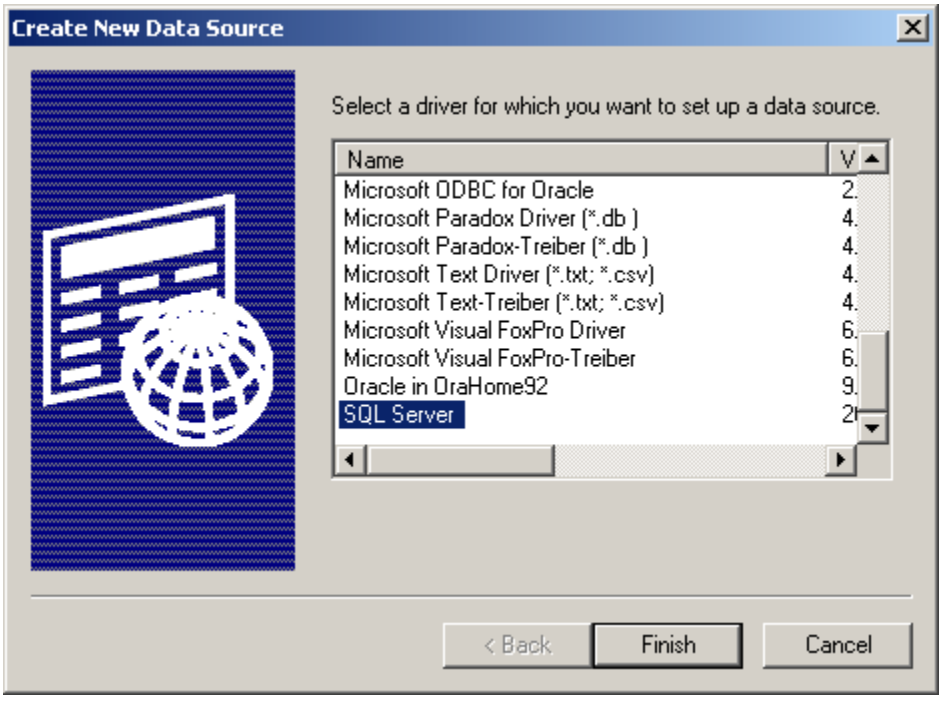
### Setting up ODBC in MSSQL

- **Download** a Microsoft SQL Server JDBC Driver *http://msdn.microsoft.com/data/jdbc*
- Microsoft provides ***sqljdbc.jar*** and ***sqljdbc4.jar*** class library files
- Click **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)**. The ODBC Data Source Administrator window appears.



