Fundamental Common Technologies

Advanced Information and Communication

Technology Training (Mandalay)

[Fundamental Database]

Contents at a Glance

Table of Contents

1. Database and DBMS

1.1. Types of Databases

Almost all the databases today are implemented by Database Management System (DBMS). DBMS is collection of software programs that manages the database structure, query and access to the data which is stored in the database. A DBMS will be able to support several types of databases in terms of such as number of users or/and usage of the data. Those databases can be classified according to different aspects as defined in the table below:

Table 1 - Types of Databases

Category	Types of Database	Description
Number of Users	Single-user database	Supports one user at a time
	Multi-user database	Supports concurrent users at a time
Data Location	Centralized database	Data located in a single site
	Distributed database	Data distributed across different sites
Data Usage	Operational database	Supports day-by-day operation for the organization (OLTP)
	Data warehouse	Storing data used to generate information for analyses (OLAP)
Data Availability	Standalone database	Supports no fault tolerant mechanism
	Clustering database	Supports fault tolerant mechanism

OLTP: On-Line Transaction Processing

OLAP: On-line Analytical Processing

Table below is, as example, a list of the database products (DBMS) available today and its characteristics are categorized based on the types of database defined in Table above. The table can be dividend into two according to the software license model such as commercial software license and open source software license. How to select a DBMS is very much depends on system's requirements, capacity planning of data volume as well as you need to take cost for software development into account. MySQL and PostgreSQL are most popular open source database management systems however there are differences between the licensing models. PostgreSQL adapts very flexible almost unlimited usage license called BSD license, but for MySQL, license need to be selected either GNU GPL license or commercial license depending on the usages of the database.

Table 2 - Database products comparison

Database Products	Market price/ License type	Number of Users	Data Location	Data Usage	Data Availability
Microsoft Access 2003	Approximately USD200/User	Single	Centralized	Operational	Standalone
Microsoft SQL Server 2005	Approximately USD5,000/CPU	Multi	Distributed	Operational Data warehouse	Clustering
Oracle Database 10g	Approximately USD5,000/CPU	Multi	Distributed	Operational Data warehouse	Clustering
MySQL 5 (Community)	N/A GNU GPL	Multi	Distributed	Operational Data warehouse	Clustering
MySQL 5.1 (Community)	N/A GNU GPL	Multi	Distributed	Operational Data warehouse	Clustering
PostgreSQL 8	N/A BSD License	Multi	Distributed	Operational Data warehouse	Clustering

GNU GPL: GNU General Public License

BSD License: Berkeley Software Distribution License

Table 3 highlights major differences in between MySQL and PostgreSQL. However as can be seen, current stable version as of today for both open source RDBMS are mostly the same. In other words, PostgreSQL has been improving dramatically its performance to catch up with MySQL, as well as MySQL has been widely expanding its functionalities that are provided by PostgreSQL.

Table 3 - MySQL .vs. PostgreSQL

Features	MySQL 5.0	PostgreSQL 8.2
Licensing Model	Dual (GPL/Commercial)	BSD
Multi-platform support	Yes	Yes
Multi-user support	Yes	Yes
Standard SQL (SQL:92/99) support	Yes	Yes
Transaction support	Yes *	Yes
Foreign-key constraint	Yes *	Yes
Stored Procedure and Function, Trigger	Yes	Yes

Database replication	Yes	Yes

Note: * Available only for InnoDB storage engine

1.2. Types of DBMS

1.2.1. DBMS Market Share

Table below shows a list of commercial DBMS products market share in Japan (2005):

Table 4 - DBMS Market Share

No	DBMS	Sales (Billion JP Yen)
1	RDBMS	151.1 (85.7%)
2	Pre/Post RDBMD	13.9 (7.9%)
3	End-User DBMS	10.4 (5.9%)
4	OODBMS/XML DBMS	0.9 (0.5%)
		Total: 176.3 (100%)

Source: IDC Japan, 2005 (http://www.idcjapan.co.jp/)

Pre/Post RDBMS refers mainly to DBMS for mainframe computers. End-User DBMS refers to desk top (stand-alone) DBMS such as Microsoft Access. As can be seen, majority of share is still taken by RDBMS however OODBMS/XML DBMS market share would be grown according to some of analysis in next few years time.

1.2.2 Relational Database Management System (RDBMS)

Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by Edgar F. Codd in 1970. Relational databases are the most common kind of database in use today thus in this session focuses RDBMS as main subject to learn.

1.2.3 XML Database Management System (XML DBMS)

There are two types of XML databases are defined such as XML-enabled and Native XML database. XML-enabled database is, for example, additional functions to existing relational database to accept XML document as input to be stored and rendering those stored information to output as XML document. However “XML Database” nowadays means in most cases is Native XML database that should have the internal model of databases depends on XML and uses XML documents as the fundamental unit of storage. It also supports XQuery (includes XPath) as query language to perform database accessing. Followings are some of the Native

XML databases in the market:

NeoCore: http://www.xpriori.com/

DB2 9: http://www-306.ibm.com/software/data/db2/9/ (Hybrid DBMS with RDBMS)

1.2.4 Object Oriented Database Management System (OODBMS)

An Object-Oriented Database Management System or OODBMS is a database management system that supports DBMS functions combined with object programming language capabilities. It should be consistent with the current features of object-oriented programming languages such as Java, C++ and C#. In other words, ODBMS uses exactly the same model as object-oriented modeling and programming languages. Therefore, it is not necessary to map application objects with two-dimension tables like using relational database. Followings are some of the OODBMS in the market:

Object Store: http://www.progress.com/objectstore/index.ssp

Caché: http://www.intersystems.com/cache/index.html

db4objects: http://www.db4o.com/

1.3 Database Systems

1.3.1 Database Systems Environment

DBMS nowadays act very important roles in most of especially mission critical information systems such as Web-based enterprise applications. The figure below depicts how those enterprise applications segregate its layers. In this layering, DBMS sits in between application layer which handles business processes and storage layer that is responsible persisting information stored in the database. Since the DBMS must be able to process multi users concurrent access and provides high availability as well as and fault tolerant mechanism, technologies such as connection pooling, database replication and load balancing are also need to take into account when you design the database environment. A capacity planning is normally conducted in the beginning of project to estimates such as transaction volumes, data growth and numbers of concurrent users in peak hours to identify what technologies are required in the database environment.

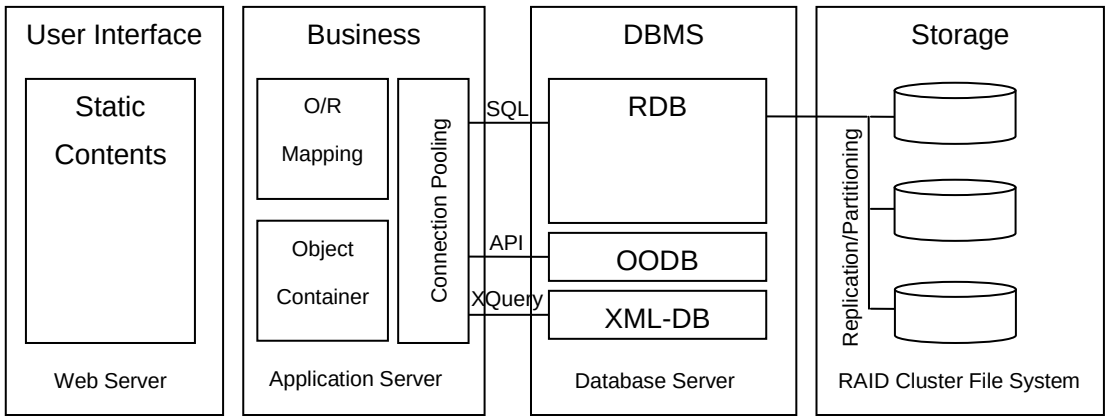


Figure 1 - Enterprise Web Application Layering and Database Systems

1.3.2 DBMS Functions

DBMS provides important functions to guarantee the integrity and consistency of the data in the database such as:

- Data dictionary management
  Provides to mechanism to define the schema of the database. A data dictionary stores definition of the elements and their relationships.

- Database query language and Application Programming Interface (API)
  Provide data access through a non-procedural query language such as Structured Query Language (SQL) as de fact and procedural languages such as Java, PL/pgSQL (PostgreSQL).
- Transaction management and multi user access control
  Provide data integrity and data consistency. (Minimize data redundancy and maximize data consistency)
- Security management
  Provides to create security system that enforces user security and data privacy. It allows a database administrator to determine access levels of users and access rights to each database object.
- Data backup and recovery management
  Provides backup and data recovery to ensure data safety and integrity. Each DBMS provides own utilities to carry out those backup and recovery operations.

Exercise 1. - Database and DBMS selection for case study projects

Discuss with your group members to find out what are the most suitable DBMS to choose for following case study projects. Also discuss to come up with overall database system design for each project.

- In-house accounting database application for SME with minimum cost for development. The application should be developed shortest time frame. The application will be used only by accounting section staff. The application should be able to provide easy-to-use look & feel user interface for Microsoft Windows/Office users.

- E-Commerce on-line book shop Web Site development project with minimum cost for development. Estimated members are 10,000 from initial year and projected growth rate of 20 % in next 5years. However due to the nature of the business 24X7X365 availability of the system is required.

- Internet banking system development project for an international bank. Their branches are across overseas all together about 100. High security from any Internet risks, high availability and high sustainability are must. All the data transaction must be able to keep for audit purpose. High level technical support must be provided by experts within minimum turnaround time.

2 Introduction to PostgreSQL

2.1 Features

PostgreSQL (http://www.postgresql.org/) is one of the most popular open source relational database management systems (RDBMS), or more precisely it is called as object-relational database management systems (ORDBMS) which were released under a flexible BSD License. It was originally called **Ingress** which was developed by Michael Stonebraker at UC Berkeley in 1977. After went through a long history to go, PostgreSQL today that is pronounced as "post-gress-Q-L" dramatically improved its performance and enhanced its functionalities. The latest version available today is 8.2.1 (As of January 2007) which provides following features:

Table 5 - Brief History of PostgreSQL

Year	Product	Description
1977-1985	Ingres	Spawned Relational Technologies, purchased by Computer Associate
1986-1994	Postgres	Spawned Illustra, purchased by Informix
1994-1995	Postgres95	Added standard SQL, spawned PostgreSQL
1996-	PostgreSQL	PostgreSQL (Version 8.2 released in January 2007)

Features:

- Support multi platform; it runs on all major operating systems, including Linux, UNIX and Windows.
- Support transaction management; full ACID (Atomicity, Consistency, Isolation and Durability) transaction compliant.
- Support foreign keys, joins, views, triggers, and stored procedures (in multiple languages).
- Support most of SQL92 and SQL99 data types. It also supports storage of binary large objects. Native application programming interfaces for C/C++, Java, .Net, Perl, Python, Ruby, Tcl, and ODBC.
- Support technologies for an enterprise class database using such as Multi-Version Concurrency Control (MVCC), point in time recovery (PITR),
- Support technologies for high availability and high scalability table spaces, asynchronous replication, nested transactions (save points), online/hot backups, a query planner/optimizer, and write ahead logging for fault tolerance.

PostgreSQL is maintained by following version numbers such as major and minor versions: Major Version. Major Version. Minor Version (Example: 8.2.1)

2.2 Architecture Overview

PostgreSQL is designed based on the client/server type architecture such as front-end (client) and back-end (server). Front-end application, first of all, need to establish the connection to back-end PostgreSQL server with certain protocols such as TCP/IP or UNIX Domain Socket as can be seen the figure below:

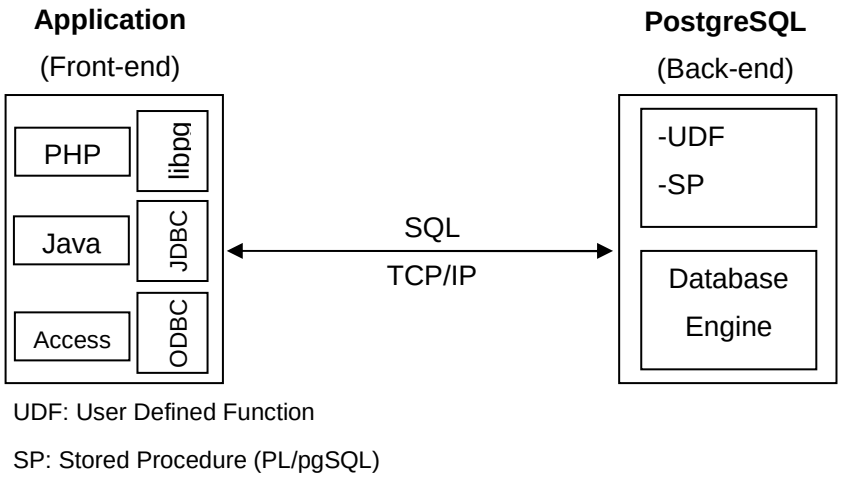


Figure 2 - PostgreSQL Client/Server Architecture overview

PostgreSQL back-end consists of several internal processes as depicted in figure below: "postmaster" is a stationed process to manage all back-end processes. This is singleton process creates "postgres" process each time when front-end client established the connection to back-end server.

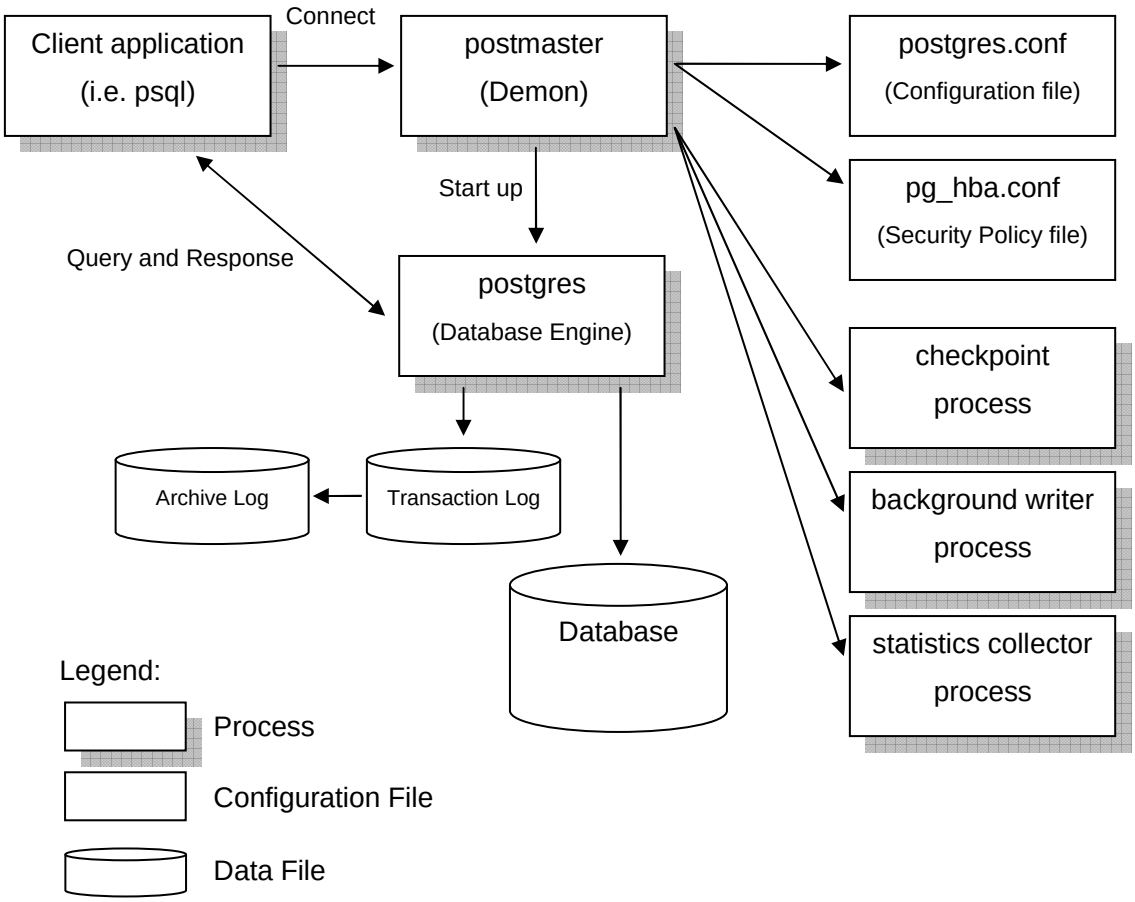


Figure 3 - PostgreSQL Process Architecture overview

2.3 Development Tools

Followings are commonly used PostgreSQL front-end development commands and tools:

- Initdb

initdb is a command to initialize a PostgreSQL database cluster.

Table 6 – initdb command

Command	Description
initdb	Initialize database cluster --pgdata=database directory ,-D database directory

Example:

```
postgres> initdb –-no-locale –-encoding=UTF-8
```

$PGDATA is an environment variable that points to the location of database cluster and default (if PostgreSQL was installed by package) located at $PGDATA=/var/lib/pgsql/data/. $PGDATA contains sub directories for such as data, archive and log and configuration files as can be seen in below.

$PGDATA (/var/lib/pgsql/data/)

PG_VERSION	PostgreSQL version number file
pg_hba.conf	PostgreSQL server configuration file
postgresql.conf	Host based authentication file
base/	Database files
global/	Common objects
pg_clog/	Transaction commitment logs
pg_xlog/	Write-Ahead- Log (WAL) logs
pg_subtrans/	Sub-transaction status
pg_tblspc/	Symbolic link to table spaces
pg_twophase/	Two-phase commitment status
pg_multixact/	Multi-transaction status

Figure 4 - $PGDATA (PostgreSQL Database Cluster)

- pg_ctl

pg_ctl is a command to control PostgreSQL server process.

Table 7 - pg_ctl command

Command	Description
pg_ctl start	Start server
pg_ctl status	Check status of server
pg_ctl restart	Restart server
pg_ctl reload	Reload server configuration
pg_ctl stop	Stop server

Examples:

```
postgres> pg_ctl status
postgres> pg_ctl start
postgres> pg_ctl stop
postgres> pg_ctl –m smart stop          ‘Quit after all clients have disconnected
postgres> pg_ctl –m first stop          ‘Quit directly, with proper shutdown
postgres> pg_ctl –m immediate stop      ‘Quit without complete shutdown
```

- psql

psql is a front-end application to query to PostgreSQL database.

Table 8 - psql commands

Command	Description
\l	Display list of databases
\d	Display list of tables
\d TABLE NAME	Display table definition
\c DATABASE NAME	Connect other database
\h SQL COMMAND	Display SQL command help
\?	Display psql help
\q	Quit psql

- createdb/dropdb

createdb and dropdb are command to create and drop database.

Table 9 - createuser/dropuser commands

Command	Description
createdb DB NAME	Create database
dropdb DB NAME	Drop database

Examples:

```
postgres> createdb s-ct-d
postgres> dropdb s-ct-d
```

- createuser/dropuser

createuser and dropuser are command to crate and drop user.

Table 10 - createuser/dropuser commands

Command	Description
createuser USER NAME	Create user
dropuser USER NAME	Drop user

Examples:

```
postgres> createuser –P aict
postgres> dropuser aict
```

- pgAdmin III

pgAdmin III (http://www.pgadmin.org/) is the most popular and feature rich administration and development platform for PostgrSQL. Easy to manage GUI is provided to administrate databases just like what psql can do.

RPM for SUSE Linux can be downloaded from following site:
http://rpm.pbone.net/index.php3/stat/4/idpl/3007466/com/pgadmin3-1.4.3-1.guru.suse101.i686.rpm.html

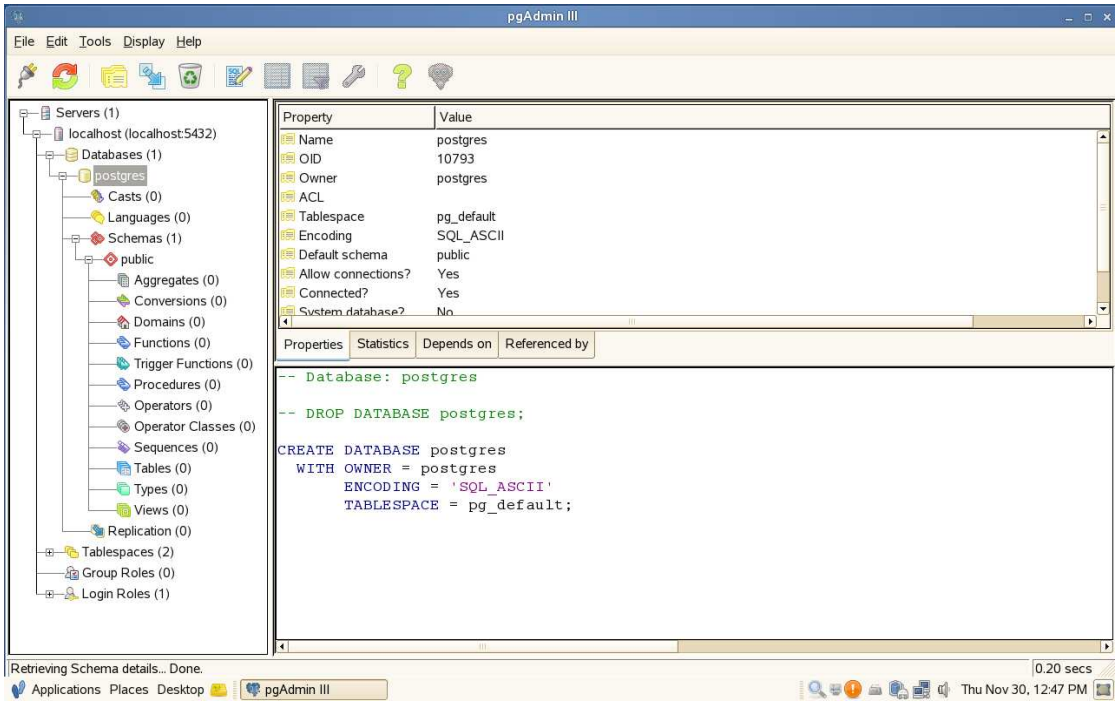


Figure 5 - pgAdmin III Look & Feel

- Others

There are very useful PostgreSQL **technical documents** for self-study as well as references for development are available online from the Web site below:
http://www.postgresql.org/docs/

Exercise 2. - PostgreSQL Quick Start

PostgreSQL Installation

Install PostgreSQL Packages

- System -> YaST2 -> Software Management
- Change Filter to “Search” and search by “postgresql”
- Select packages below and click “Accept” to install

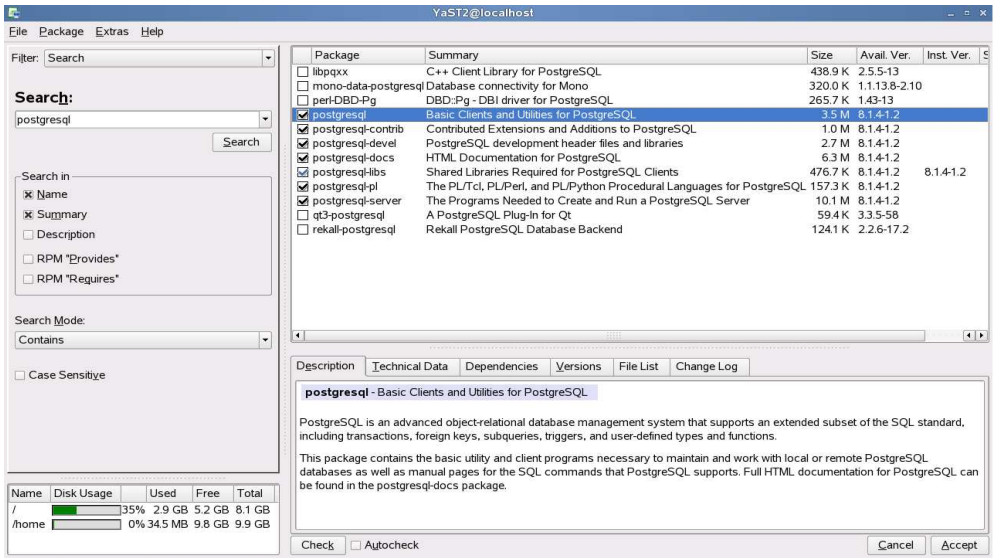


Figure 6 - PostgreSQL Packages

PostgreSQL Technical Documents

Extract PostgreSQL document zip file into local environment and bookmark the URL

- Copy **postgresql-docs-8.2.0.tar.gz** file and extract it into a local folder.
- Open Web browser and add bookmark for index.html of the document