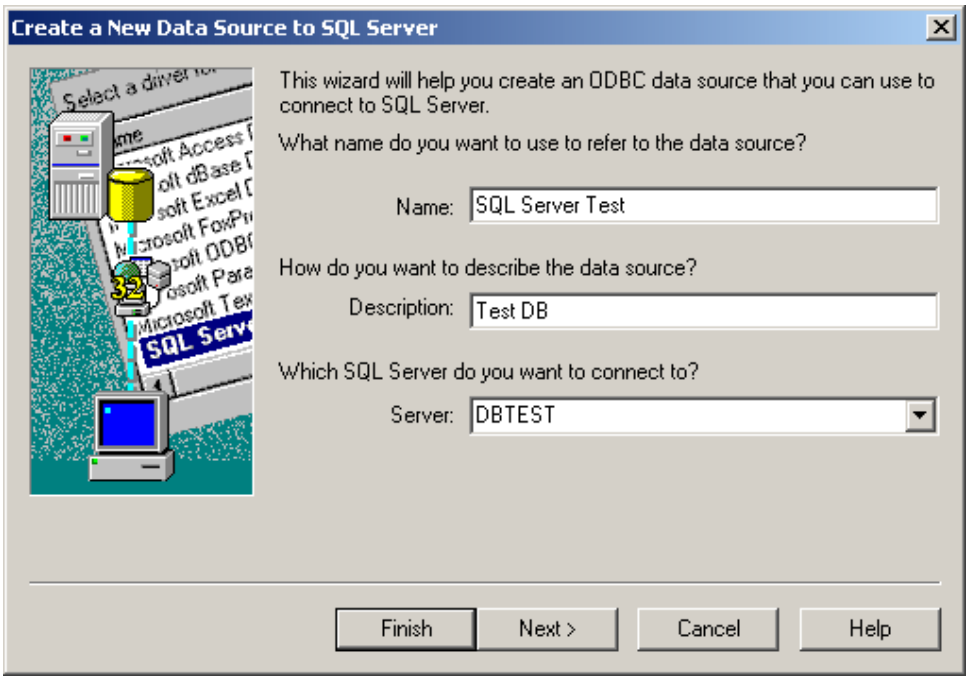

ODBC Data Source Administrator Window

- Click the **System DSN** tab.
- Click **Add**. The Create New Data Source window appears.



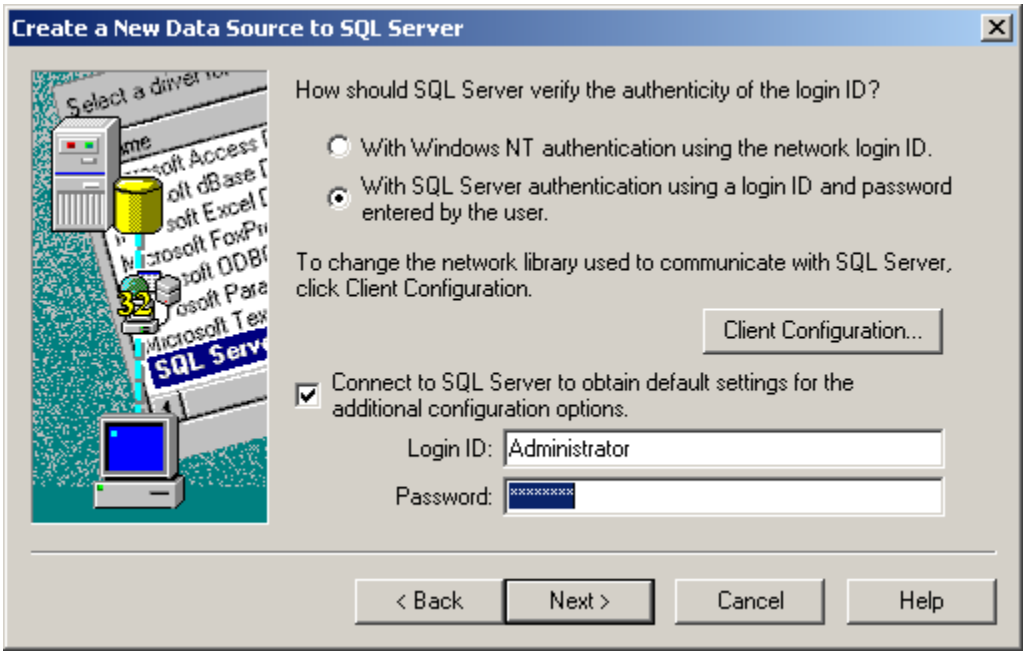
Selecting SQL Server

- Highlight **SQL Server** and click **Finish**. The Create a New Data Source to SQL Server window appears.
- Enter the **Name** of the Data Source. This is the name that will appear under System Data Sources in the System DSN tab.
- **(Optional)** Enter a **Description** of the data source, if desired.
- Specify the **SQL Server** that you want to connect to, and click **Next**.