PostgreSQL configuration and administration practice

- Set password for postgres user

```
>su -
#passwd postgres
Enter password: postgres
```

- Check installed PostgreSQL RPM packages using rpm -qa | grep postgresql command

```
>su - postgres
```

```
>rpm -qa | grep postgresql
```

- View PostgreSQL environment variables using env command

```
>env
Note: Check where (which directory) PGDATA is pointed to.
```

- Change directory to PostgreSQL database cluster and view configuration files

```
>cd /var/lib/pgsql/data/
>ls -la
>view postgresql.conf
>view pg_hba.conf
```

- Initialize PostgreSQL database cluster using initdb command

```
>initdb --no-locale –-encoding=UTF-8
```

- Check PostgreSQL server status using pg_ctl status command

```
>pg_ctl status
```

- Start PostgreSQL server using pg_ctl start command

```
>pg_ctl start
>ps x
>psql –version

Note: PostgreSQL service will be able to register as auto start with OS startup:
>su -
#chkconfig postgresql on
#service postgresql start
```

- Stop PostgreSQL server using pg_ctl stop command

```
#su - postgres
>pg_ctl stop
```

Practice psql

- Connect PostgreSQL server from psql

```
>su – postgres
#psql
```

- Practice psql commands

```
postgres=#\?    'Display Help
postgres=#\l    'Display list of databases
postgres=#SELECT version();  'Display PostgreSQL version number
postgres=#\q    'Quit psql
```

- Create User and Database using createuser and createdb commands

```
User: aict, Databse: s-ct-d
>createuser aict
,,, Superuser (y/n) n
,,, Create databases (y/n) y
,,, Create more new roles (y/n) y
>createdb s-ct-d –O s-ct-d
>psql –l          'Display list of databases
```

- Create a table, insert a record, update the record and delete the record using SQL

```
>psql –U aict s-ct-d
s-ct-d=#CREATE TABLE book (id INTEGER PRIMARY KEY, name text);
s-ct-d=# \d
s-ct-d=# \d book
s-ct-d=# INSERT INTO book VALUES (1, 'PostgreSQL Basics');
s-ct-d=# INSERT INTO book VALUES (2, 'MySQL Basics');
s-ct-d=# SELECT * FROM book;
s-ct-d=# DELETE FROM book;
s-ct-d=# \q
```

Install and practice pgAdmin III and Query Tool

- Installation from RPM

Double click pgAdmin III RPM file and click "**Install**" button:

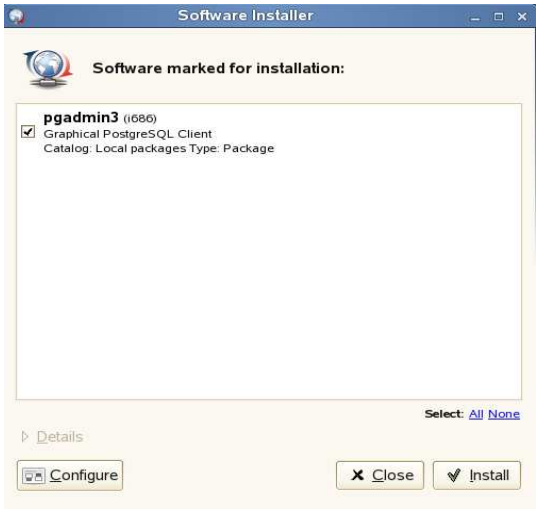


Figure 7 – Install pgAdmin III

- Start pgAdmin III from Run Application

Start “Run Application” from “**Applications**” and click “**Run**” button:

Figure 8 - Start pgAdmin III

- Configure New Server Registration

Enter Address, Description, Username and password. Click OK button to confirm:

Figure 9 - New Server Registration

- Browse to view User, Database and Database Objects from pgAdmin III

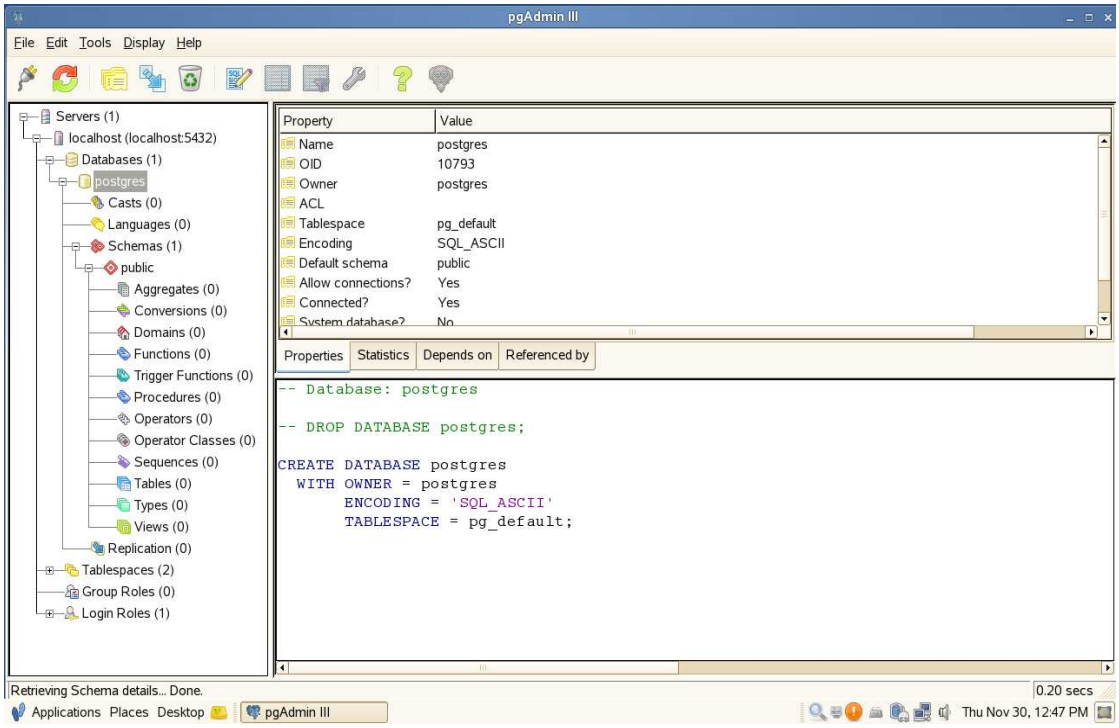


Figure 10 - Browse Database Object with pgAdmin III

- Try Query Tool to access to PostgreSQL database

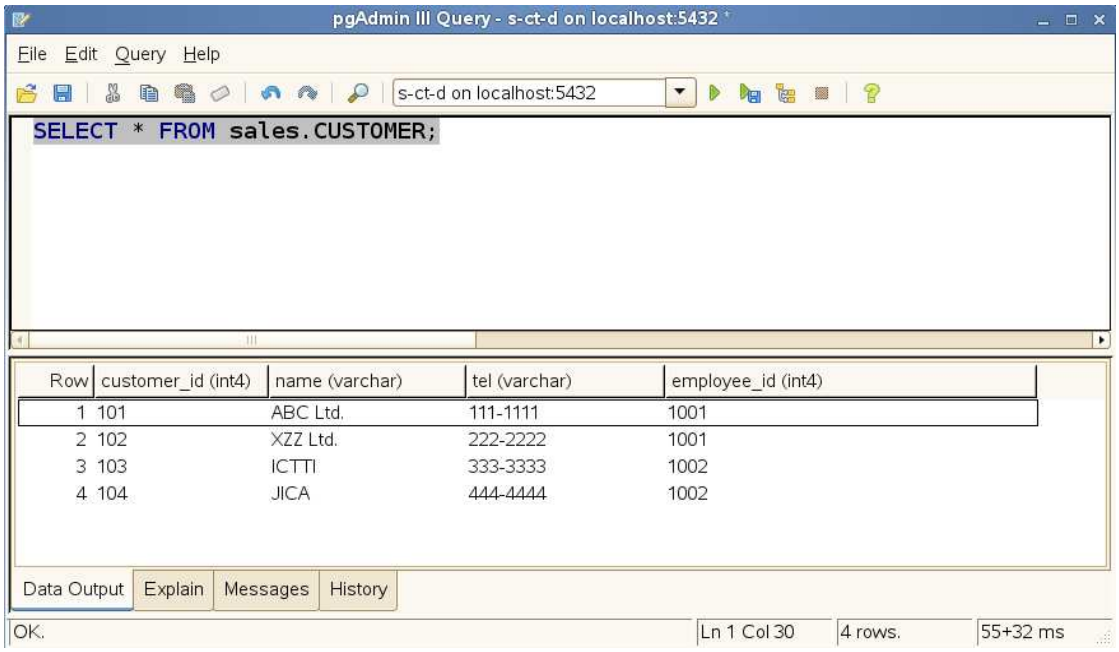


Figure 11 - pgAdmin III Query Tool

3 Introduction to MySQL

3.1 Features

MySQL (http://www.mysql.com/) (pronounced "my ess cue el") is one of the most popular open source relational database management systems (RDBMS) developed and maintained by MySQL AB with more than 10 million installations according to their Web site. MySQL is generally famous for its accessing speed to retrieve data, reliability, and flexibility under the condition of best configuration and management and without executing heavy transactions. The current stable version available today is 5.1. (As of May 2008) which provide following features:

Table 11 - Brief History and features of MySQL

Year	Version	Features
2001 -	3.23	MyISAM, InnoDB, Replication support
2003 -	4.0	Query-cache, Full-text Index support
2004 -	4.1	Sub-query, utf8, ucs2, NDB Clustering support
2005 -	5.0	View, Stored Procedure, Stored Function, Trigger, Information Schema support
2008-	5.1	View, Stored Procedure, Stored Function, Trigger, Information Schema support, Partitioning, Row-based replication, Plug-in API, Event Scheduler, Server log tables, Upgrade program, MySQL Cluster

Features:

- Support multi platform; it runs on all major operating systems, including Linux, UNIX, Mac OS and Windows.
- Support transaction management; full ACID (Atomicity, Consistency, Isolation and Durability) transaction compliant.
- Support foreign keys, joins (Full-outer join will be supported from MySQL 5.1), views, triggers, stored procedures, stored functions and information schema.
- Support Partitioning, Row-based replication, Plug-in API, Event Scheduler, Server log tables, Upgrade program, MySQL Cluster.
- Support most of SQL99 and SQL2003 standards. It also supports storage of binary large objects. Native application programming interfaces for C/C++, Java, .Net, Perl, Ruby, PHP, ODBC and some others.
- Support technologies for an enterprise class database using such as Multi-Version Concurrency Control (MVCC) with InnoDB storage engine.

- Support technologies for high availability and high scalability features, a query planner/optimizer, and write ahead logging for fault tolerance.
- Support pluggable storage engine architecture to choose several table types such as MEMORY, MyISAM, InnoDB, NDB, FEDERATED and some others.

3.2 What's New in MySQL 5.1

The following features have been added to MySQL 5.1.

- **Partitioning.** This capability enables distributing portions of individual tables across a file system, according to rules which can be set when the table is created. In effect, different portions of a table are stored as separate tables in different locations, but from the user point of view, the partitioned table is still a single table.
- **Row-based replication.** Replication capabilities in MySQL originally were based on propagation of SQL statements from master to slave. This is called *statement-based replication*. MySQL 5.1 supports *row-based replication*. Instead of sending SQL statements to the slave, the master writes events to its binary log that indicate how individual table rows are affected.
- **Plugin API.** MySQL 5.1 adds support for a very flexible plugin API that enables loading and unloading of various components at runtime, without restarting the server. Although the work on this is not finished yet, *plugin full-text parsers* are a first step in this direction. This allows users to implement their own input filter on the indexed text, enabling full-text search capability on arbitrary data such as PDF files or other document formats. A pre-parser full-text plugin performs the actual parsing and extraction of the text and hands it over to the built-in MySQL full-text search.
- **Event scheduler.** MySQL Events are tasks that run according to a schedule. When you create an event, you are creating a named database object containing one or more SQL statements to be executed at one or more regular intervals, beginning and ending at a specific date and time.
- **Server log tables.** Before MySQL 5.1, the server writes general query log and slow query log entries to log files. As of MySQL 5.1, the server's logging capabilities for these logs are more flexible. Log entries can be written to log files (as before) or to the general_log and slow_log tables in the mysql database. If logging is enabled, either or both destinations can be selected. The --log-output option controls the destination or destinations of log output.
- **Upgrade program.** The **mysql_upgrade** program checks all existing tables for incompatibilities with the current version of MySQL Server and repairs them if necessary. This program should be run for each MySQL upgrade.

- **MySQL Cluster replication.** Replication between MySQL Clusters is now supported. It is now also possible to replicate between a MySQL Cluster and a non-cluster database.
- **Backup of tablespaces.** The **mysqldump** utility now supports an option for dumping tablespaces. Use -Y or --all-tablespaces to enable this functionality.
- **Improvements to INFORMATION_SCHEMA.** MySQL 5.1 provides much more information in its metadata database than was available in MySQL 5.0. New tables in the INFORMATION_SCHEMA database include FILES, EVENTS, PARTITIONS, PROCESSLIST, ENGINES, and PLUGINS.
- **XML functions with XPath support.** ExtractValue() returns the content of a fragment of XML matching a given XPath expression. UpdateXML() replaces the element selected from a fragment of XML by an XPath expression supplied by the user with a second XML fragment (also user-supplied), and returns the modified XML.
- **Load emulator.** The **mysqlslap** program is designed to emulate client load for a MySQL server and report the timing of each stage. It works as if multiple clients were accessing the server.

3.3 Architecture Overview

MySQL is designed based on the client/server type architecture such as front-end (client) and back-end (server) as illustrated below:

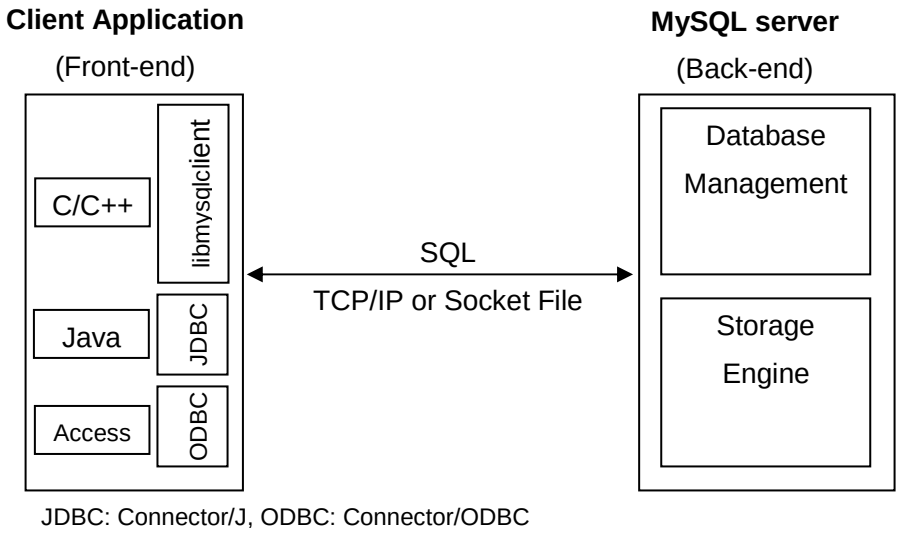


Figure 12 - MySQL Architecture Overview

MySQL server closer look as logical architecture view as shown in below. As can be seen it consists of two levels such as database management and storage management:

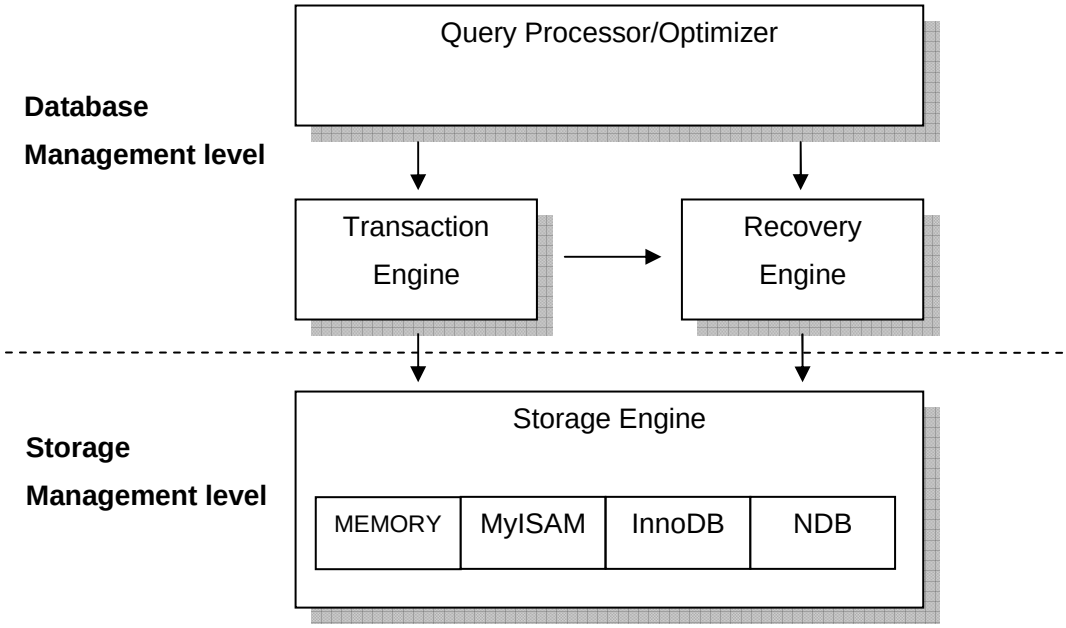


Figure 13 - MySQL Logical Architecture overview

As mentioned earlier, one of the most interesting features of MySQL is their pluggable storage engine architecture. Table below shows some of the popular table types available for the storage engine:

Table 12 - MySQL Table Type

Table Type	Description
MEMORY	Use memory (RAM) as storage in order to implement faster access database however the data will be lost when MySQL server shuts down.
MyISAM	No-transaction is supported and known as faster access table type for mainly reference purpose (Non-transactional) databases.
InnoDB	Transaction supported table type. Default table type for MySQL 5.1.

Table 13 - MySQL installation directory by non RPM package

Directory	Description
bin/	Binary commands
data/	Data directory (datadir)
include/	Header files
lib/	Library files

scripts/	Script files
share/	Fill_help_tables.sql, mysql_fix_privilege_table_sql
share/mysql/	Error message files
support-files/	Mysql.server, my.cnf
docs/	Documents
man/	Online Manual
sql-bench/	Benchmark test
mysql-test/	Mysql-test script
tests/	Test scripts

3.4 MySQL CUI Commands

Followings are commonly used MySQL commands and scripts:

- mysql.server

mysql.server is a script to start and stop MySQL server (mysqld).

```
-- Start MySQL server
#/usr/local/mysql/support-files/mysql.server start
-- Stop MySQL server
#/usr/local/mysql/support-files/mysql.server stop
```

- mysqladmin

mysqladmin is a command line tool to manage and administrate MySQL server.

```
linux> mysqladmin [Option] Command
```

Table 14 - mysqladmin options

Option	Description
--user=*User name* -u *User name*	Specify MySQL user name
--password [=*Password*] -p[*Password*]	Specify password for the user. For the security reason, it is recommended to enter password separately.
--host=*Hostname* -h *Hostname*	Specify MySQL Server hostname or IP address
--port=*Port Number* -P *Port Number*	Specify MySQL Server port number. Default port number would be 3306

Table 15 - mysqladmin commands

Command	Description
create [Database]	Create a new database
drop [Database]	Drop existing database
Password [New password]	Change (or set) password for MySQL user
ping	Check response (Alive or dead process) from MySQL server
Version	Check MySQL server version
Status	Check MySQL server status
Variables	Check MySQL server variables
Shutdown	Shutdown MySQL server

Example:

```
-- Set password for root user as 'root'
linux> mysqladmin –u root password 'root'
-- Check MySQL server process either alive or dead
linux> mysqladmin –u root –p ping
-- Create a new database and drop the database
linux> mysqladmin –u root –p create mydb
linux> mysqladmin –u root –p drop mydb
-- Shutdown MySQL server
linux> mysqladmin –u root –p shutdown
linux> mysqladmin –u root –p ping
```

- mysql

Mysql is one of the most important tools that allow connecting to MySQL server.

```
linux> mysql [Option] [Database]
```

Table 16 - mysql options

Option	Description
--user=*User name* -u *User name*	Specify MySQL user name
--password [=*Password*] -p[*Password*]	Specify password for the user. For the security reason, it is recommended to enter password separately.
--host=*Hostname* -h *Hostname*	Specify MySQL Server hostname or IP address

--port=*Port Number* -P *Port Number*	Specify MySQL Server port number. Default port number would be 3306

Example:

```
-- Connect to MySQL
linux> mysql –u root –p

-- mysql Help commands
mysql> help
mysql> help [SQL statement]
-- View status of MySQL server
mysql> STATUS;
-- Connect (select) to database and list existing tables
mysql> SHOW DATABASES;
mysql> USE [Database];
-- Execute DDLs and show databases and tables
mysql> CREATE DATABASE ,,,;
mysql> SHOW DATABASES;
mysql> SHOW TABLES;
mysql> CREATE TABLE ,,,,;
mysql> SHOW TABLES;
mysql> DROP TABLE ,,,,;
mysql> DROP DATABASE ,,,;
mysql> SHOW TABLES;
mysql> SHOW DATABASES;
-- Quit mysql
mysql> exit
```

Exercise 3. - MySQL Quick Start

MySQL Installation (By Linux non RPM package)

- Uninstall MySQL RPM packages

1. System -> YaST2 -> Software Management
2. Change Filter to “Search” and search by “mysql”
3. Uninstall mysql RPM packages in case it has been installed

- Download MySQL by non RPM package

Download MySQL Community Server 5.1 Linux non RPM packages from
http://dev.mysql.com/downloads/mysql/5.1.html#downloads.
Or
http://www.ictti.site/share/linux/MySQL5.1/mysql-5.1.25-rc-linux-i686-glibc23.tar.gz

- Add User and Group for MySQL with Linux “root” user

```
localhost:/ # cd /usr/local
localhost:/usr/local #
localhost:/usr/local # groupadd mysql
localhost:/usr/local # useradd -g mysql -d /usr/local/mysql mysql
```

- Extract downloaded package

```
--Copy package file to /usr/local/
localhost:/usr/local # cp /home/ono/mysql-5.1.25-rc-linux-i686-glibc23.tar.gz .

--Extract the package file
localhost:/usr/local # gzip –dc mysql-5.1.25-rc-linux-i686-glibc23.tar.gz | tar xvf -
```

- Create Symbolic link as mysql

```
localhost:/usr/local # ln -sf mysql-5.1.25-linux-i686 mysql
```

- Initialize MySQL database

```
localhost:/usr/local # cd mysql
localhost:/usr/local/mysql # ./scripts/mysql_install_db --user=mysql
```

- Change owner of MySQL database directory (datadir)

```
localhost:/usr/local/mysql # chown -R mysql data
localhost:/usr/local/mysql # chgrp -R mysql data
```

```
localhost:/usr/local/mysql # ls -la
```

- Start MySQL server

```
localhost:/usr/local/mysql # ./bin/mysqld_safe --user=mysql &
```

- Check MySQL server status

```
-- By using Linux command
localhost:/usr/local/mysql # ps -e | grep mysql

-- By mysqladmin command
localhost:/usr/local/mysql # ./bin/mysqladmin ping
```

- Set password for root

```
localhost:/usr/local/mysql # ./bin/mysqladmin -u root password 'root'
```

- How to login to **mysql** and exit from **mysql** command

```
-- Open another terminal
tda@localhost:~> /usr/local/mysql/bin/mysql -u root –p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.25-rc-community MySQL Community Server (GPL)
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>exit;
```

- Stop MySQL server

```
localhost:/usr/local/mysql # /usr/local/mysql/bin/mysqladmin -u root -p shutdown
Enter password:
STOPPING server from pid file /usr/local/mysql/data/localhost.pid
070428 16:39:21 mysqld ended
```

- Add mysql path

```
tda@localhost:~> pwd
/home/tda
tda@localhost:~> vim ~/.profile
-- Add following PATH:
PATH=$PATH: $HOME/bin: /usr/local/mysql/bin
tda@localhost:~> source ~/.profile
```

- Auto MySQL start up with init.d

```
# cp /usr/local/mysql/support-files/mysql.server /etc/init.d/mysql
# /etc/init.d/mysql start
Starting MySQL                                    done
# /etc/init.d/mysql stop
Shutting down MySQL..                             done

localhost:/usr/local/mysql # chkconfig --add mysql

-- Restart Linux and check mysqld is running or not
# ps –e | grep mysql
```

- How to use mysql

```
linux> mysql –u root -p

mysql>CREATE DATABASE aict;
mysql>SHOW DATABASES;
mysql>USE aict;
mysql>SHOW TABLES;
mysql>CREATE TABLE book (id INTEGER PRIMARY KEY, name TEXT);
mysql>SHOW CREATE TABLE book;
mysql>SHOW TABLES;
mysql>SHOW COLUMNS FROM book;
mysql>INSERT INTO book VALUES (1, 'Ming Ga La Ba PostgreSQL');
mysql>INSERT INTO book VALUES (2, 'Ming Ga La Ba MySQL');
mysql>SELECT * FROM book;
mysql>DELETE FROM book;
mysql>DROP TABLE book;
mysql>SHOW TABLES;
mysql>SHOW STATUS;
mysql>SHOW VARIABLES;
```

Table 17 - MySQL server Default Variable Setting

Variable	Value	Description
basedir	/usr/local/mysql/	Base directory
datadir	/usr/local/mysql/data/	Database directory
my.cnf	/etc/my.cnf	MySQL configuration file
User	mysql	MySQL user
default-character-set	latin1	Default character set
Port Number	3306	TCP Port Number

Socket File	/tmp/mysql.sock	UNIX Socket

MySQL GUI Tools Installation (By RPM packages)

- Download MySQL GUI Tools RPM packages from http://dev.mysql.com/downloads/gui-tools/5.0.html
- Install MySQL GUI Tools RPM packages for SUSE11.1

```
localhost:/# rpm -ivh mysql-gui-tools-5.0r12-195.1-i586.rpm
localhost:/# rpm -ivh mysql-administrator-5.0r12-195.1-i586.rpm
localhost:/# rpm –ivh mysql-migration-toolkit-5.0r12-195.1-i586.rpm
localhost:/# rpm -ivh mysql-workbench- 5.0r12-195.1-i586.rpm
localhost:/# rpm -ivh mysql-query-browser-5.0r12-195.1-i586.rpm
```

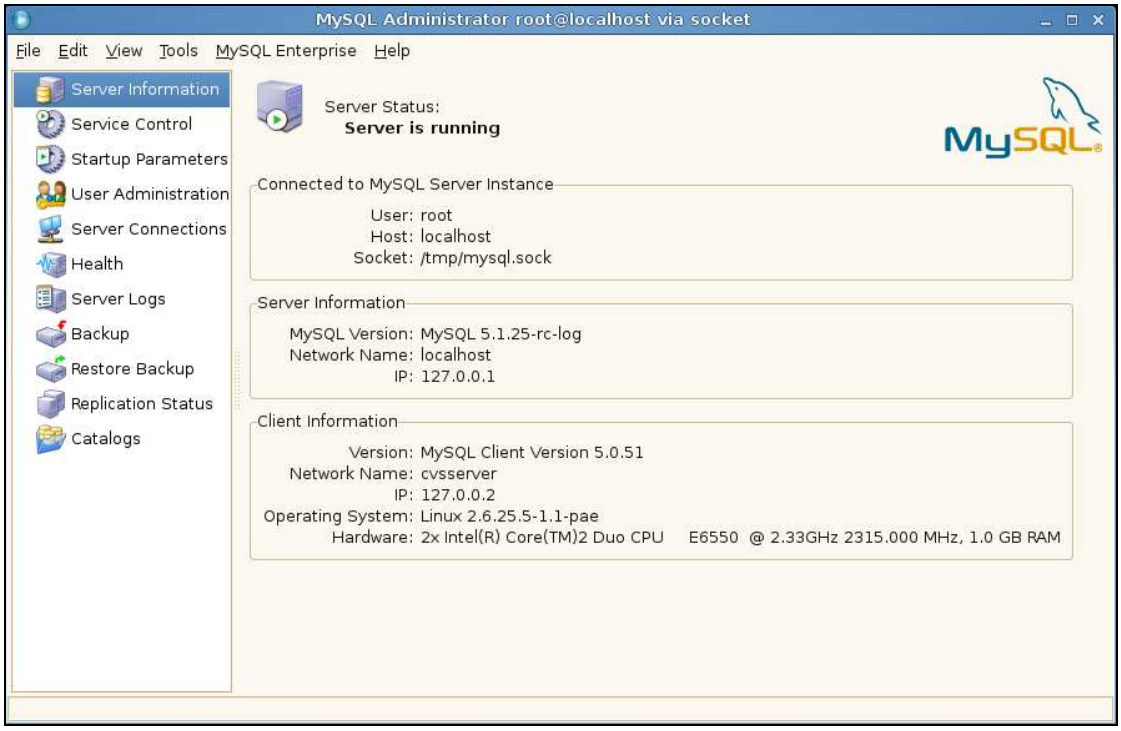


Figure 14 - MySQL Administrator

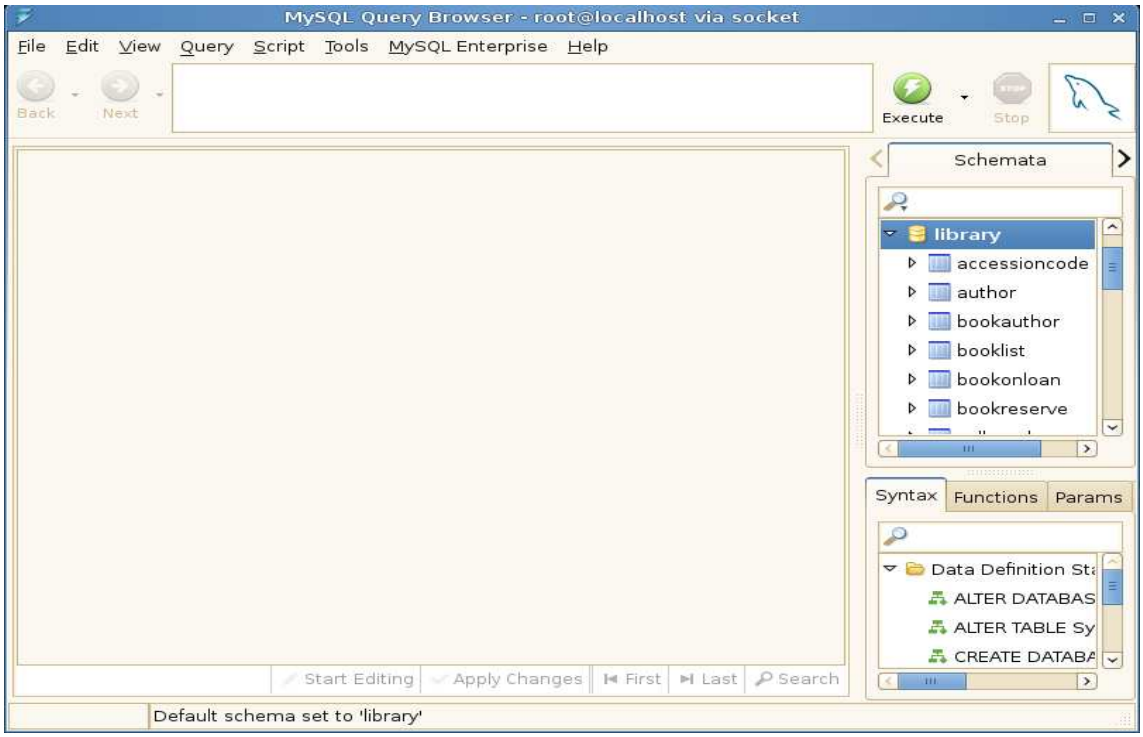


Figure 15 - MySQL Query Browser

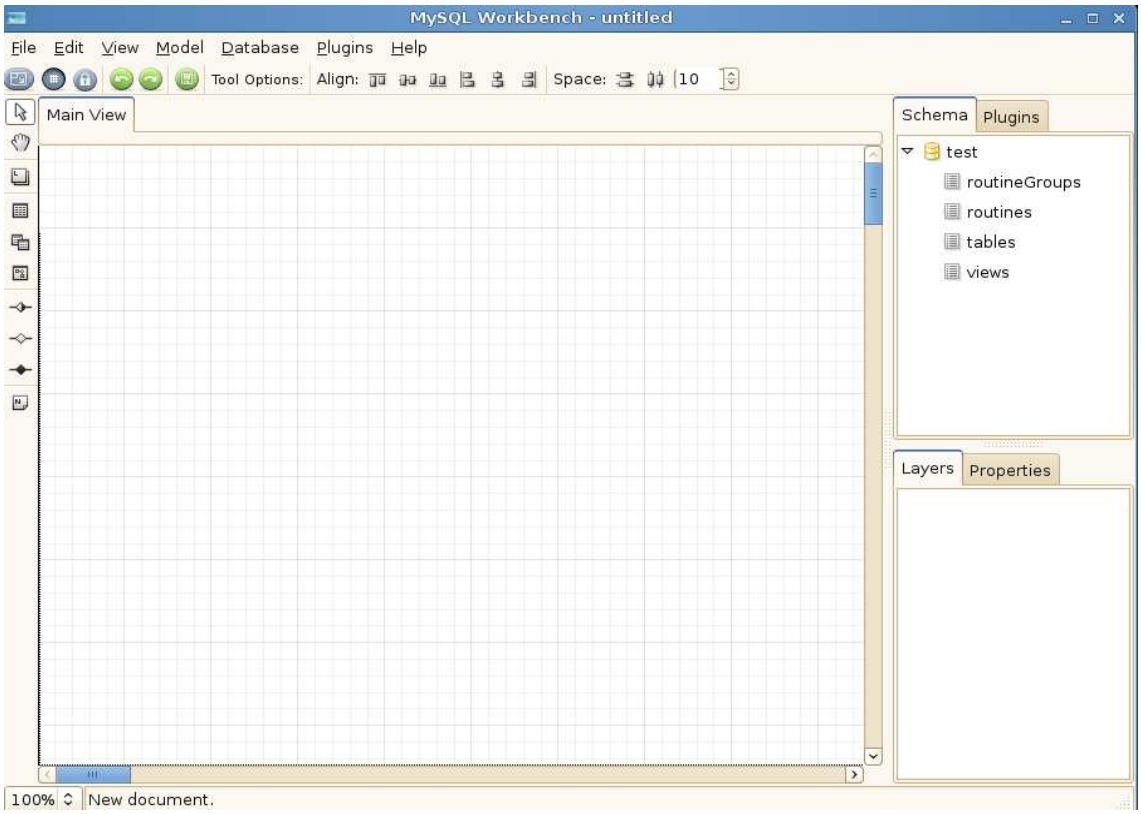
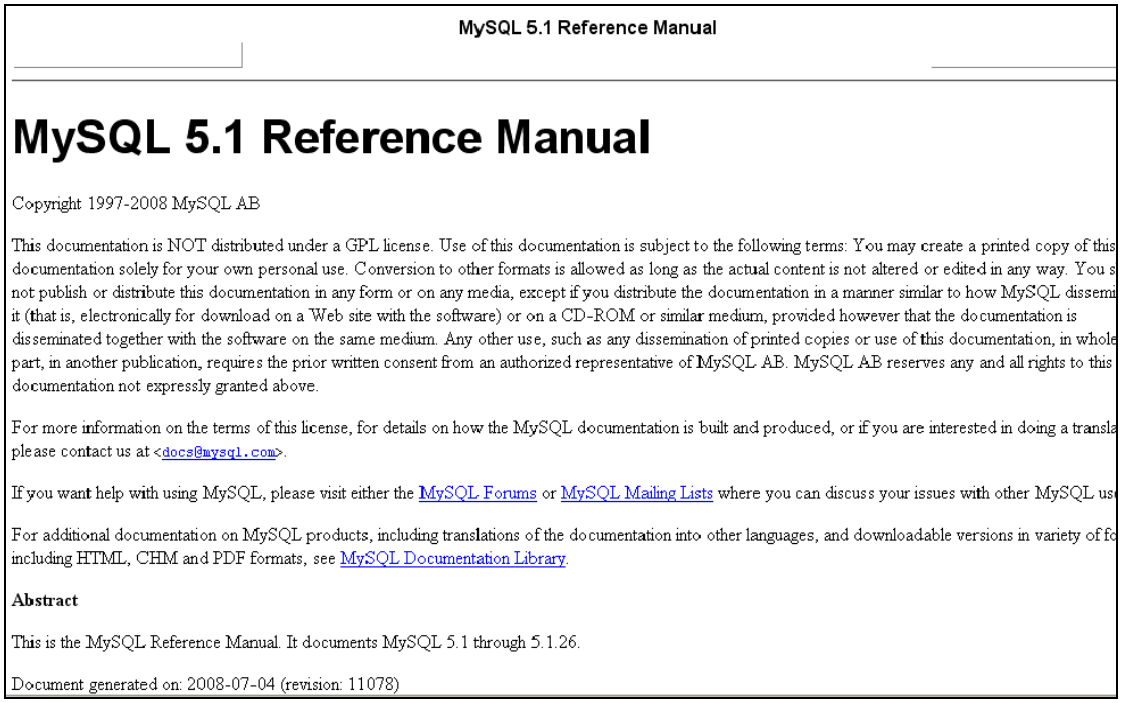


Figure 16 - MySQL Workbench

MySQL Reference Manual installation

1. Download MySQL Reference Manual from http://dev.mysql.com/doc/ English v5.1
2. Extract archive file into local PC folder like /home/tda/refman-5.0-en.html-chapter
3. Add bookmark for index.html with the Web browser
4. Please go through this Reference Manual whenever you have time. This is important document for you to understand MySQL from overview to details, highly recommend.


MySQL 5.1 Reference Manual

MySQL 5.1 Reference Manual

Copyright 1997-2008 MySQL AB

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms: You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You s not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how MySQL dissemi it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole part, in another publication, requires the prior written consent from an authorized representative of MySQL AB. MySQL AB reserves any and all rights to this documentation not expressly granted above.

For more information on the terms of this license, for details on how the MySQL documentation is built and produced, or if you are interested in doing a transla please contact us at <docs@mysql.com>.

If you want help with using MySQL, please visit either the MySQL Forums or MySQL Mailing Lists where you can discuss your issues with other MySQL us

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of fo including HTML, CHM and PDF formats, see MySQL Documentation Library.

Abstract

This is the MySQL Reference Manual. It documents MySQL 5.1 through 5.1.26.

Document generated on: 2008-07-04 (revision: 11078)

Figure 17 - MySQL Reference Manual

MySQL Quick Tour

Now it is time to go around a quick tour of MySQL. We will learn the syntax and details of each SQL in this tour later in Chapter 6. “Introduction to SQL”. So don’t worry about at this moment but just to enjoy what MySQL can do, here we go!

- Start mysql

```
# Open a Linux Terminal and Start mysql
>mysql –u root -p
```

- Create a database and use it

```
mysql>show databases;
mysql>create database s-ct-d;
Query OK, 1 row affected (0.06 sec)
```

```
mysql>show databases;
+--------------------+
| Database           |
+--------------------+
| mysql              |
| s-ct-d             |
| test               |
+--------------------+
5 rows in set (0.00 sec)

mysql>use s-ct-d;
Database changed
```

- Create 4 tables (DEPT, EMP, BONUS and SALGRADE)

```
# Copy and paste following CREATE TABLE statements into mysql
CREATE TABLE DEPT
      (
       DEPTNO NUMERIC(2) PRIMARY KEY,
       DNAME VARCHAR(14) ,
       LOC VARCHAR(13)
       );
CREATE TABLE EMP
      (
       EMPNO NUMERIC(4) PRIMARY KEY,
       ENAME VARCHAR(10),
       JOB VARCHAR(9),
       MGR NUMERIC(4),
       HIREDATE DATE,
       SAL NUMERIC(7,2),
       COMM NUMERIC(7,2),
       DEPTNO NUMERIC(2),
       FOREIGN KEY(DEPTNO) REFERENCES DEPT(DEPTNO)
       );
CREATE TABLE BONUS
       (
       ENAME VARCHAR(10),
       JOB VARCHAR(9),
       SAL NUMERIC,
       COMM NUMERIC
       );
CREATE TABLE SALGRADE
      (
```

```
    GRADE NUMERIC,
    LOSAL NUMERIC,
    HISAL NUMERIC
    );
```

- Check list of tables and structure of each table

```
mysql> show tables;
+-----------------+
| Tables_in_s_ct_d |
+-----------------+
| bonus           |
| dept            |
| emp             |
| salgrade        |
+-----------------+
5 rows in set (0.00 sec)

mysql> desc DEPT;
mysql> desc EMP;
mysql> desc BONUS;
mysql> desc SALGRADE;
```

- Insert sample data into 4 tables

```
# Copy and paste following INSERT statements into mysql
INSERT INTO DEPT VALUES (10,'ACCOUNTING','NEW YORK');
INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS');
INSERT INTO DEPT VALUES (30,'SALES','CHICAGO');
INSERT INTO DEPT VALUES (40,'OPERATIONS','BOSTON');

INSERT INTO EMP VALUES
(7369,'SMITH','CLERK',7902, str_to_date('17-12-1980','%d-%m-%Y'),800,NULL,20);
INSERT INTO EMP VALUES
(7499,'ALLEN','SALESMAN',7698,str_to_date('20-2-1981','%d-%m-%Y'),1600,300,3
0);
INSERT INTO EMP VALUES
(7521,'WARD','SALESMAN',7698,str_to_date('22-2-1981','%d-%m-%Y'),1250,500,3
0);
INSERT INTO EMP VALUES
(7566,'JONES','MANAGER',7839,str_to_date('2-4-1981','%d-%m-%Y'),2975,NULL,2
0);
INSERT INTO EMP VALUES
(7654,'MARTIN','SALESMAN',7698,str_to_date('28-9-1981','%d-%m-%Y'),1250,140
```

```
0,30);
INSERT INTO EMP VALUES
(7698,'BLAKE','MANAGER',7839,str_to_date('1-5-1981','%d-%m-%Y'),2850,NULL,3
0);
INSERT INTO EMP VALUES
(7782,'CLARK','MANAGER',7839,str_to_date('9-6-1981','%d-%m-%Y'),2450,NULL,1
0);
INSERT INTO EMP VALUES
(7788,'SCOTT','ANALYST',7566,str_to_date('13-7-1987','%d-%m-%Y'),3000,NULL,2
0);
INSERT INTO EMP VALUES
(7839,'KING','PRESIDENT',NULL,str_to_date('17-11-1981','%d-%m-%Y'),5000,NUL
L,10);
INSERT INTO EMP VALUES
(7844,'TURNER','SALESMAN',7698,str_to_date('8-9-1981','%d-%m-%Y'),1500,0,30)
;
INSERT INTO EMP VALUES
(7876,'ADAMS','CLERK',7788,str_to_date('13-7-1987',
'%d-%m-%Y'),1100,NULL,20);
INSERT INTO EMP VALUES
(7900,'JAMES','CLERK',7698,str_to_date('3-12-1981','%d-%m-%Y'),950,NULL,30);
INSERT INTO EMP VALUES
(7902,'FORD','ANALYST',7566,str_to_date('3-12-1981','%d-%m-%Y'),3000,NULL,20
);
INSERT INTO EMP VALUES
(7934,'MILLER','CLERK',7782,str_to_date('23-1-1982','%d-%m-%Y'),1300,NULL,10)
;

INSERT INTO SALGRADE VALUES (1,700,1200);
INSERT INTO SALGRADE VALUES (2,1201,1400);
INSERT INTO SALGRADE VALUES (3,1401,2000);
INSERT INTO SALGRADE VALUES (4,2001,3000);
INSERT INTO SALGRADE VALUES (5,3001,9999);
```

- SELECT statement basics

```
# SELECT
mysql> SELECT EMPNO, ENAME FROM EMP;
mysql> SELECT * FROM EMP;
mysql> SELECT * FROM DEPT;
# See what is going on

# Order by
mysql> SELECT * FROM EMP ORDER BY ENAME ASC;
```

```
mysql> SELECT * FROM EMP ORDER BY ENAME DESC;
# See what is going on

# Distinct rows
mysql> SELECT DEPTNO FROM EMP;
mysql> SELECT DISTINCT DEPTNO FROM EMP;
# See what is going on

# Calculation
SELECT SAL, COMM, SAL + COMM FROM EMP;
SELECT SAL, SAL * 1.5 FROM EMP;
# See what is going on

# Where condition
SELECT * FROM EMP WHERE EMPNO=7788;
SELECT * FROM EMP WHERE DEPTNO=10;
SELECT * FROM EMP WHERE SAL >= 2000;
SELECT * FROM EMP WHERE HIREDATE >= '82-01-01';
# See what is going on

# Join tables (DEPT and EMP)
SELECT DEPT.DEPTNO, DEPT.DNAME, DEPT.LOC, EMP.EMPNO, EMP.ENAME,
EMP.JOB
FROM DEPT, EMP WHERE DEPT.DEPTNO = EMP.DEPTNO ORDER BY DEPTNO;
# See what is going on
```

- INSERT, UPDATE and DELETE statement basics

```
# INSERT
mysql> INSERT INTO EMP
        VALUES(
        9999,
        'TDA',
        'CLERK',
        7788,
        '2006-06-30',
        1000,
        NULL,
        30);

mysql> SELECT * FROM EMP;
# See what is going on
```

```
# UPDATE
mysql> UPDATE EMP SET SAL=800, COMM=500 WHERE ENAME=' TDA';

mysql> SELECT * FROM EMP;
# See what is going on

# DELETE
mysql> DELETE FROM EMP WHERE ENAME=' TDA';
mysql> SELECT * FROM EMP;
# See what is going on

# That's all for today. How do you find MySQL? As mentioned earlier, we will discuss details in Chapter 6.
```

4 Introduction to Database Design

4.1 Data Models

4.1.1 Types of Data Models

As you may have studied about several logical data models during the study time in universities such as **Hierarchical Model** or **Network Model** described in table below, however the focus of this session is scope to **Relational Model** and **Entity Relationship Model**, which are most commonly used in actual software development projects today.

Table 18 - Data Models

Data Models	Description
Hierarchical Model	The hierarchical data model was developed in 1960s to organize data in a tree structure. There is a hierarchy of parent and child data segments. This structure implies that a record can have repeating information, generally in the child data segments.
Network Model	The network model is conceived as a flexible way of representing objects and their relationships. It was developed into a standard specification published in 1969 by the Conference on Data Systems Languages (CODASYL) Consortium. Where the hierarchical model structures data as a tree of records, with each record having one parent record and many children, the network model allows each record to have multiple parent and child records.
Relational Model	The relational model was developed by E.F. Codd in 1970. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organized in tables. A table is a collection of records and each record in a table contains the same fields.
Entity-Relationship Model	Entity-Relationship model was introduced by P.P.Chen in 1976 and is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram which is used to visually represent data objects.