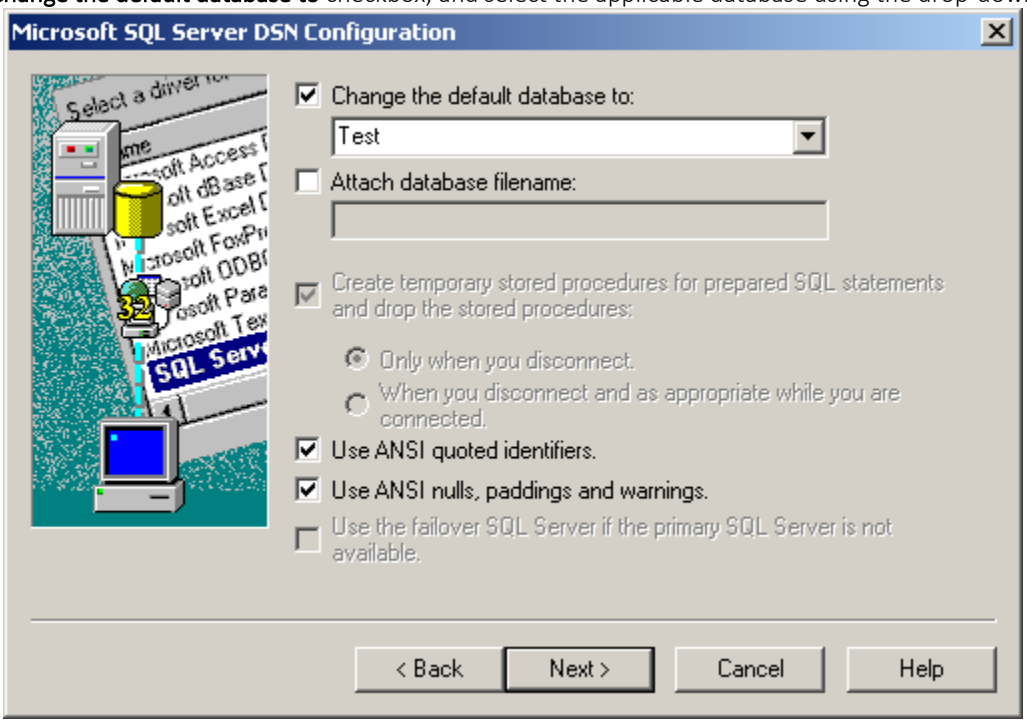

Creating a New Data Source For SQL Server

- Ensure that **With SQL Server authentication using a login ID and password entered by the user** and **Connect to SQL Server to obtain default settings for the additional configuration options** are selected.
- Enter the username and password information for the SQL account you are using, and click **Next**. This account should have full access to the database.



Ensuring the Correct Options are Selected

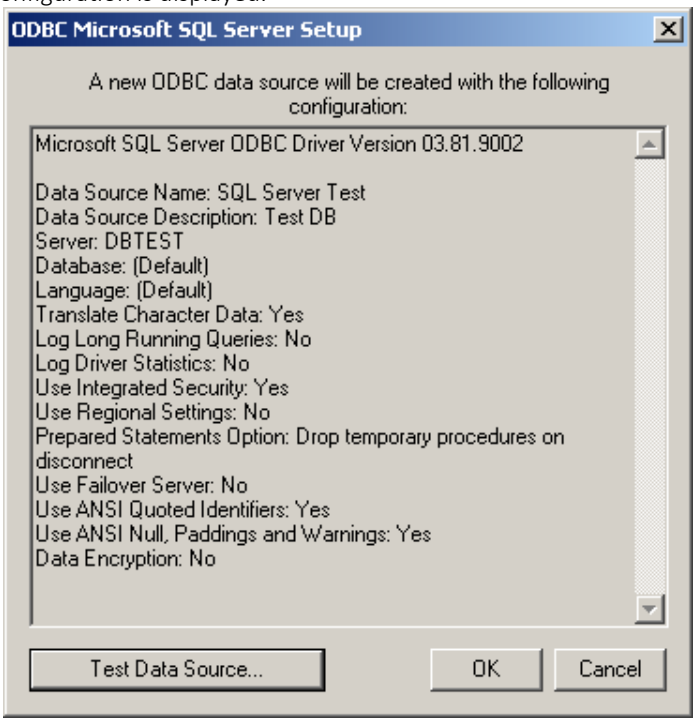
- Select the **Change the default database to** checkbox, and select the applicable database using the drop-down arrow.




Changing the Default Database

- Click **Next** > **Finish** to complete the setup.