4.1.2 Degrees of Data Abstraction by ANSI/SPARC

A framework for data modeling based on degrees of data abstraction by American National Standards Institute (ANSI) Standard Planning and Requirements Committee (SPARC) was defined in 1978 and it is a sort of de-fact standard guideline of data modeling today. The ANSI/SPARC architecture provides three levels of data abstraction such as **external**, **conceptual** and **internal** as depict in the figure below:

- External model

The external model is the end-users' view of the data environment. The user means those who use application programs to access to create data and manipulate the information.

- Conceptual model

The conceptual model is global or organizational view as high-level description of the external model. It is independent from both hardware and software however it depends on chosen data model with DBMS type.

- Internal model

The internal model is the representation of the database as seen by DBMS. The model creates by database designer to map with conceptual model using by the chosen database.

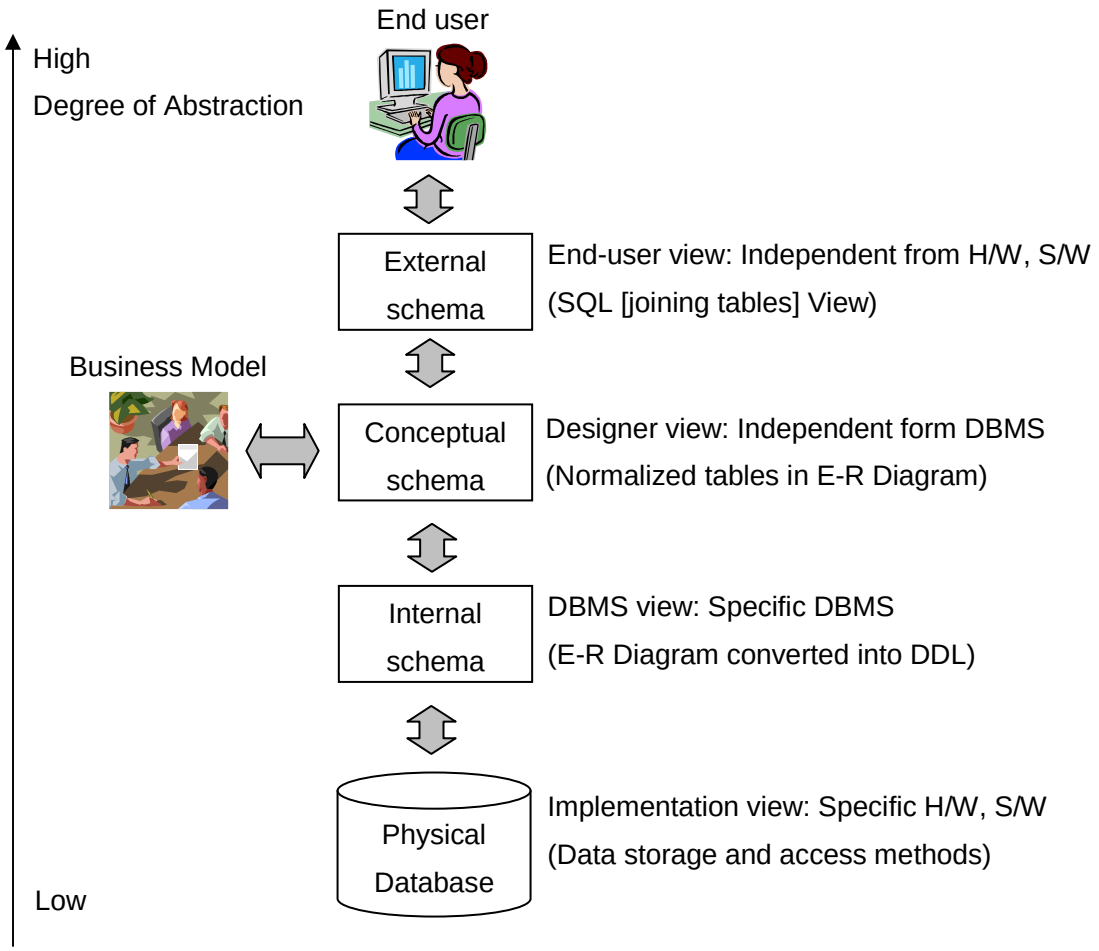


Figure 18 - ANSI/SPARC degrees of data abstraction.

4.2 Relational Database Model

4.2.1 Relational Database Model Building Blocks

Basic building blocks for relational database model are such as entities, attributes, relationships and constraints as explained in the table below:

Table 19 - Data Model Building Blocks

Building Blocks	Description
Entities	Entities are the principal data object about which information is to be collected.
Attributes	Attributes describe the entity of which they are associated. A particular instance of an attribute is a value.

Relationships	Relationships describe an association among entities. There are three types such as: ● One-to-many (1:M) relationship This is when at most one instance of a entity A is associated with one instance of entity B. ● Many-to-many (M:N) relationship This is when for one instance of entity A, there are zero, one, or many instances of entity B, but for one instance of entity B, there is only one instance of entity A. ● One-to-one (1:1) relationship This is when for one instance of entity A, there are zero or one instance of entity B and for one instance of entity B there are zero or one instance of entity A.
Constraints	Constraints are restriction placed on the data. Constraints help to ensure data integrity.

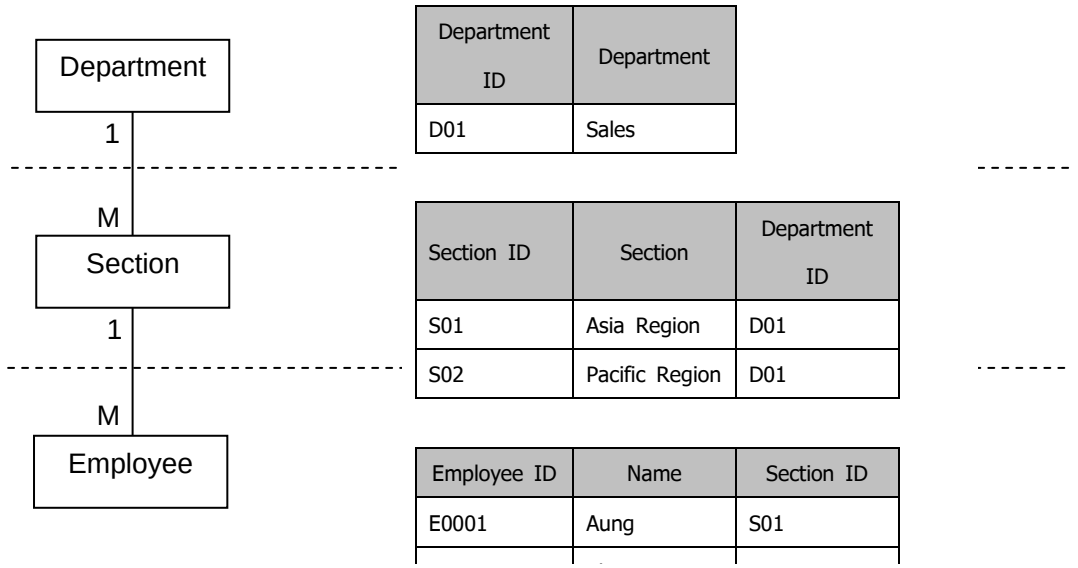
4.2.2 Logical View of Relational Database Model

Relational Data Model is for the relational database model to examine a logical view of data. It was introduced by E.F.Codd in 1970. Table below is, in order to avoid confusion, to compare between Relational model and Entity-Relationship model building blocks:

Table 20 - Database, Relational Model and Entity-Relationship Model

Database	Relational Model	Entity-Relationship Model
Table	Relation	Entity
Column	Attribute	Attribute
Row	Tuple	Occurrence

Two figures below depict the relational database model (left) and actual tables, columns and rows (right):

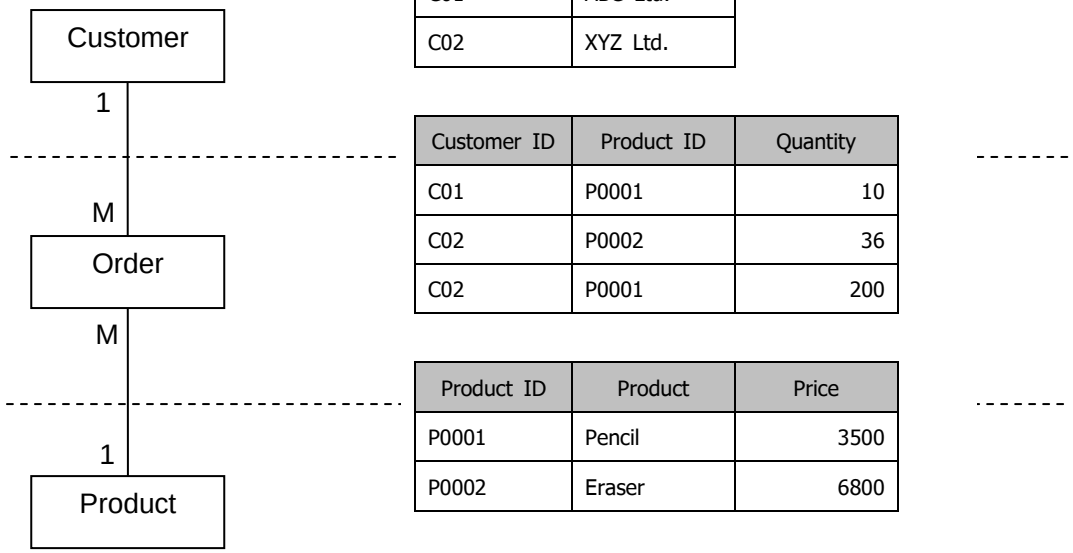


Department ID	Department
D01	Sales

Section ID	Section	Department ID
S01	Asia Region	D01
S02	Pacific Region	D01

Employee ID	Name	Section ID
E0001	Aung	S01

Figure 19 - Sample Relational Model (1/2)



Customer ID	Customer
C01	ABC Ltd.
C02	XYZ Ltd.

Customer ID	Product ID	Quantity
C01	P0001	10
C02	P0002	36
C02	P0001	200

Product ID	Product	Price
P0001	Pencil	3500
P0002	Eraser	6800

Figure 20 - Sample Relational Model (2/2)

4.2.3 Keys

A key which consists of one or more attributes is to determine rows in a table. Table below is a list of keys in relational database model:

Table 21 - Relational Database Keys

Key Type	Definition
Super Key	An attribute or combination of attributes that uniquely identifies each row in a table.
Candidate Key	A minimal super key. A super key without redundant attributes.
Primary Key	A candidate key selected to use mainly. Null value does not allow for the primary key.
Foreign Key	An attribute or combination of attributes in one table whose values must either match the primary key in another table or be null.

4.2.4 Integrity Rules

There are two types of integrity rules defined such as entity integrity and reference integrity as shown in table below. Those integrity rules are normally enforced by DBMS automatically.

Table 22 - Integrity Rules

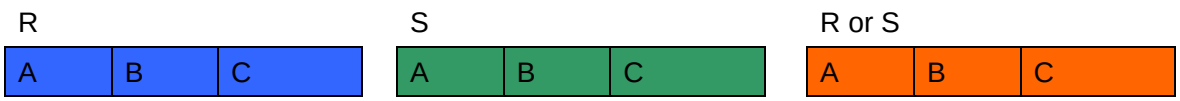
Type of Integrity	Definition
Entity Integrity	All Primary Key entries must be unique and must have entry (Null is not allowed).
Reference Integrity	It is impossible for attribute to have invalid entry.

4.2.5 Relational Algebra

Relational algebra defines the theoretical way of manipulating table contents using following operators, however to be considered minimally relational, normally DBMS must support the key relational operators at least such as SELECT (Selection), PROJECT (Projection) and JOIN (Join):

- Union

Union combines all rows from two tables excluding duplicate rows as shown in Figure below:



1	MM	Myanmar
2	JP	Japan

1	MM	Myanmar
3	SG	Singapore
4	MY	Malaysia

1	MM	Myanmar
2	JP	Japan
3	SG	Singapore
4	MY	Malaysia

R S R or S

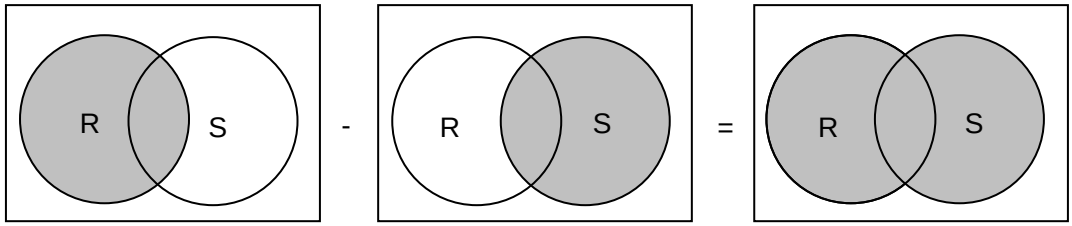


Figure 21 - Union

- Intersection

Intersect yields only the rows that appear in both tables as shown in Figure below:

R S R and S

A	B	C
1	MM	Myanmar
2	JP	Japan

A	B	C
1	MM	Myanmar
3	SG	Singapore
4	MY	Malaysia

A	B	C
1	MM	Myanmar

R S R and S

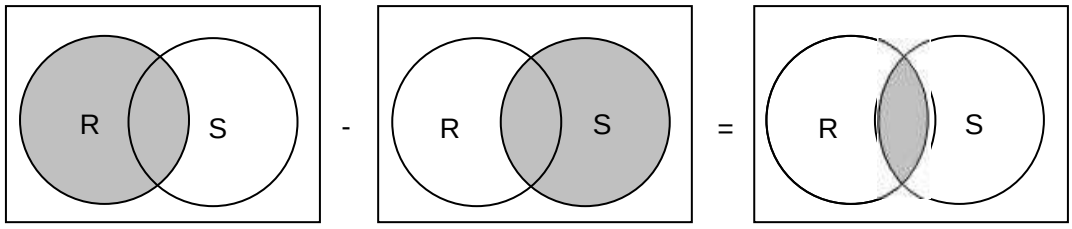


Figure 22 - Intersection

- Difference

Difference yields all rows in one table that are not found in the other table as shown in Figure below:

R S R - S

A	B	C
1	MM	Myanmar
2	JP	Japan

A	B	C
1	MM	Myanmar
3	SG	Singapore
4	MY	Malaysia

A	B	C
2	JP	Japan

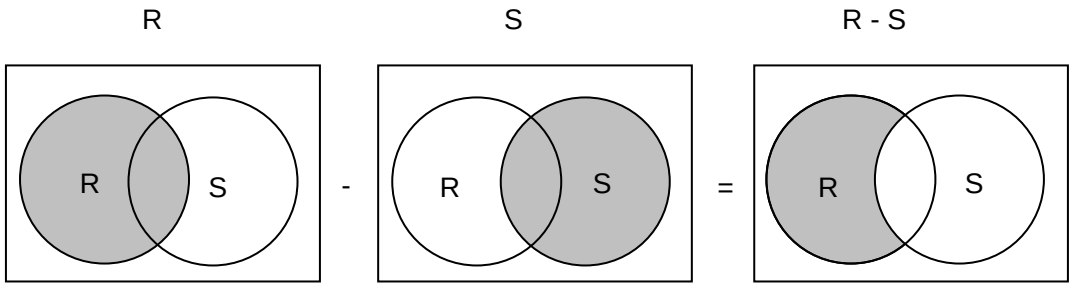


Figure 23 - Difference

- Product

Product yields all possible pairs of rows from two tables as shown in Figure below. This is equivalent with SQL statement of "SELECT * FROM R, S".

R

A	B
1	MM
2	JP

S

C	D
X	xx
Y	yy
Z	zz

R X S

A	B	C	D
1	MM	X	xx
1	MM	Y	yy
1	MM	Z	zz
2	JP	X	xx
2	JP	Y	yy
2	JP	Z	zz

Figure 24 - Product

- Projection

Projection yields all values for selected attributes. It means projection yields a vertical subset of a table as shown in Figure below:

R

A	B	C

R[Attribute B]

B

1	MM	Myanmar	MM
2	JP	Japan	JP

Figure 25 - Projection

- Selection

Selection yields values for all rows found in a table. It can yield only those row values match a specified criterion as shown in Figure below.

R

A	B	C
1	MM	Myanmar
2	JP	Japan

R[Attribute A=1]

A	B	C
1	MM	Myanmar

Figure 26 - Selection

- Join

Join yields information combined from two or more tables. Join links tables by selecting rows with common values in their common attributes as shown in two figures below:

Equi-join

R

A	B
1	MM
2	JP

S

A	C
1	Myanmar
2	Japan
3	Singapore

R[R.A = S.A]

R.A	B	S.A	C
1	MM	1	Myanmar
2	JP	2	Japan

Figure 27 - Equi-join

Natural Join

R

A	B
1	MM
2	JP

S

A	C
1	Myanmar
2	Japan
3	Singapore

Natural Join

A	B	C
1	MM	Myanmar
2	JP	Japan

Figure 28 - Natural Join

- Division

Division is opposite operation of Product as shown in Figure below:

R

A	B	X
1	11	A
2	22	A
1	11	B
2	22	B
1	11	C
2	22	C

S

A	B
1	11
2	22

`

R÷S

X
A
B
C

Figure 29 - Division

4.2.6 Data Dictionary

Data dictionary provides all the details of relational data model such as table, attribute, types of attribute, keys and references. Table below is a sample data dictionary that for database designer to logically design the database.

Table 23 - Data Dictionary

Table	Attribute	Type	Format/ Range	Required	PK/ FK	FK Reference Table
Employee	employee_id	INTEGER		Yes	PK	
	name	VARCHAR(50)		Yes		
	hire_date	DATE	DD/MM/YY	Yes		
	department_id	INTEGER		Yes	FK	Department
Department	department_id	INTEGER		Yes	PK	
	name	VARCHAR(50)		Yes		
	location	VARCHAR(50)				

PK/FK: Primary Key/Foreign Key

4.3 Entity Relationship Modeling

4.3.1 E-R Model

An entity-relationship model (ERM) provides a high-level description of a conceptual data model in a graphical notation for representing such data models in the form of entity-relationship diagrams (ERD). The ERM is independent from database type and used for conceptual to logical relational database design. There are several types of E-R modeling notations are existed such as J.Martin (Crows' Feet), IDEF1X, Chen, BACHMAN and so on and E-R model and its diagrams are described with J.Martin (Crow's Feet) model.

- Entity

An entity represents a discrete object of the interest to the end user and it can be thought of as nouns. There are three types of entities can be categorized for actual database designing stage and those are:

1. Entities for resources (Static entity such as Employee, Customer)
2. Entities for transactional data (Dynamic entity such as Order, Invoice)
3. Entities for summary reports (Aggregated view such as Monthly Sales Report)

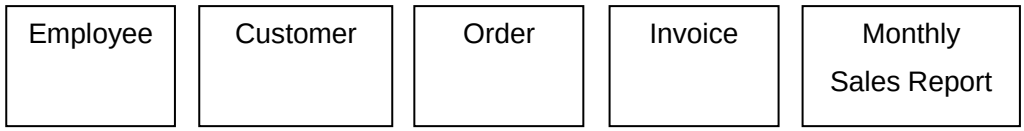


Figure 30 - Entities

- Attribute

Attribute is property or characteristics of entity. There are two types of attributes such as identical and non-identical attributes. Identical attribute is called primary key of the entity and underlined its attribute as shown in below:

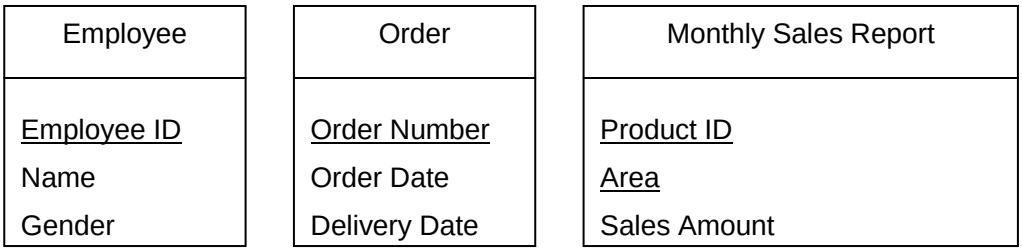


Figure 31 – Entity and Attributes

- Relationship

Relationship describes an association between entities based on business rules among those. There are three types of connectivity are identified as shown in figure below:

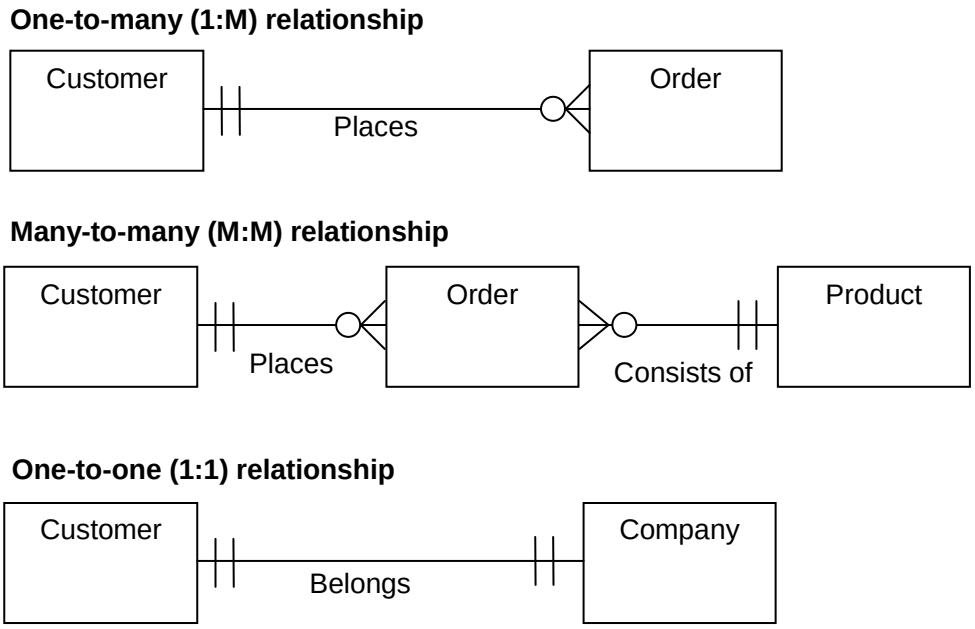


Figure 32 - Relationships

- Optionality and Cardinality

There are two type relationship in terms of connectivity of the relationship such as optional relationship and mandatory relationship. Cardinality expresses the minimum and maximum number of entity occurrences that associated with the related entity.

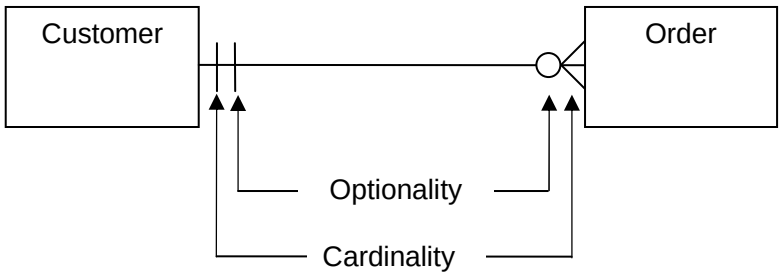
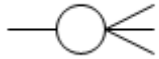
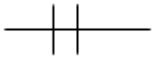
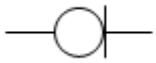


Figure 33 - Cardinality and Connectivity

Table 24 - Crow's Foot Symbols

Symbol	Cardinality	Remarks
	(0, N)	Many side is optional
	(1, N)	Many side is mandatory
	(1, 1)	1 side is mandatory
	(0,1)	1 side is optional

4.3.2 E-R Diagram Development

Before start drawing actual E-R diagrams, we also need to understand following concepts in order to design accurate E-R model.

- Primary Key

A primary key is a field composite field to uniquely identify each row of the instance of entity.

- Foreign Key

A foreign key is a field composite field in entity that points to a primary key of another entity to implement integrity rules between entities.

Customer ID in Order entity is a Foreign Key of Customer ID in Customer entity:

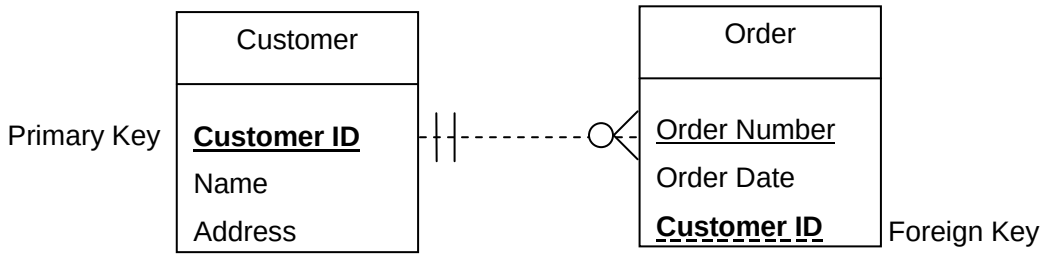


Figure 34 - Primary Key and Foreign Key

- Identifying (Strong) Relationship

A non-identifying relationship means that the child entity (many side of the entity) can not be

uniquely identified without the parent (one side of the entity).

- Non-Identifying (Weak) Relationship

A non-identifying relationship is one where the child entity can be identified independently of the parent entity.

- Sample E-R Diagram for Order and Stock database

The E-R diagram below is drawn by above concepts of E-R modeling:

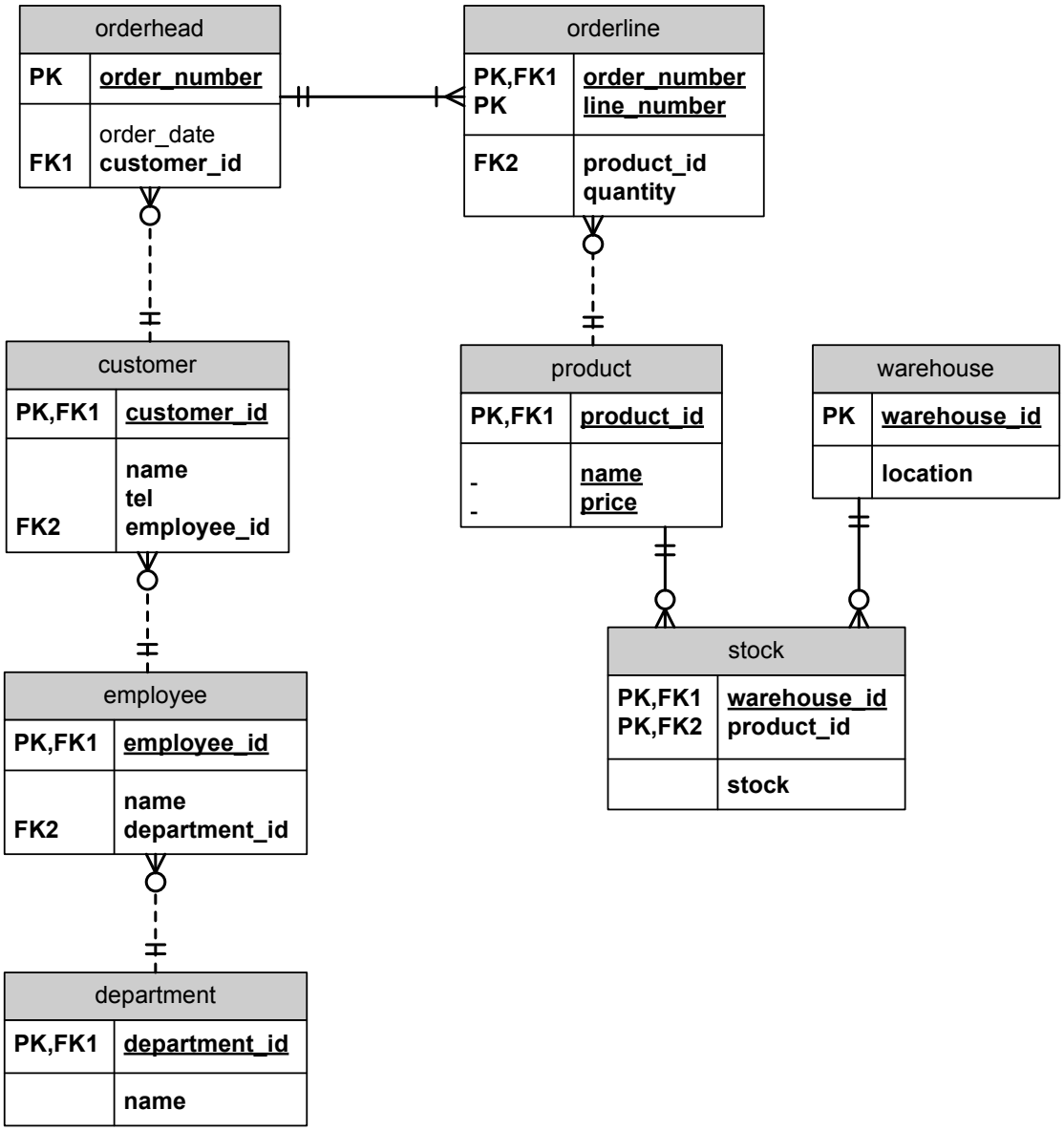


Figure 35 – Sample ER Diagram for Order and Warehouse Stock Database

Exercise 4-1. - Entity Relationship Modeling Practice1

1. To Install Eclipse

- Yast->Software Management-> type “Java”-> click “search’ button-> click “java 1.6”
- Copy “eclipse-jee-janymede-fall2-linux-gtk.tar” from usr/local/share/linux
- Extract (or) unzip this file
- Start Eclipse

2. Install Clay Database Modeling plug-in for Eclipse

- Extract “jp.azzurri.clay.core_1.4.0.bin.dist.20050831.zip” file
- Copy (overwrite) extracted “plugins” and “features” folders into Eclipse folder
- Start Eclipse and check Clay has been installed
  Window -> Preferences -> Clay Database Modeling

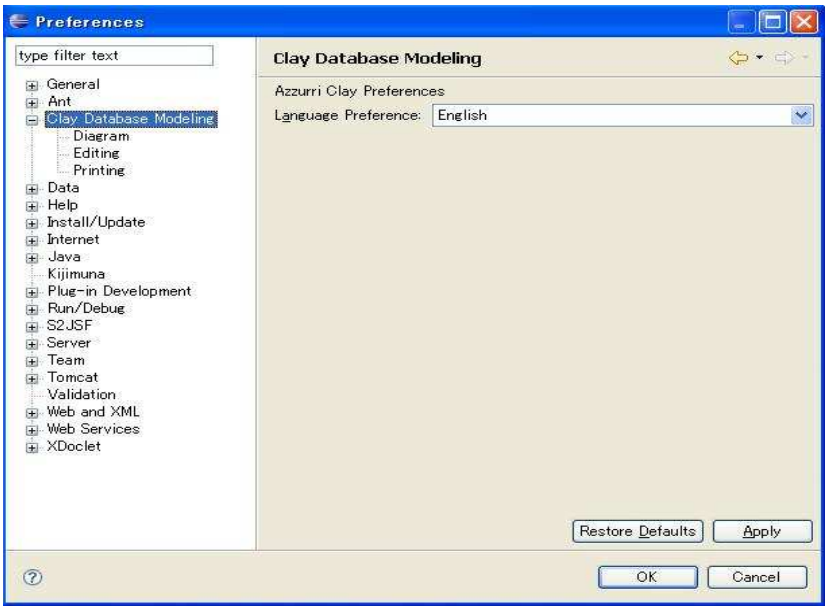


Figure 36 - Eclipse Clay Plug-in

Create and draw a Data Model (ER Diagram) with Eclipse Clay plug-in

- Create a new project

File -> New -> Project -> Java Project -> {Enter Project Name} -> and click Finish to create.

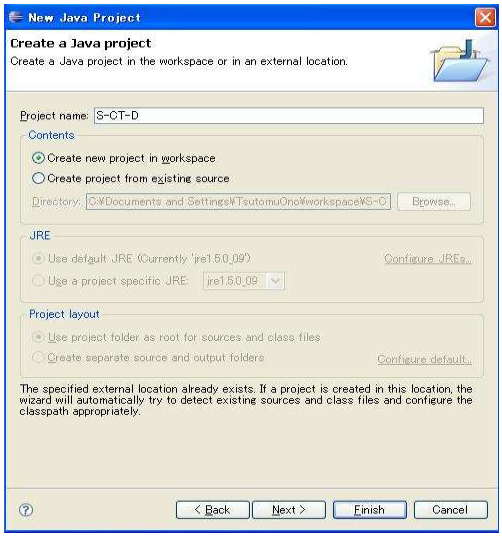


Figure 37 – Create Java Project

- Create a new database design diagram

File->New->Other, then select Database Modeling Azzurri Clay Database Design Diagram

Select SQL Dialect as **PostgreSQL7.3/7.4** or **MySQL 5.0** and click Finish

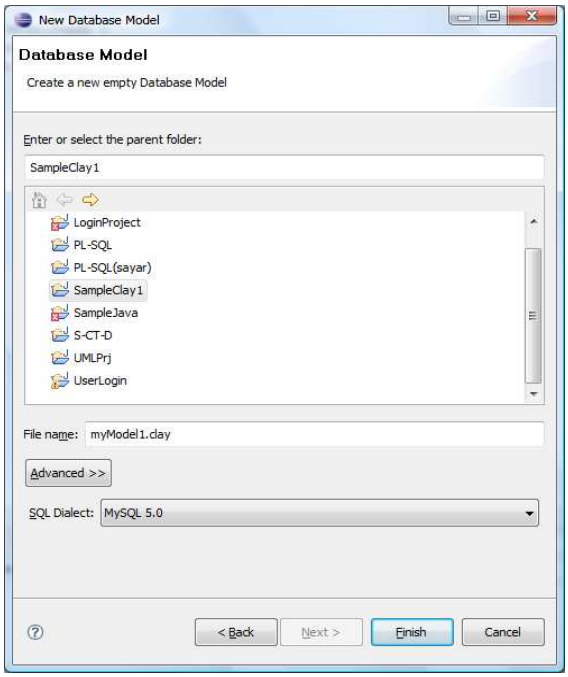


Figure 38 - Clay Database Model

- Draw ER diagram discussed in Chapter 3 Entity Relationship Modeling
  (Follow “Clay Quick Start Guide” for learn basic operation by yourself before drawing)

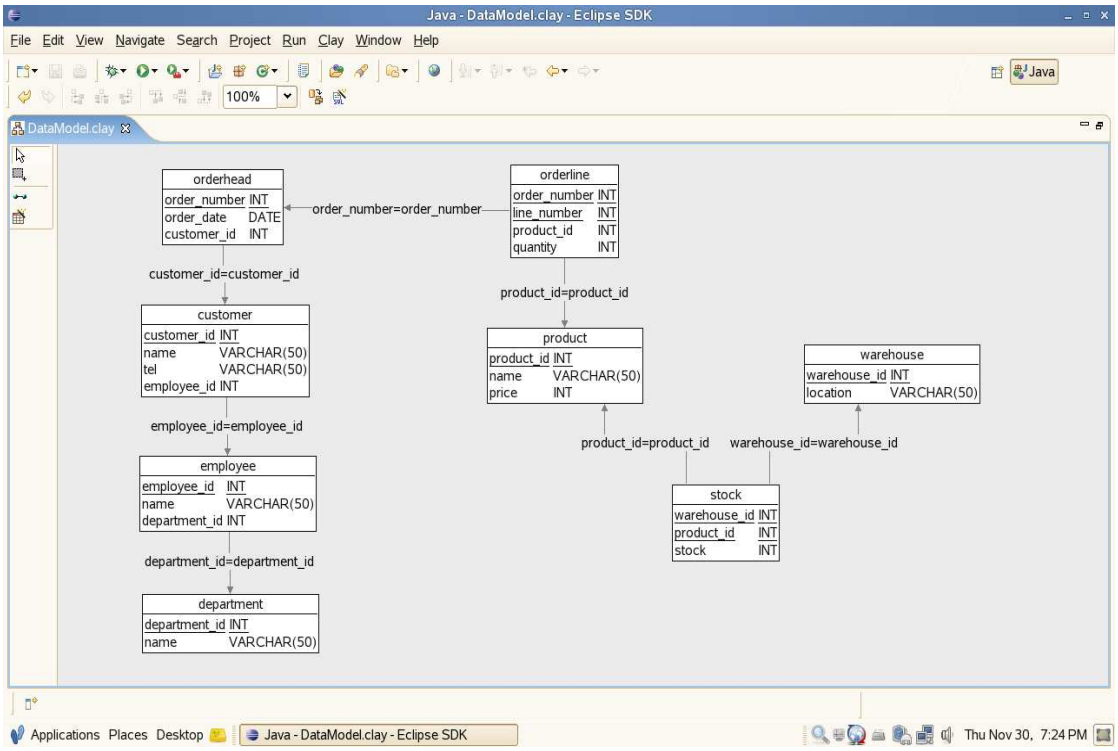


Figure 39 - Clay Database Model

Table 25 - Data Dictionary for sample Clay Database Model

Table	Attribute	Type	Format/ Range	Required	PK/ FK	FK Reference Table
employee	employee_id	INTEGER		Yes	PK	
	name	VARCHAR(50)		Yes		
	hire_date	DATE	DD/MM/YY	Yes		
	department_id	INTEGER		Yes	FK	department
department	department_id	INTEGER		Yes	PK	
	name	VARCHAR(50)		Yes		
customer	customer_id	INTEGER		Yes	PK	
	name	VARCHAR(50)		Yes		
	tel	VARCHAR(50)				
	employee_id	INTEGER		Yes	FK	employee
orderhead	order_number	INTEGER		Yes	PK	
	order_date	DATE				
	customer_id	INTEGER		Yes	FK	customer
orderline	order_number	INTEGER		Yes	PK	
	line_number	INTEGER		Yes	PK	
	product_id	INTEGER		Yes	FK	product
	quantity	INTEGER		Yes		
product	product_id	INTEGER		Yes	PK	
	name	VARCHAR(50)		Yes		
	price	INTEGER		Yes		
warehouse	warehouse_id	INTEGER		Yes	PK	
	location	VARCHAR(50)				
stock	warehouse_id	INTEGER		Yes	PK/FK	warehouse
	product_id	INTEGER		Yes	PK/FK	product
	stock	INTEGER				

Exercise 4-2. - Entity Relationship Modeling Practice2

Draw the E-R Diagram for the following **"**Phase 2 Software Development Workshop: Hotel Reservation System Reference Database Design".

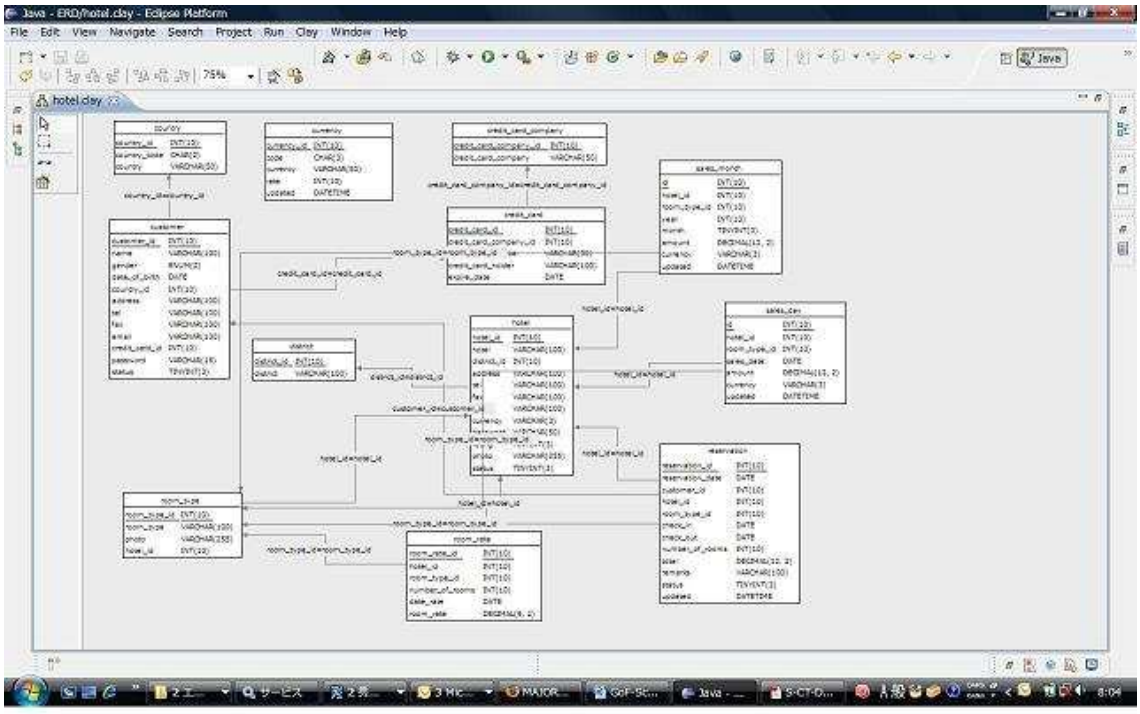


Figure 40 - Clay Database Model for Hotel Reservation System

Table 26 - Data Dictionary for Hotel Reservation System

Table	Attribute	Type	Required	PK/ FK	FK Reference Table
district	district_id	INTEGER	Yes	PK	
	district	VARCHAR(100)	Yes		
hotel	hotel_id	INTEGER	Yes	PK	
	hotel	VARCHAR(100)	Yes		
	district_id	INTEGER		FK	district
	address	VARCHAR(100)	yes		
	tel	VARCHAR(100)	yes		
	fax	VARCHAR(100)	yes		
	email	VARCHAR(100)	yes		
	currency	VARCHAR(3)			
	password	VARCHAR(50)	yes		
	rating	TINYINT			
	photo	VARCHAR(255)			

	status	TINYINT			
room_type	room_type_id	INTEGER	Yes	PK	
	room_type	VARCHAR(100)			
	photo	VARCHAR(255)			
	hotel_id	INTEGER	Yes	FK	hotel
room_rate	room_rate_id	INTEGER	Yes	PK	
	hotel_id	INTEGER		FK	hotel
	room_type_id	INTEGER	Yes	FK	room_type
	number_of_room	INTEGER	Yes		
	date_rate	DATE	Yes		
	room_rate	DECIMAL(9,2)			
credit_card_company	credit_card_company_id	INTEGER	Yes	PK	
	credit_card_company	VARCHAR(50	Yes	PK	
credit_card	credit_card_id	INTEGER	Yes	PK	
	credit_card_company_id	INTEGER	Yes	FK	credit_card_company
	credit_card_number	VARCHAR(50)	Yes		
	credit_card_holder	VARCHAR(100)	Yes		
	expire_date	DATE	Yes		
country	country_id	INTEGER	Yes	PK	
	country_code	CHAR(2)	Yes		
	country	VARCHAR(50)	Yes		
customer	customer_id	INTEGER	Yes	PK	
	name	VARCHAR(50)	Yes	PK/FK	product
	gender	ENUM	Yes		
	date_of_birth	DATE	Yes		
	country_id	INTEGER	Yes	FK	country
	address	VARCHAR(100)	Yes		
	tel	VARCHAR(100)	Yes		
	fax	VARCHAR(100)	Yes		
	email	VARCHAR(100)	Yes		
	credit_card_id	INTEGER	Yes	FK	credit_card
	password	VARCHAR(16)			
	status	TINYINT			
reservation	reservation_id	INTEGER	Yes	PK	

	reservation_date	DATE	Yes		
	customer_id	INTEGER	Yes	FK	customer
	hotel_id	INTEGER	Yes	FK	hotel
	room_type_id	INTEGER	Yes	FK	room_type
	check_in_date	DATE	Yes		
	check_out_date	DATE	Yes		
	number_of_rooms	INTEGER	Yes		
	total	DECIMAL(12,2)	Yes		
	remarks	VARCHAR(100)	Yes		
	status	TINYINT	Yes		
	updated	DATETIME	Yes		
sales_day	id	INTEGER	Yes	PK	
	hotel_id	INTEGER	Yes	FK	hotel
	room_type_id	INTEGER	Yes	FK	room_type
	sales_date	DATE	Yes		
	amount	DECIMAL(12,2)	Yes		
	currency	VARCHAR(3)	Yes		
	updated	DATETIME	Yes		
sales_month	id	INTEGER	Yes	PK	
	hotel_id	INTEGER	Yes	FK	hotel
	room_type_id	INTEGER	Yes	FK	room_type
	year	INTEGER	Yes		
	month	TINYINT	Yes		
	amount	DECIMAL(12,2)	Yes		
	currency	VARCHAR(3)	Yes		
	updated	DATETIME	Yes		
email_log	id	INTEGER	Yes	PK	
	reservation_id	INTEGER	Yes		
	email_hotel	VARCHAR(100)	Yes		
	email_customer	VARCHAR(100)	Yes		
	updated	DATETIME	Yes		
payment_log	id	INTEGER	Yes	PK	
	reservation_id	INTEGER	Yes		
	credit_card_id	INTEGER	Yes		
	updated	DATETIME	Yes		
currency	currency_id	INTEGER	Yes	PK	
	code	CHAR(3)	Yes		
	currency	VARCHAR(50)	Yes		

	rate	INTEGER	Yes		
	updated	DATETIME	Yes		

4.4 Normalization

4.4.1 Introduction to Normalization

Normalization is the process of evaluating and restructuring the logical data model of a database to eliminate or minimize redundancy, organize data efficiently in order to reduce repeating data and to reduce the likelihood of data anomalies. It may also improve data consistency and simplify future extension of the logical data model. Normalization process through a series of stage called normal forms (NF) and most of rhetorical books of the database discuss the stage as below:

- First normal form (**1NF**) lays the groundwork for an organized database design:
  1. Ensure that each table has a primary key: minimal set of attributes which can uniquely identify a record.
  2. Eliminate repeating groups (categories of data which would seem to be required a different number of times on different records) by defining keyed and non-keyed attributes appropriately.
  3. Atomicity: Each attribute must contain a single value, not a set of values.
- Second normal form (**2NF**) If a table has a composite key, all attributes must be related to the whole key:
  1. The database must meet all the requirements of the first normal form.
  2. Data which is redundantly duplicated across multiple rows of a table is moved out to a separate table.
- Third normal form (**3NF**) requires that data stored in a table be dependent only on the primary key, and not on any other field in the table:
  1. The database must meet all the requirements of the second normal form.
  2. Any field which is dependent not only on the primary key but also on another field is moved out to a separate table.