A summary of your MS SQL Server configuration is displayed.

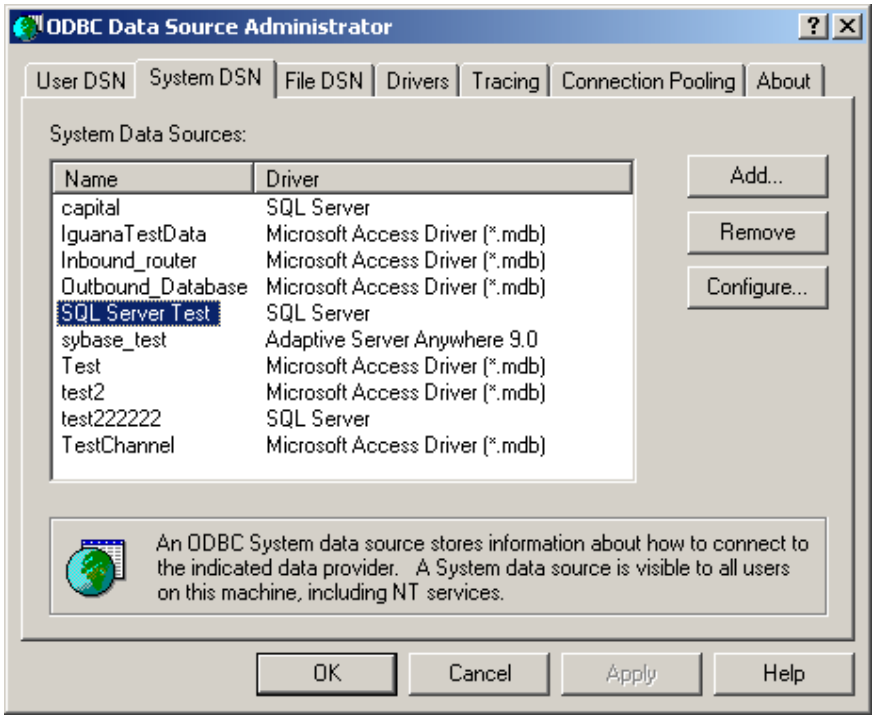



Summary of SQL Server Configuration Settings

- Click **Test Data Source** to ensure that you can connect to the specified MS SQL Server. A window displays the results of the test, as shown below:


SQL Server ODBC Test Results

- Click **OK** to close the results window.
- Click **OK** again to close the summary of your MS SQL Server configuration. The ODBC Data Source Administrator window displays your created data source, as shown below.