4.4.2 Normalization Process

The figure below illustrates how original form (or data model) go through the normalization process such as 1NF, 2NF and 3NF:

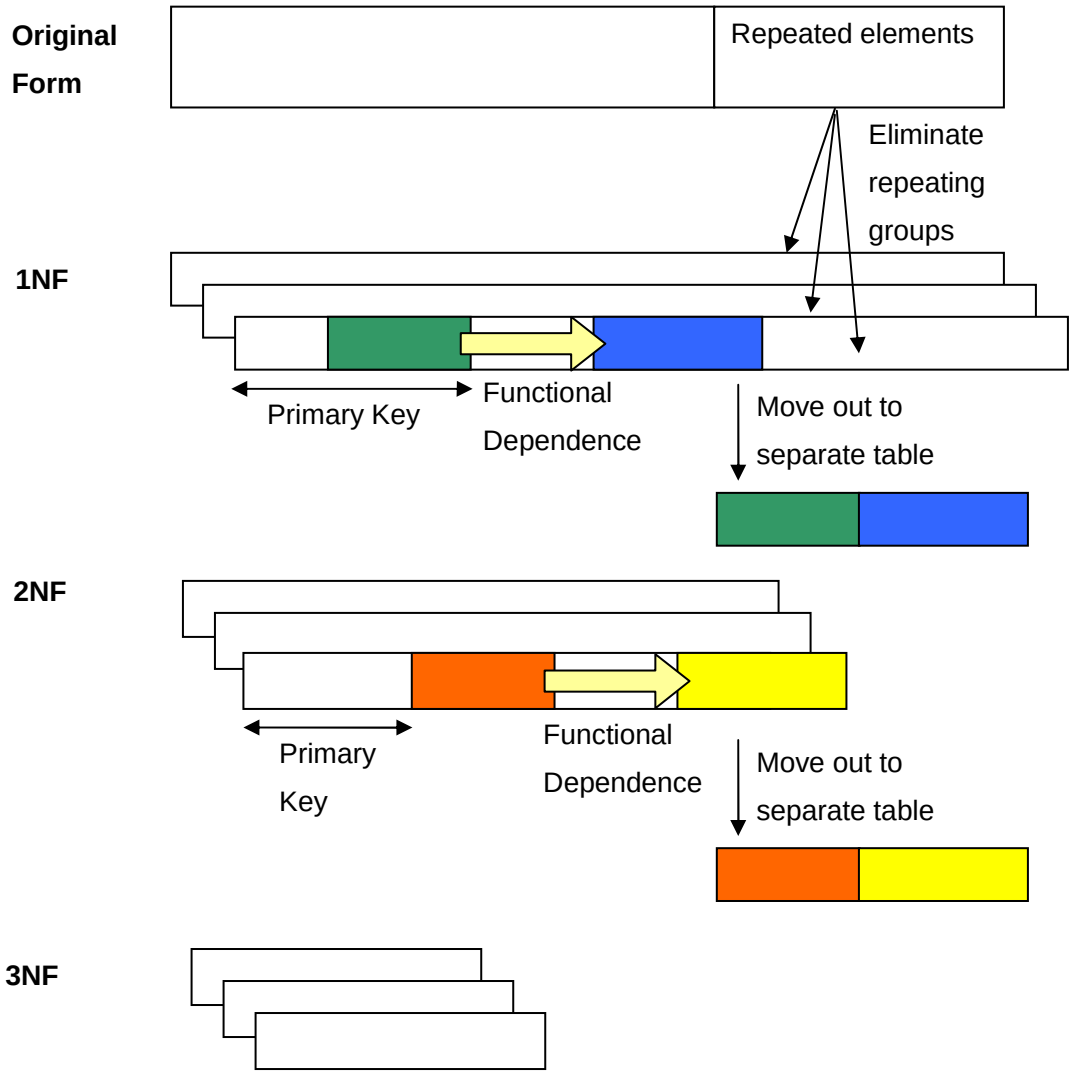


Figure 41 – Process original form to 1NF, 2NF and 3NF

Table below is a sample original form for sales orders that is to go through the normalization process. An attribute with underline represents the primary key of the form.

Table 27 - Sample original form

Order Number	Order Date	Customer ID	Customer Name	Customer Address	Product ID Product Quantity Unit Price	Product ID Product Quantity Unit Price	Product ID Product Quantity Unit Price
1	10/01/2007	1	ABC	Yangon	A01 Pencil 5 20	B99 Notebook 1 95	C01 Eraser S 3 50
2	31/01/2007	2	XYZ	Bagan	C01 Eraser S 1 50	C02 Eraser M 1 60	C03 Eraser L 1 80
3	31/01/2007	1	ABC	Yangon	A01 Pencil 1 20		

(1) First Normal Form (1NF)

Eliminate repeating group of order detailed information by splitting into four attributes such as Product ID, Product, Quantity and Unit Price as shown in Table below:

Table 28 - 1NF

Order Number	Order Date	Customer ID	Customer Name	Customer Address	Product ID	Product	Quantity	Unit Price
1	10/01/2007	1	ABC	Yangon	A01	Pencil	5	20
1	10/01/2007	1	ABC	Yangon	B99	Notebook	1	95
1	10/01/2007	1	ABC	Yangon	C01	Eraser S	3	50
2	31/01/2007	2	XYZ	Bagan	C01	Eraser S	1	50
2	31/01/2007	2	XYZ	Bagan	C02	Eraser M	1	60
2	31/01/2007	2	XYZ	Bagan	C03	Eraser L	1	80
3	31/01/2007	1	ABC	Yangon	A01	Pencil	1	20

(2) Second Normal Form (2NF)

Remove functional dependences for the primary key (Order Number and Product ID) attributes to separate tables such as Order, Order Details and Product tables as shown in Table below:

Table 29 - 2NF

Order

Order Number	Order Date	Customer ID	Customer Name	Customer Address
1	10/01/2007	1	ABC	Yangon
2	31/10/2007	2	XYZ	Bagan
3	31/10/2007	1	ABC	Yangon

Order Details (2NF to be continued…)

Order Number	Product ID	Quantity	Unit Price
1	A01	5	20
1	B99	1	95
1	C01	3	50
2	C01	1	50
2	C02	1	60
2	C03	1	80
3	A01	1	20

Order Details

Order Number	Product ID	Quantity
1	A01	5
1	B99	1
1	C01	3
2	C01	1
2	C02	1
2	C03	1
3	A01	1

Product

Product ID	Product	Unit Price
A01	Pencil	20
B99	Note	95
C01	Eraser S	50
C02	Eraser M	60
C03	Eraser L	80

(3) Third Normal Form (3NF)

Remove functional dependences for non-primary key (Order Number and Product ID) attributes to separate tables such as Customer as shown in Table below:

Table 30 - 2NF

Order

Order Number	Order Date	Customer ID
1	10/01/2007	1
2	10/01/2007	2

Order Details

Order Number	Product ID	Quantity
1	A01	5
1	B99	1
1	C01	3
2	C01	1
2	C02	1
2	C03	1
3	A01	1

Product

Product ID	Product	Unit Price
A01	Pencil	20
B99	Note	95

C01	Eraser S	50
C02	Eraser M	60
C03	Eraser L	80

Customer

Customer ID	Customer Name	Customer Address
1	ABC	Yangon
2	XYZ	Bagan

4.4.3 Normalization and Database Design

Relational database model based on the normalization process will be described as shown in Figure below:

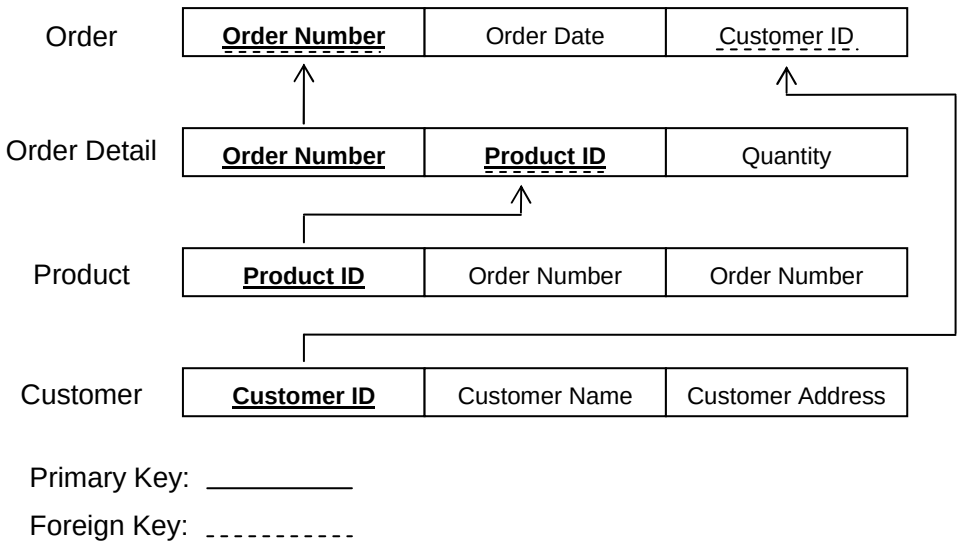


Figure 42 - Relational Database Model

Exercise 5. - Normalization Practice

Normalize following form and draw ER diagram with Eclipse Clay plug-in

Table 31 - Travel Agent Flight Information

Airline Code	Flight No.	Airline	Aircraft Code	Aircraft	Departure	Arrival	Departure Time	Arrival Time	Airfare (USD)
SQ	777	Singapore Airline	B76	B-767	Singapore	Yangon	7:00	9:00	Normal: 100, Off-season: 50, Holiday: 200, Peak-season: 250
TG	888	Thai Airways	A30	A300	Bangkok	Yangon	7:00	8:00	Normal: 50, Off-season: 30, Holiday: 100, Peak-season:160
MH	777	Malaysia Airline	A30	A300	Kuala Lumpur	Yangon	7:00	8:30	Normal: 80, Off-season: 60, Holiday: 160, Peak-season: 200

Note: Airfare can be categorized into four seasons such as normal off, holiday and peak.

- Process to come up with first normal form (1NF) from Table above
- Process to come up with second normal form (2NF) from the 1NF
- Process to come up with third normal form from (3NF) from the 2NF
- Draw final database design diagram with Eclipse Clay plug-in

Following Exercise will be carried out after learning about “Introduction to SQL”:

- Generate a DDL with Clay plug-in and create the tables
- Insert sample data for each table
- Write SQL (Join Tables query) to display as same as table above
- Create a View for above Join Tables query

5 Introduction to SQL

Structured Query Language (SQL) is non-procedural language used to create, modify, retrieve and manipulate data from relational database management systems. It consists of two parts such as basic SQL as ANSI/ISO standard and the own direct SQL extended by each RDBMS. All the SQL commands introduced in this chapter is based on the ANSI/ISO standard SQL and with a sample Sales Order database which is composed of the following tables as depicted in the following figures:

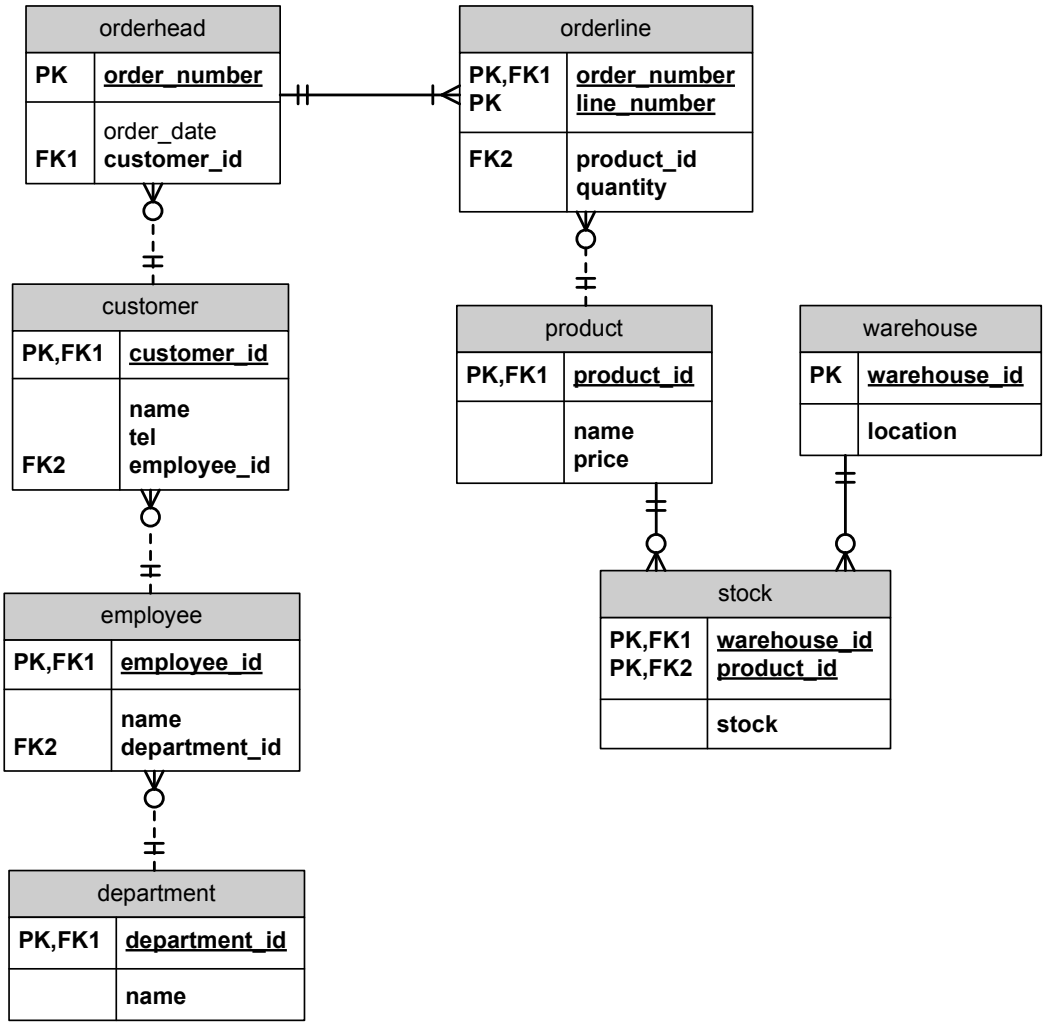


Figure 43 – E-R diagram of Sales Order database

orderhead

order_number [PK] int4	order_date date	customer_id int4
1	2006-12-25	103
2	2006-12-25	104
3	2007-01-10	101
4	2007-01-15	101

orderline

order_number [PK] int4	line_number [PK] int4	product_id int4	quantity int4
1	1	101	10
1	2	102	10
2	1	103	100
2	2	104	50
2	3	105	50
3	1	101	3
4	1	102	6

customer

customer_id [PK] int4	name varchar	tel varchar	employee_id int4
101	ABC Ltd.		1001
102	XZZ Ltd.	222-2222	1001
103	ICTTI	333-3333	1002
104	JICA	444-4444	1002

product

product_id [PK] int4	name varchar	price int4
101	Pencil	100
102	Notebook	250
103	Eraser S	40
104	Eraser M	60
105	Eraser L	80

employee

employee_id [PK] int4	name varchar	department_id int4
1001	Nyaing	10
1002	Lwin	10
1003	Yee	20
1004	Naing	20
1005	Thein	30
1006	Ohn	30

stock

warehouse_id [PK] int4	product_id [PK] int4	stock int4
10	101	100
10	102	100
10	103	500
10	104	600
10	105	800
20	101	50
20	102	50
30	103	200
30	104	200
30	105	200

department

department_id [PK] int4	name varchar
10	Sales
20	Accounting
30	Production

warehouse

warehouse_id [PK] int4	location varchar
10	Yangon
20	Bagan
30	Mandalay

Figure 44 – Sample data of Sales Order database tables

5.1 Data Definition Language (DDL)

Data Definition Language (DDL) is a set of commands to define database objects such as create/drop database schema, table and view, and grant/revoke privileges of database objects to database users. Table below is summary of DDL commands:

Table 32 - DDL Commands

Command	Description
CREATE SCHEMA	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have null value
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column
CHECK	Constraint used to validate data in an attribute
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows/columns from one or more tables
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
ALTER TABLE	Adds, modifies, or deletes attributes or constraints for a table
DROP TABLE	Permanently deletes a table
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view
GRANT	Grants access permission of database object to user
REVOKE	Revokes access permission of database objects from user

SQL syntax in this chapter describes with following rules:

Bold: SQL statements
Italic: name of objects or conditions
[ ]: Conditional statement
,,,: Statement that can be repeated

5.1.1 CREATE

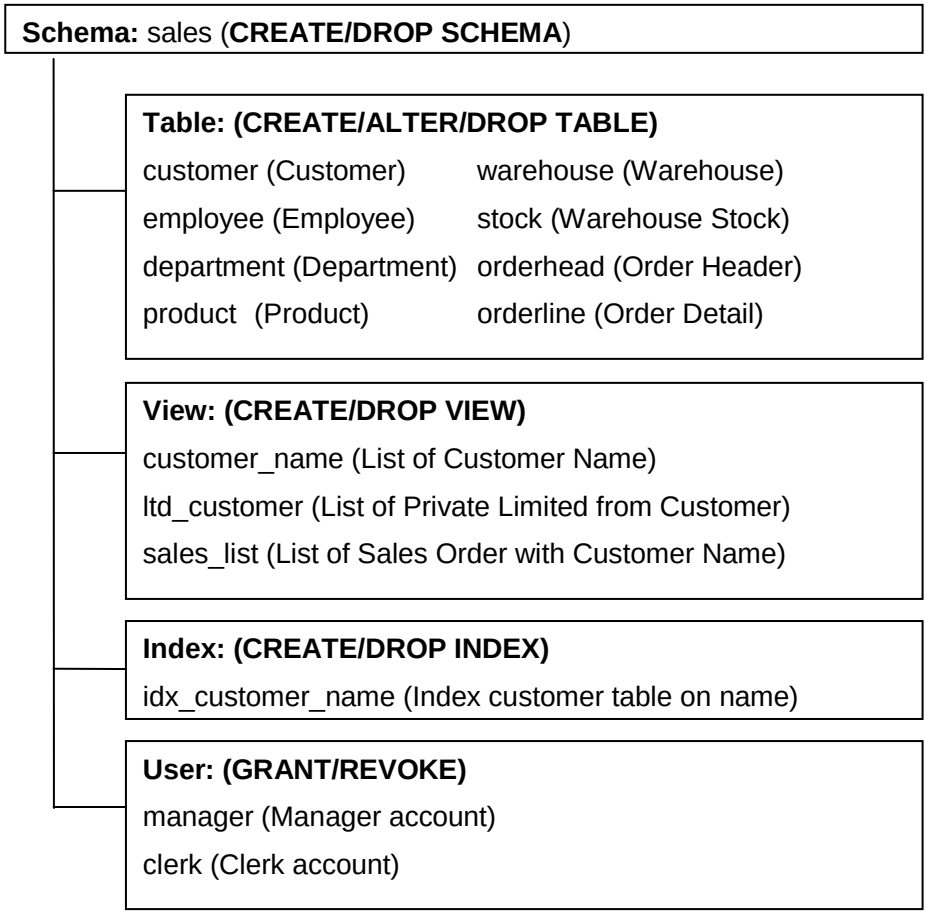


Figure 45 - Sales Order database object structure to be created in this chapter

- CREATE SCHEMA

Schema is a namespace for database objects belongs to the same database. CREATE SCHEMA creates the namespace as a logical database segment in the database.

Syntax:

CREATE SCNEMA schema_name [**AUTHORIZATION**] user_name

Example:

Creates **sales** schema in the database **s-ct-d** and authorized it to user **postgres**:

Connect s-ct-d database

```
CREATE SCHEMA sales AUTHORIZATION postgres;
```

- CREATE TABLE

CREATE TABLE creates database tables.

Syntax:

```
CREATE TABLE table_name (field_name data_type [column_constranit]
                [,field_name data_type]
                [table_constraint])
```

Data Types:

Table below is a list of some of the commonly used data types for PostgreSQL and MySQL:

Table 33 – PostgreSQL and MyQL Data Types

PostgreSQL	MySQL	Type	Description
integer	integer	Numeric	Signed four-byte integer
smallint	smallint	Numeric	Signed two-byte integer
bigint	bigint	Numeric	Signed eight-byte integer
real	real	Numeric	Single precision floating-point number
double precision	double	Numeric	Double precision floating-point number
numeric(p,s)	numeric(p,s)	Numeric	Exact numeric of selectable precision (p: Precision, s: scale)
decimal (p,s)	decimal (p,s)	Numeric	
char(n)	char(M)	Character	Fixed-length character string
varchar(n)	varchar(n)	Character	Variable-length character string
Text	Text	Character	Variable-length character string
Date	Date	Date/Time	Calendar date (year, month, day)
Time	Time	Date/Time	Time of day
timestamp	timestamp	Date/Time	Date and time
Boolean	Bit	Boolean	Logical Boolean (true/false)

Note: (n): n byte, (M): M character

Constraints:

There are two types of constraint; such as column constraint and table constraint:

Table 34 - Column Constraint

Constraint	Description
DEFAULT	Defines a default value for a column
NOT NULL	Ensures that a column will not have null value
PRMARY KEY	Defines primary key of a table
UNIQUE	Ensures that a column will not have duplicate values
REFERENCES	Defines referential integrity
CHECK	Constraint used to validate data in an attribute

Table 35 - Table Constraint

Constraint	Description
PRIMARY KEY (Composite Primary Key)	Defines primary key of a table which consists of one or more attributes
UNIQUE	Ensures that a column will not have duplicate values
FOREIGN KEY	Defines foreign key of a table
CHECK	Constraint used to validate data in an attribute

Table Constraint:

Example:

Creates orderhead and order_line tables in the sales schema under the s-ct-d database:

```
CREATE TABLE sales.orderhead (
    order_number INT NOT NULL
  , order_date DATE
  , customer_id INT NOT NULL
  , PRIMARY KEY (order_number)
  , CONSTRAINT FK_orderhead_1 FOREIGN KEY (customer_id)
        REFERENCES sales.customer (customer_id) );
CREATE TABLE sales.order_line (
    order_number INT NOT NULL
  , line_number INT NOT NULL
  , product_id INT NOT NULL
  , quantity INT NOT NULL CHECK (quantity>=1)
  , PRIMARY KEY (order_number, line_number)
  , CONSTRAINT FK_order_line_1 FOREIGN KEY (order_number)
        REFERENCES sales.orderhead (order_number)
  , CONSTRAINT FK_order_line_2 FOREIGN KEY (product_id)
        REFERENCES sales.product (product_id) );
```

- CREATE INDEX

CREATE INDEX creates an index for a table. Details of index will be discussed in Database Design and Administration in Software Development Track.

Syntax:

```
CREATE [UNIQUE] INDEX name ON table(field, [field],,,,)
[USING method] [WHERE condition];
```

Example:

Creates index for customer table on name column:

```
CREATE INDEX idx_customer_name ON customer(name);
```

- CREATE VIEW

View is a virtual table based on a SELECT query. CREATE VIEW creates a dynamic subset of rows/columns from one or more tables.

Syntax:

```
CREATE [OR REPLACE] VIEW name AS select_query;
```

Example:

```
Creates list of Customer Name from customer table:
CREATE VIEW view_customer_name AS SELECT customer_id, name
FROM customer;

Creates list of Private Limited from Customer from customer table:
CREATE VIEW view_ltd_customer AS SELECT * FROM customer
WHERE name LIKE '%Ltd.';

Creates list of Sales Order with customer name from customer and orderhead tables:
CREATE VIEW view_sales_list(order_number, order_date, name)
AS SELECT o.order_number, o.order_date, c.name
FROM orderhead o, customer c
WHERE o.customer_id = c.customer_id;

Please refer to 4.2. DML for SELECT statements details
```

5.1.2 ALTER TABLE

ALTER TABLE allows changing existing database table definitions. Three options such as ADD, DROP and ALTER are available.

Syntax:

ALTER TABLE table_name [options];

Example:

Add a column to customer table:
ALTER TABLE customer ADD address varchar(50);
(OR)
ALTER TABLE customer ADD address varchar(50) AFTER name;
This column must be placed right behind the “name” column.

Add two new columns to customer table:
ALTER TABLE customer
ADD (address varchar(50) NOT NULL, gender varchar(2) NOT NULL);

Change column name “address” to “location”:
ALTER TABLE customer CHANGE address location varchar(50);

Increase/Decrease length of a column:
ALTER TABLE customer CHANGE location location varchar(30);

Modify column data type in customer table:
ALTER TABLE customer CHANGE location location text;

Drop a column from customer table;
ALTER TABLE customer DROP address; (OR)
ALTER TABLE customer DROP address varchar(50);

5.1.3 DROP

DROP allows to drop (delete) database objects created by CREATE commands

Syntax:

DROP SCHEMA schema_name [CASCADE | RESTRICT];
DROP TABLE table_name [CASCADE | RESTRICT];
DROP VIEW schema_name [CASCADE | RESTRICT];
DROP INDEX index_name [CASCADE | RESTRICT];

Examples:

```
DROP SCHEMA sales CASCADE;
DROP TABLE department CASCADE;
DROP VIEW view_customer_name;
DROP INDEX idx_customer_name;

Note: CASDADE is option to delete all the dependent objects.
```

5.1.4 GRANT

GRANT grants (gives) access permission of database object to user.

Syntax:

```
GRANT privileges ON TABLE table_name TO user_name;
```

Example:

```
-- Grants all access permission for customer table to manager:
GRANT ALL ON customer TO manager;
-- Grant SELECT and UPDATE permissions for customer table to clerk:
GRANT SELECT, UPDATE ON customer TO public;
```

5.1.5 REVOKE

REVOKE revokes (removes) access permission of database object from user.

Syntax:

```
REVOKE privileges ON TABLE table_name FROM user_name;
```

Example:

```
-- Revoke all access permission for customer table from manager:
REVOKE ALL ON customer FROM manager;
-- Revoke SELECT and UPDATE permissions for customer table from clerk:
REVOKE SELECT, UPDATE ON customer FROM clerk;
```

Exercise 6. - DDL practice with PostgreSQL and MySQL

Create Database, Users and Schema

- Create Database (Database name: s-ct-d)
- Create users

  Username: manager (Password: manager)

  Username: clerk (Password: clerk)
- Create schema (Schema name: sales) Note: This practice is only for PostgreSQL
- Grant customer table ALL to manager, grant SELECT, UPDATE to clerk

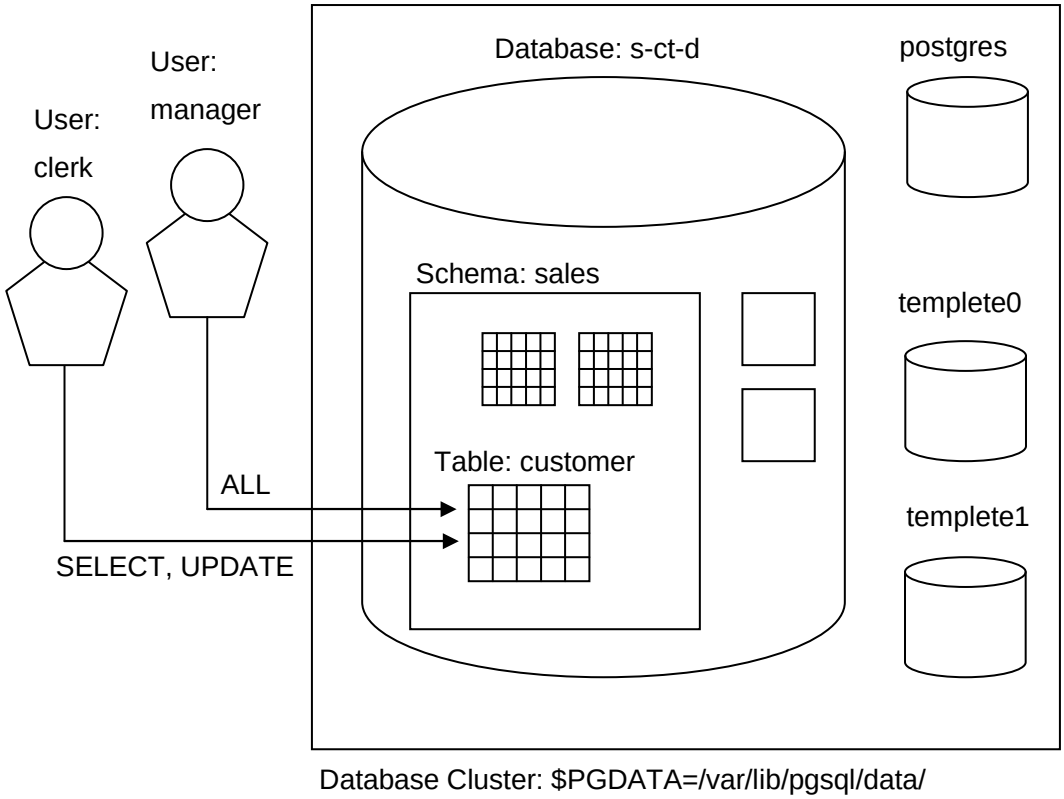


Figure 46 - PostgreSQL database cluster, database, user and schema

Note: Following SET is to enable viewing schema

```
s-ct-d=# SET search_path =  public, sales;
s-ct-d=# SHOW search_path;
```

5.1.5.1.1 Create Tables with DDL script

- Generate SQL script with Clay
  - Run Eclipse
  - Open Database Design Model done in Exercise 3.1
  - Select from Menu -> Clay -> Generate SQL (CREATE TABLE) Script to generate
  - Modify schema name in the generated script as “sales” if the name is different
  - Copy the generated script and past it into either psql or pgAdmin III Query tool
  - Confirm the tables are created in the database schema

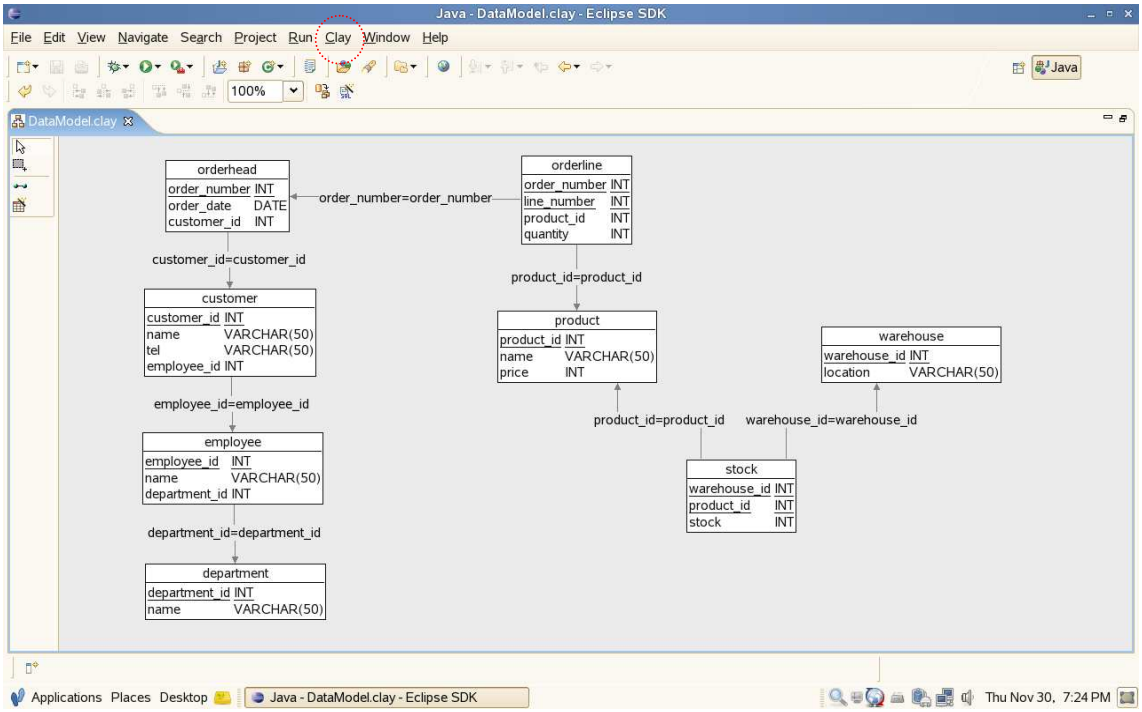


Figure 47 - Sales Order database created by reverse engineering

5.1.5.1.2 Insert sample data

- Insert sample data either with psql or pgAdmin III Query tool (Copy & Paste)
  postgres# psql –f [FILE NAME] [DATABASE NAME]

- View sample data with either psql or pgAdmin III

5.1.5.1.3 Grant and revoke privileged

- Grant permission

  Grant all privileges on “customer” table to user “manager”

  Grant SELECT and UPDATE privileges on “customer” table to user “clerk”

- Test granted privileges

  Log on to PostgreSQL as “manager” to check granted privileges on customer

  Log on to PostgreSQL as “clerk” to check granted privileges on customer

- Revoke privileges

  Revoke privileges from “manager”

  Revoke privileges from “clerk”

5.2 Data Manipulation Language (DML)

Data Manipulation Language (DML) is a set of commands to insert, update, delete and retrieve data within the database tables. Table below is summary of DML commands:

Table 36 - DML commands

Command	Description
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
INSERT	Insert one or more rows into a table
UPDATE	Modifies an attribute's values in one or more table's row
DELETE	Deletes one or more rows from a table

Table 37 - DML commands: Operators and Functions

Operators	
Comparison Operators: =, <, >, <=, >=, <>	Comparison Operators which will be used in conditional expressions
Logical Operators: AND/OR/NOT	Logical Operators which will be used in conditional expressions
Special Operators:	
BETWEEN	Checks an attribute value is within a range or not
IS NULL	Checks an attribute value is null or not
LIKE	Checks an attribute value matches a given string pattern
IN	Checks an attribute value matches any value within a given value list
EXISTS	Checks a sub-query returns any rows or not
DISTINCT	Limits values to unique values
Aggregate Functions:	
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column

AVG	Returns the average of all values for a given column

5.2.1 SELECT

SELECT selects attributes from rows in one or more tables or views.

Syntax:

```
SELECT column list
FROM table list
[WHERE condition list];
```

- Basics usage of SELECT

Examples:

- Select all attributes and rows from customer table:

```
SELECT * FROM customer;
```

- Count number of rows from customer table using aggregate function:

```
SELECT COUNT(*) FROM customer;
```

- Projection (Project Customer ID and Customer Name from customer table):

```
SELECT customer_id, name FROM customer;
```

- Selection (Select row(s) from customer table with specified Customer ID):

```
SELECT * FROM customer WHERE customer_id = 1;
```

- SELECT useful tips

Examples:

- Operator (Returns 20% discount price list from product table):

```
SELECT product_id, name, price * 0.8 AS disc_price FROM product;
```

- Concatenation (Concatenate ‘Product Name:’ and name to display as one attribute):

-Change MySQL mode to concatenation string

```
SET sql_mode=”PIPES_AS_CONCAT”;
SELECT 'Product Name: ' || name AS prod_name FROM product;
```

- Alias (c as alias of customer table):

```
SELECT c.customer_id FROM customer c WHERE c.customer_id = 101;
```

- Distinct rows (Get rid of redundant rows and display as one row):

```
SELECT DISTINCT employee_id FROM customer;
```

- SELECT with WHERE condition

Examples:

- BETWEEN (Returns rows of Order Date are within a range of December 2006):

```
SELECT * FROM orderhead WHERE order_date
BETWEEN '2006-12-01' AND '2006-12-31';
```

- IN (Returns rows of Order Date are matches within a given '2006-12-25', '2007-01-10'):

```
SELECT * FROM orderhead WHERE order_date
IN ('2006-12-25', '2007-01-10');
```

- LIKE (Returns rows of Customer Name matches a given string pattern using %, _):

```
SELECT * FROM customer WHERE name LIKE '%Ltd.%';
SELECT * FROM customer WHERE name LIKE '_BC%';
SELECT * FROM customer WHERE name LIKE 'JIC_';
%: One string
_: One character
```

- IS NULL (Returns rows of telephone number is null):

```
SELECT * FROM customer WHERE tel IS NULL;
```

- EXISTS (Checks a sub-query returns any rows or not):

```
SELECT ,,, WHERE EXISTS (Sub-query);
Please refer to Sub-query for details
```

- NOT (NOT negate BETWEEN, IN, LIKE IS NULL, EXISTS):

```
SELECT * FROM orderhead WHERE order_date
NOT BETWEEN '2006-12-01' AND '2006-12-31';

SELECT * FROM orderhead WHERE order_date
NOT IN ('2006-12-25', '2007-01-10');
```

```
SELECT * FROM customer WHERE name NOT LIKE '%Ltd.%';
SELECT * FROM customer WHERE tel IS NOT NULL;
```

- SELECT with GROUP BY and Aggregate Functions

GROUP BY groups the selected rows based on one or more attributes

Syntax:

```
SELECT column list
FROM table list
GROUP BY column, ,,,
[HAVING condition]
```

Examples:

- Returns a list of total quantity grouped by Order Number from Order Detail table:

```
SELECT order_number, SUM(quantity) AS total FROM orderline
GROUP BY order_number;
```

- Returns a list of total quantity grouped by Product ID from Order Detail table:

```
SELECT product_id, SUM(quantity) AS total FROM orderline
GROUP BY product_id;
```

-HAVING restricts the selection of grouped rows based on a condition:

```
SELECT order_number, SUM(quantity) AS total FROM orderline
GROUP BY order_number HAVING COUNT(*) >= 2;

 order_number | total
--------------+-------
            2 |   200
            1 |    20

SELECT product_id, SUM(quantity) AS total FROM orderline GROUP BY product_id
HAVING COUNT(*) >= 2;
 product_id | total
------------+-------
        102 |    16
        101 |    13
```

- SELECT with ORDER BY

ORDER BY orders the selected rows based on one or more attributes.

Examples:

```
SELECT customer_id, name FROM customer ORDER BY name ASC;
 customer_id |   name
-------------+----------
         101 | ABC Ltd.
         103 | AICTI
         104 | JICA
         102 | XYZ Ltd.

SELECT customer_id, name FROM customer ORDER BY name DESC;
 customer_id |   name
-------------+----------
         102 | XYZ Ltd.
         104 | JICA
         103 | AICTI
         101 | ABC Ltd.
```

- Join SELECT

There are two types of joins are classified such as INNER JOIN and OUTER JOIN. Details are described in the Table and the Figure below:

Table 38 - Join SELECT styles

Join Classification	Join Type	SQL syntax	Description
INNER	JOIN	SELECT * FROM T1, T2 WHERE T1.C1 = T2.C1	Returns only the rows that meet the joining condition between T1 and T2.
	NATURAL JOIN	SELECT * FROM T1 NATURAL JOIN T2	
	JOIN USING	SELECT * FROM T1 JOIN T2 USING(C1)	
	JOIN ON	SELECT * FROM T1 JOIN T2 ON T1.C1 = T2.C1	
OUTER	LEFT JOIN	SELECT * FROM T1 LEFT OUTER JOIN T2	Returns rows with matching values and includes all rows from the **left table** (T1) with

		ON T1.C1 = T2.C1	unmatched values.
	RIGHT JOIN	SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.C1 = T2.C1	Returns rows with matching values and includes all rows from the **right table** (T2) with unmatched values.
	FULL JOIN	SELECT * FROM T1 FULL OUTER JOIN T2 ON T1.C1 = T2.C1	Returns rows with matching values and includes all rows from the **both tables** with unmatched values.

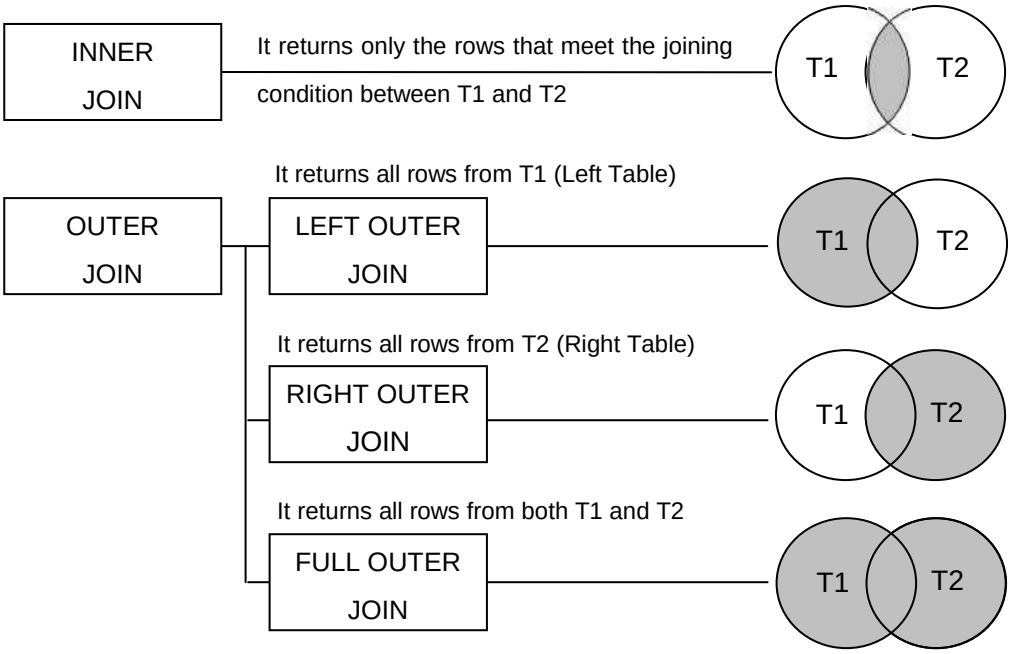


Figure 48 - Join Classification

Inner Join

- Join

```
SELECT o.order_number, o.order_date, c.customer_id, c.name
FROM sales.orderhead o, sales.customer c
WHERE o.customer_id = c.customer_id;
```

- Natural Join

```
SELECT o.order_number, o.order_date, c.customer_id, c.name
FROM sales.orderhead o NATURAL JOIN sales.customer c;
```

- Join Using

```
SELECT o.order_number, o.order_date, c.customer_id, c.name
FROM sales.orderhead o JOIN sales.customer c USING(customer_id);
```

- Join On

```
SELECT o.order_number, o.order_date, c.customer_id, c.name
```

```
FROM sales.orderhead o
JOIN sales.customer c ON o.customer_id = c.customer_id;
```

Order (sales.orderhead)

order_number integer	order_date date	customer_id integer
1	2006-12-25	103
2	2006-12-25	104
3	2007-01-10	101
4	2007-01-15	101
5	2007-01-21	999

Customer (sales.customer)

customer_id integer	name character	tel character	employee_id integer
101	ABC Ltd.		1001
102	XYZ Ltd.	222-2222	1001
103	ICTTI	333-3333	1002
104	JICA	444-4444	1002

Inner Join

order_number integer	order_date date	customer_id integer	name character
1	2006-12-25	103	ICTTI
2	2006-12-25	104	JICA
3	2007-01-10	101	ABC Ltd.
4	2007-01-15	101	ABC Ltd.

Figure 49 – Inner Join SELECT

Outer Join

- Left Outer Join

```
SELECT o.order_number, o.order_date, c.customer_id, c.name
FROM sales.orderhead o LEFT OUTER JOIN sales.customer c
ON o.customer_id = c.customer_id;
```

- Right Outer Join

```
SELECT o.order_number, o.order_date, c.customer_id, c.name
FROM sales.orderhead o RIGHT OUTER JOIN sales.customer c
ON o.customer_id = c.customer_id;
```

- Full Outer Join

```
SELECT o.order_number, o.order_date, c.customer_id, c.name
FROM sales.orderhead o FULL OUTER JOIN sales.customer c
ON o.customer_id = c.customer_id;
```

Note: MySQL 5.0 does NOT support Full Outer Join (It will be supported from 5.1), thus using UNION to unite Left Outer join and Right Outer join is an option to implement Full Outer Join as stated below:

```
SELECT o.order_number, o.order_date, c.customer_id, c.name
FROM sales.orderhead o LEFT OUTER JOIN sales.customer c
ON o.customer_id = c.customer_id
UNION
SELECT o.order_number, o.order_date, c.customer_id, c.name
```

```
FROM sales.orderhead o RIGHT OUTER JOIN sales.customer c
ON o.customer_id = c.customer_id;
```

Order (sales.orderhead)

order_number integer	order_date date	customer_id integer
1	2006-12-25	103
2	2006-12-25	104
3	2007-01-10	101
4	2007-01-15	101
5	2007-01-21	999

Customer (sales.customer)

customer_id integer	name character	tel character	employee_id integer
101	ABC Ltd.		1001
102	XYZ Ltd.	222-2222	1001
103	ICTTI	333-3333	1002
104	JICA	444-4444	1002

Left Outer Join

order_number integer	order_date date	customer_id integer	name character
1	2006-12-25	103	ICTTI
2	2006-12-25	104	JICA
3	2007-01-10	101	ABC Ltd.
4	2007-01-15	101	ABC Ltd.
5	2007-01-21		

Right Outer Join

order_number integer	order_date date	customer_id integer	name character
4	2007-01-15	101	ABC Ltd.
3	2007-01-10	101	ABC Ltd.
		102	XYZ Ltd.
1	2006-12-25	103	ICTTI
2	2006-12-25	104	JICA

Full Outer Join

order_number integer	order_date date	customer_id integer	name character
4	2007-01-15	101	ABC Ltd.
3	2007-01-10	101	ABC Ltd.
		102	XYZ Ltd.
1	2006-12-25	103	ICTTI
2	2006-12-25	104	JICA
5	2007-01-21		

Figure 50 – Inner Join SELECT

- Sub-query

Sub-query is a query that is embedded (nested) inside another query. The inner query is always executed first and returns the result to main query.

Syntax:

```
SELECT column list
FROM table list
WHERE condition (SELECT column FROM table WHERE condition);
```

Examples:

Returns list of order taken by Employee (1002) based on the list of customer_id belongs to:

```
SELECT * FROM orderhead WHERE customer_id
IN (SELECT customer_id FROM customer WHERE employee_id = 1002);
```

5.2.2 INSERT

INSERT inserts one or more rows into a table.

Syntax:

```
INSERT INTO table name VALUES (value1, value2, ,,,value n);
INSERT INTO table name SELECT value1, value2, ,,, value n)
FROM table name;
```

Example:

Insert a customer into customer table:

```
INSERT INTO sales.customer(customer_id, name, tel, employee_id) VALUES(101,
'ABC Ltd.', '111-1111', 1001);
```

5.2.3 UPDATE

UPDATE modifies an attribute's values in one or more table's row.

Syntax:

```
UDATE table name SET column = expression
[WHERE condition list];
```

Example:

Update the telephone number of customer table as NULL:

```
UPDATE customer SET tel = NULL;
```

Update the telephone number of customer (104) in customer table:

```
UPDATE customer SET tel = '999-9999' WHERE customer_id = 104;
```

5.2.4 DELETE

DELETE deletes one or more rows from a table.

Syntax:

```
DELETE FROM table name [WHERE condition list];
```

Example:

Delete all rows from customer table:

```
DELETE FROM customer;
```

Delete a customer (104) from customer table:

```
DELETE FROM customer WHERE customer_id = 104;
```

Exercise 7-1. - DML practice with PostgreSQL and MySQL

SELECT

Practice SELECT statements in 5.2.1.

INSERT Statements

Practice INSERT statements in 5.2.2.

UPDATE Statements

Practice UPDATE statements in 5.2.3.

DELTE Statements

Practice DELETE statements in 5.2.4.