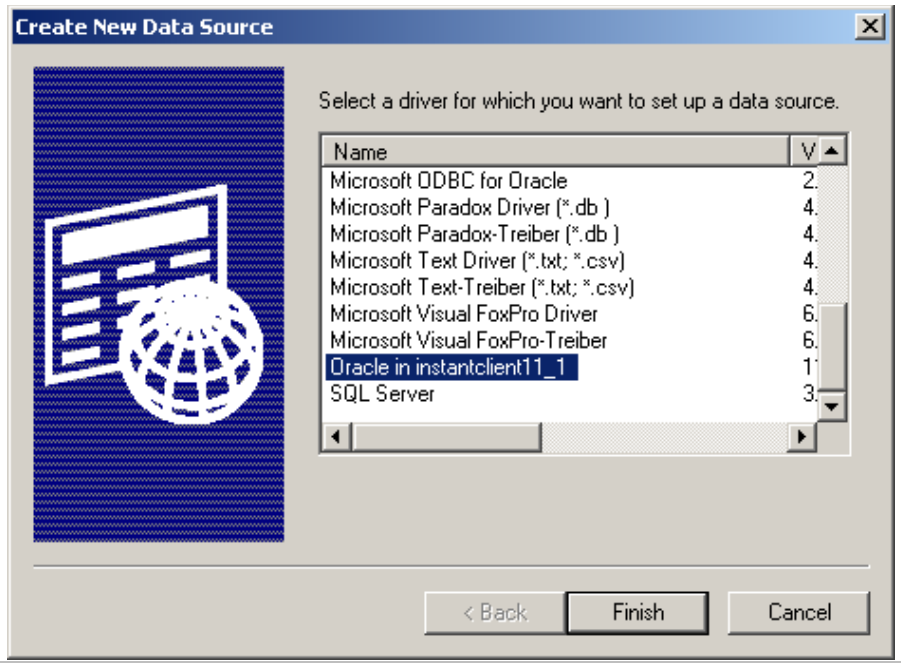

System Data Sources, Showing Created SQL Server Data Source

The next step is to modify the Iguana Log On properties so that you can connect to your SQL Server database.

## Setting Up an ODBC Data Source for Oracle

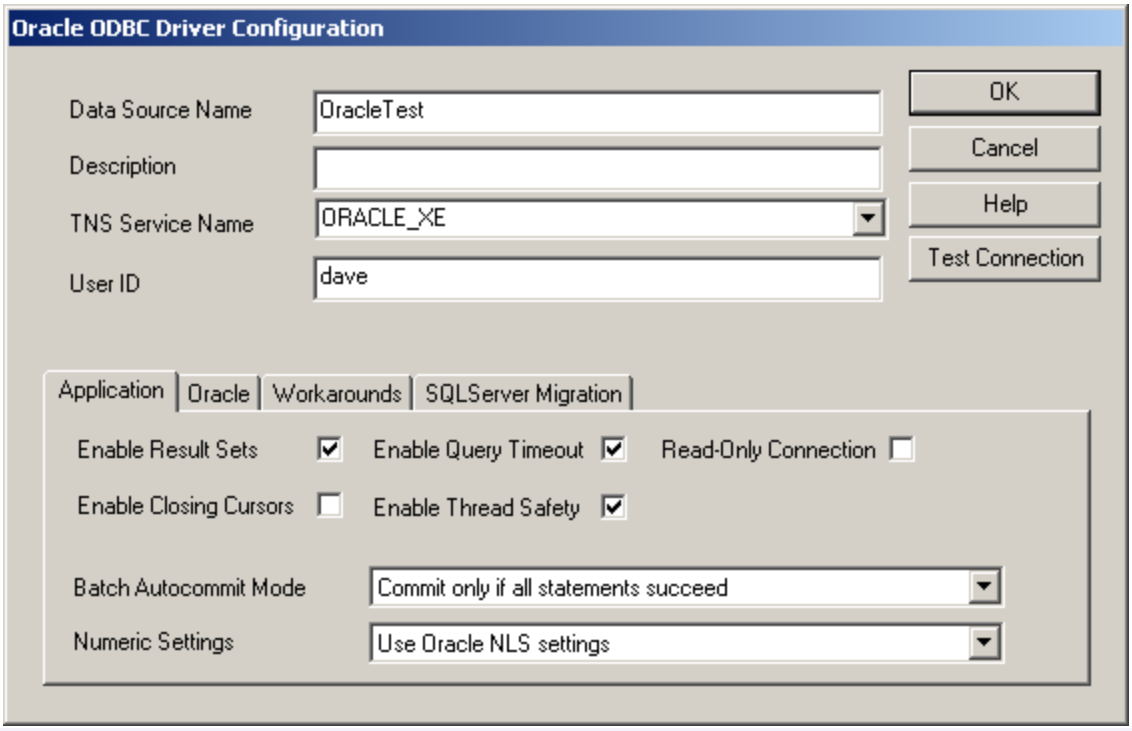
To set up an ODBC data source for your Oracle client, perform the following steps:

- From Windows, click **Start**, then select **Settings**, **Control Panel**, **Administrative Tools** and **Data Sources (ODBC)**. The ODBC Data Source Administrator window appears.
- Click the **System DSN** tab, and click **Add** to create a new data source. The Create New Data Source window appears. Your Oracle client driver should appear on this list:



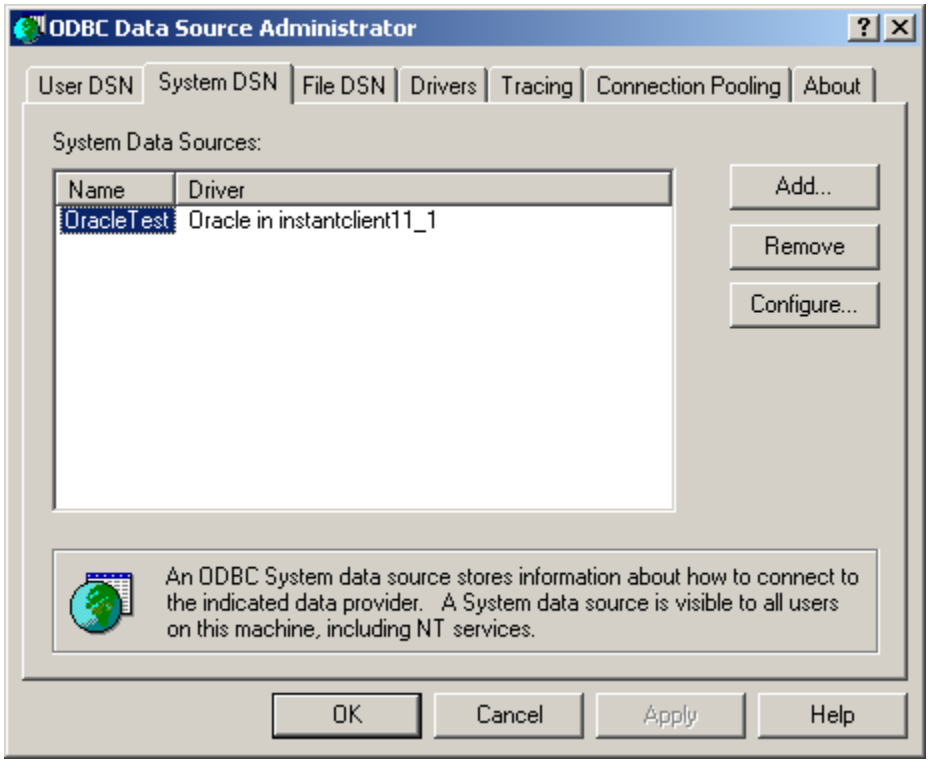

If you do not have an Oracle client, you can install the Oracle Database Instant Client, which enables you to use ODBC to interact with your Oracle server. See Setting Up an Oracle Database Instant Client for more details.

- Select your Oracle client driver from the list, and click **Finish**. The ODBC Oracle Driver Configuration window appears.
- In the **Data Source Name** field, type the name that you want to use for this ODBC data source.
- In the **TNS Service Name** field, type the name of your Oracle service. This is the name that is specified in the **tnsnames.ora** file that is defined for your Oracle client.
- In the **User ID** field, type the username that you use to log into the Oracle database. The Oracle ODBC Driver Configuration window should now look something like this:




If you want to ensure that your connection is working, click **Test Connection**. You will be asked to supply the password for y username. If the connection is defined properly, a popup box will appear, notifying you that the connection was successful

- Click **OK** to create the ODBC data source for your Oracle client. The new data source now appears in the ODBC Data Source Administrator window:




The data source is now ready for use. For details on how to configure an Iguana channel to use an Oracle ODBC data source, see Using an ODBC Oracle Data Source.

## Connecting to SQL Server with the JDBC Driver

One of the most fundamental things that you will do with the Microsoft JDBC Driver for SQL Server is to make a connection to a SQL Server database. All interaction with the database occurs through the SQLServerConnection object, and because the JDBC driver has such a flat architecture, almost all interesting behavior touches the SQLServerConnection object.

If a SQL Server is only listening on an IPv6 port, set the java.net.preferIPv6Addresses system property to make sure that IPv6 is used instead of IPv4 to connect to the SQL Server:

```
System.setProperty("java.net.preferIPv6Addresses", "true");
```

The topics in this section describe how to make and work with a connection to a SQL Server database.