SQL Quiz

- Total quantity of stock by Product
- Total sales by Product
- Total sales by Customer
- Total sales by Employee
- Monthly sales by Product
- Monthly sales by Customer
- Monthly sales (Total)

Exercise 7-2. - Create the five tables that form the sample database

```
CREATE TABLE PLAYERS(
PLAYERNO        INTEGER         NOT NULL,
NAME            CHAR(15)        NOT NULL,
INITIALS        CHAR(3)         NOT NULL,
BIRTH_DATE      DATE,
SEX             CHAR(1)         NOT NULL,
JOINED          SMALLINT        NOT NULL,
STREET          VARCHAR(30)     NOT NULL,
HOUSENO         CHAR(4),
POSTCODE        CHAR(6),
```

```
TOWN            VARCHAR(30)   NOT NULL,
PHONENO         CHAR(13),
LEAGUENO        CHAR(4),
PRIMARY KEY     (PLAYERNO));

CREATE TABLE TEAMS(
TEAMNO          INTEGER       NOT NULL,
PLAYERNO        INTEGER       NOT NULL,
DIVISION        CHAR(6)       NOT NULL,
PRIMARY KEY     (TEAMNO));

CREATE TABLE MATCHES(
MATCHNO     INTEGER       NOT NULL,
TEAMNO      INTEGER       NOT NULL,
PLAYERNO    INTEGER       NOT NULL,
WON         SMALLINT      NOT NULL,
LOST        SMALLINT      NOT NULL,
PRIMARY KEY (MATCHNO));

CREATE TABLE PENALTIES(
PAYMENTNO   INTEGER       NOT NULL,
PLAYERNO    INTEGER       NOT NULL,
PAYMENT_DATE DATE         NOT NULL,
AMOUNT      DECIMAL(7,2)  NOT NULL,
PRIMARY KEY (PAYMENTNO));

CREATE TABLE COMMITTEE_MEMBERS(
PLAYERNO    INTEGER       NOT NULL,
BEGIN_DATE  DATE,
END_DATE    DATE,
POSITION    CHAR(20),
PRIMARY KEY (PLAYERNO, BEGIN_DATE));
```

1. Fill all tables from the sample database with data.

INSERT INTO PLAYERS VALUES (6, ‘Parmenter’, ’R’, ‘1964-06-25’, ‘M’, 1977, ‘Haseltine Lane’, ‘80’, ‘1234KK’, ‘Stratford’, ‘070-47865’, ‘8467’);

INSERT INTO PLAYERS VALUES (7, ‘Wise’, ’GWS’, ‘1963-05-11’, ‘M’, 1981, ‘Edgecombe

Way', '39', '9758VB', 'Stratford', '070-34769', NULL);

INSERT INTO TEAMS VALUES (1, 6, 'first');

INSERT INTO TEAMS VALUES (2, 27, 'second');

INSERT INTO MATCHES VALUES (1, 1, 6, 3, 1);

INSERT INTO MATCHES VALUES (4, 1, 44, 3, 2);

INSERT INTO PENALTIES VALUES (1, 6, '1980-12-08', 100);

INSERT INTO PENALTIES VALUES (2, 6, '1981-05-05', 75);

INSERT INTO COMMITTEE_MEMBERS VALUES (6, '1990-01-01', '1990-12-31', 'Secretary');

INSERT INTO COMMITTEE_MEMBERS VALUES (6, '1991-01-01', '1992-12-31', 'Member');

2. Get the number, name and date of birth of each player resident in Straford, sort the result in alphabetical order of name.
3. Get the number of each player who joined the club after 1980 and is resident in Straford, order the result by player number.
4. Get all the information about each penalty.
5. Change the amount of each penalty incurred by player 44 to $200.
6. Remove each penalty with an amount greater than $100.
7. Create an index on the AMOUNT column of the PENALTIES tables.
8. Create a view in which the difference between the number of sets won and the number of sets lost are recorded for each match.
9. What is the most recent version of the MySQL database server that we use now?

Note:

Run following SQL in order to prepare for Join SELECT practice:

```
-- Drop Foreign Key Constraint in the orderhead table:
ALTER TABLE sales.orderhead DROP CONSTRAINT fk_orderhead_1;
-- Insert an order without customer_id in the customer table:
INSERT INTO sales.orderhead VALUES (5, '2007-01-21', 999);
```

6 Transaction Management and Concurrency Control

6.1 Transaction Management

6.1.1 ACID Properties

ACID properties which consist of Atomicity, Consistency, Isolation and Durability are one of the most important concepts of database transaction management. They are considered to be the key transaction processing features or properties of DBMS in order to guarantee the integrity of the database. The DBMS implements functions such as commitment control, concurrency control and disaster recovery control to support the ACID properties as shown in the table below:

Table 39 - ACID Properties

ACID Property	Description	DBMS Function
Atomicity	**Atomicity** states that database modifications must follow an "all or nothing" rule. Each transaction is said to be "atomic." If one part of the transaction fails, the entire transaction fails.	Commitment Control
Consistency	**Consistency** states that only valid data will be written to the database. If a transaction is executed that violates the consistency rules, the entire transaction will be rolled back and the database will be restored.	Concurrency Control
Isolation	**Isolation** requires that multiple transactions occurring at the same time not impact each other's execution.	
Durability	**Durability** ensures that any transaction committed to the database will not be lost.	Disaster Recovery

6.1.2 Isolation Levels

The ANSI/ISO SQL standard defines four levels of transaction isolation as stated in table above. The standard defines these levels of isolation in terms of four phenomena that must be prevented between concurrent transactions; DBMS must provide functions to prevent any inconsistencies to be happened. The four undesirable phenomena that must be prevented are listed in the table below.

Table 40 - Undesirable phenomena for Transaction

Phenomenon	Description
Dirty Read	**Dirty Reads** occurs when a transaction reads data written by concurrent uncommitted transaction.
Non-Repeatable Read	**Non-Repeatable Read** occurs when a transaction re-reads data it has previously read and finds that data has been modified by another transaction (that committed since the initial read).
Phantom Read	**Phantom Read** occurs when a transaction re-executes a query returning a set of rows that satisfy a search condition and finds that the set of rows satisfying the condition has changed due to another recently-committed transaction.
Lost Update	**Lost Update** occurs when two or more users try to update the same data at the same time

- Multi-version Concurrency Control (MVCC).

Multi-version Concurrency Control (MVCC) is a concurrency control method commonly used by DBMS to provide concurrent access to the database. In PostgreSQL (MySQL follows similar architecture hence you can read this by replacing with MySQL), MVCC is implemented by increasing transaction IDs to achieve serializability. As can be seen Figure below, User A begins a transaction to update price for Pencil to 200. PostgreSQL actually insert a new row in the physical table and increase the transaction ID. On the other hand, User B still views price of Pencil for 150, until User A commits the transaction:

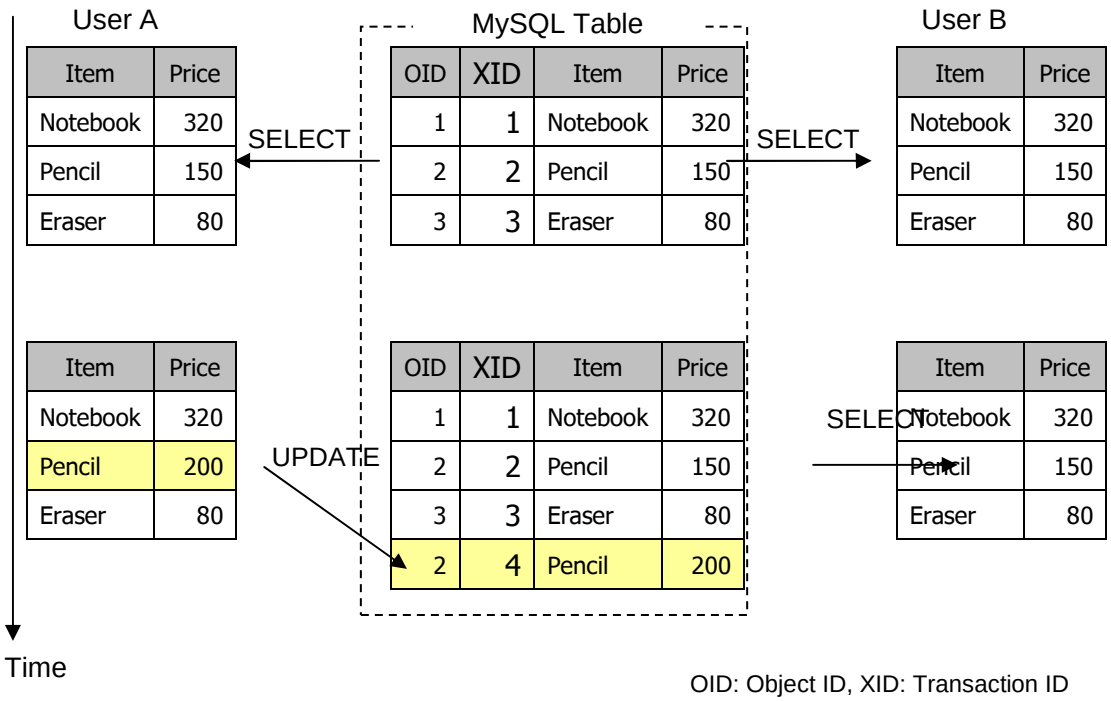


Figure 51 - MVCC image by PostgreSQL

- Dirty Read

A dirty read occurs when a transaction reads data that is being modified by another transaction that has not yet committed. For example, updated record before committed by user A will be able to read by user B:

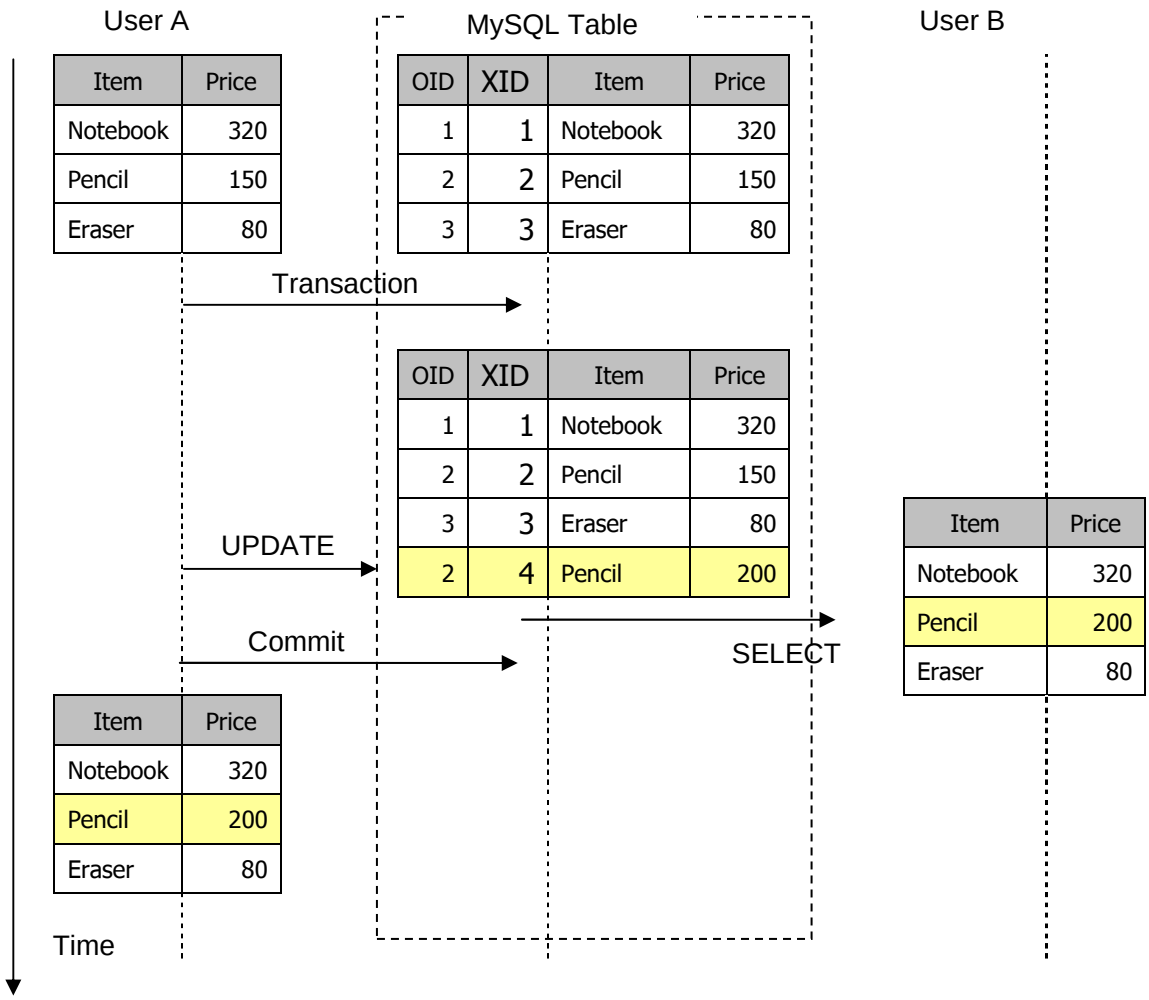


Figure 52 - Dirty Read

As for the update operation, PostgreSQL updates a record, it keeps the original copy of the record in the physical table and writes (insert) a new one. The original record, marked as expired, is used by other transactions still viewing the database in its prior state. Deletions of the records are just marked as expired, but those records are not removed from the physical table at this point of time.

Note: OID: Object ID, XID: Transaction ID

- Non-Repeatable Read

Non-repeatable reads occurs when a SELECT returns data that would be different even though the SELECT were repeated within the same transaction. It can occur when other transactions

are modifying data that a transaction is reading as follows.

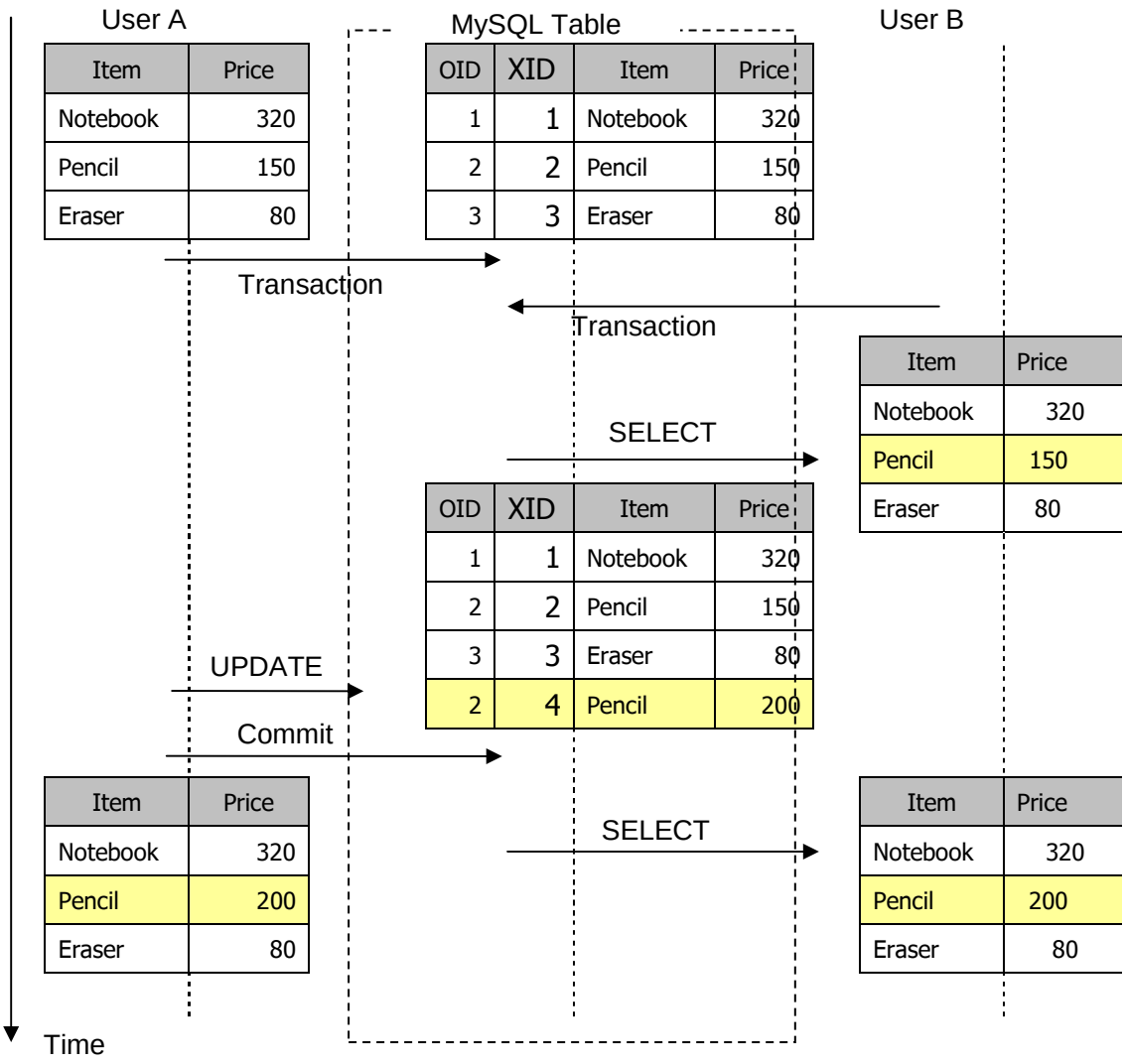
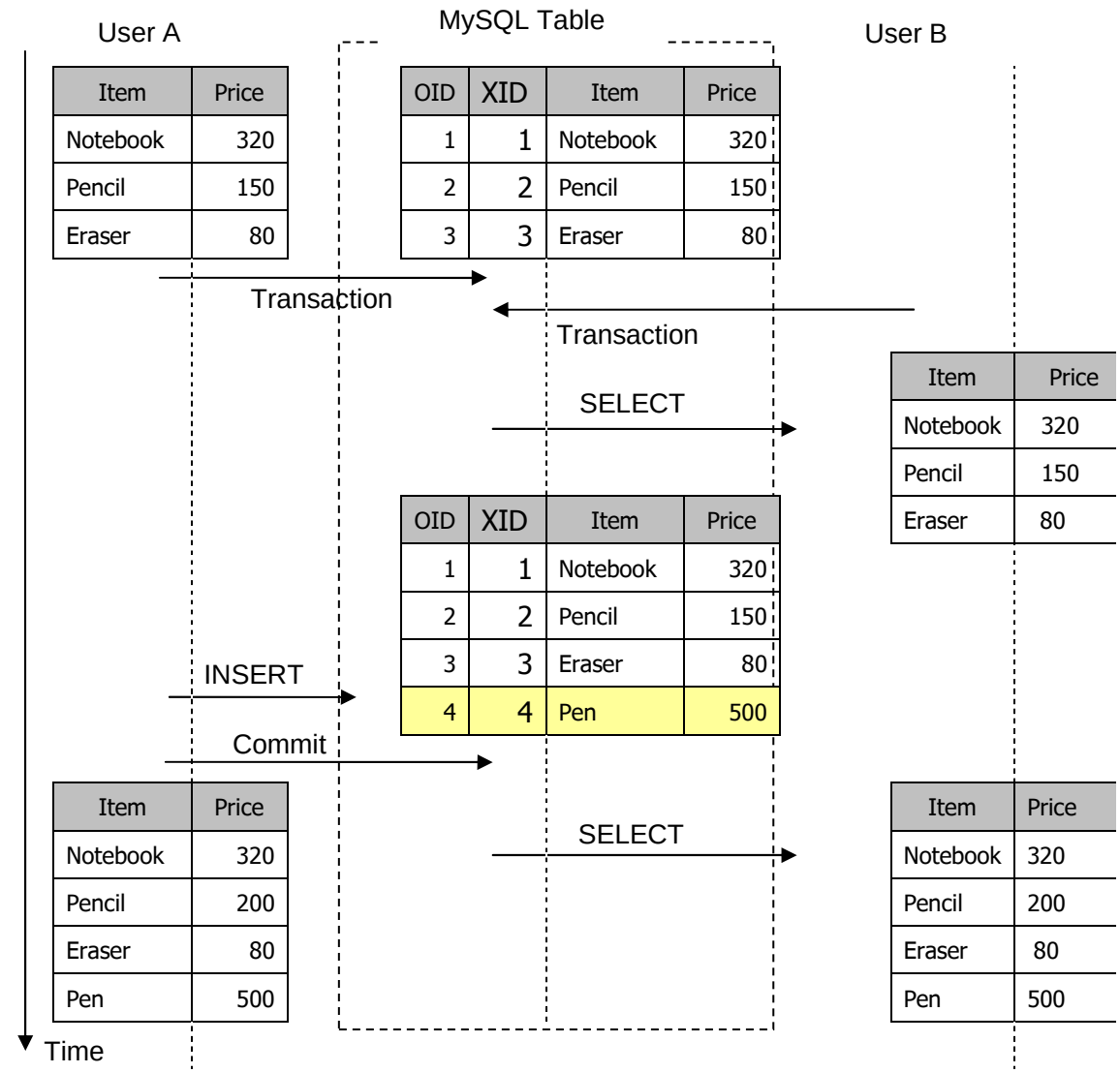


Figure 53 - Non-Repeatable Read

- Phantom Read

Phantom reads can occur when other transactions insert rows that would satisfy the WHERE clause of another transaction's (User B) statement.



`

Figure 54 - Phantom Read

- Lost Update

Lost Update occurs when a SELECT returns data that would be different even though the SELECT were repeated within the same transaction. The first transaction has disappeared when other transactions are overwritten data that a transaction is reading as follows.

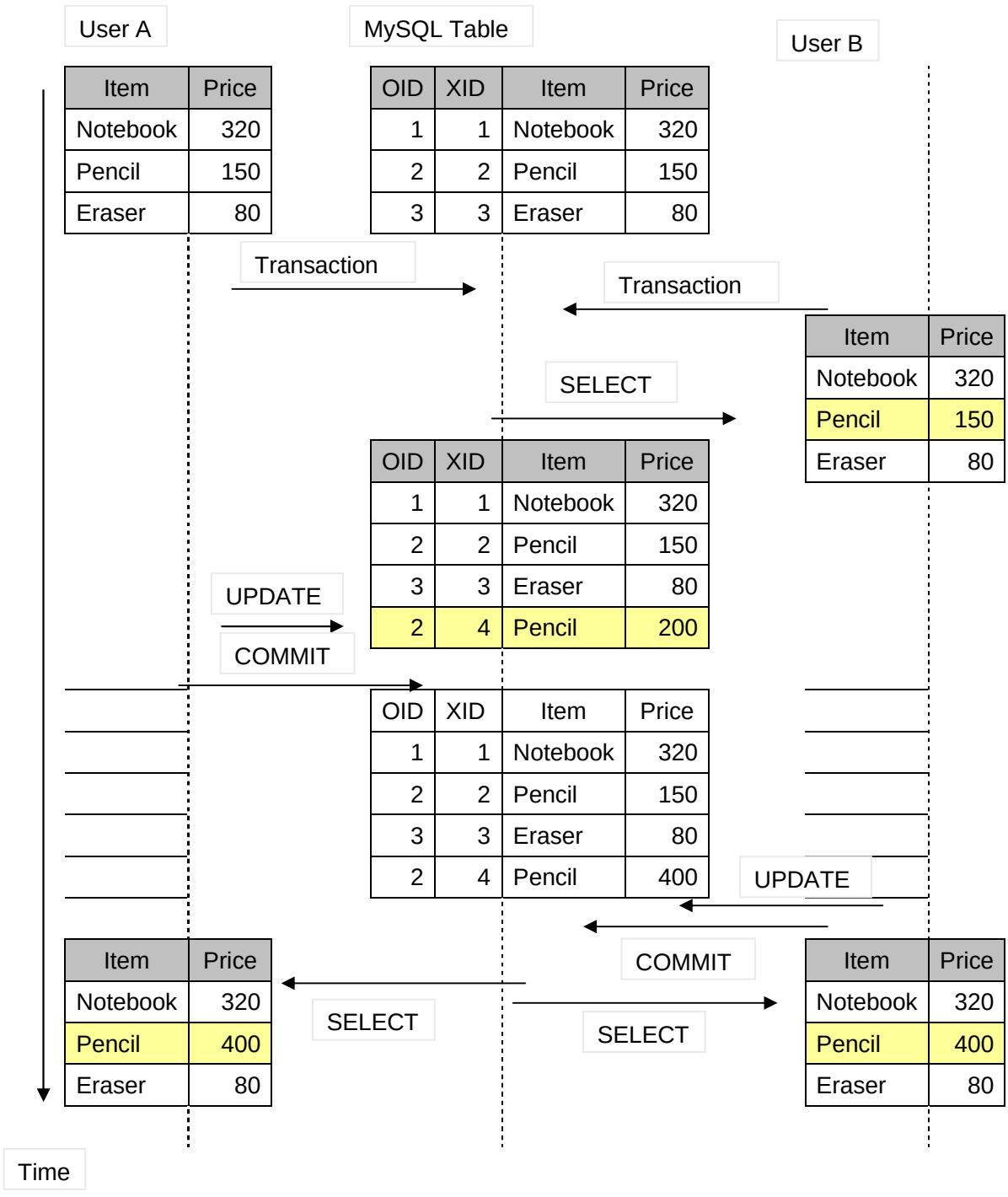


Figure 55 – Lost Update

Table 41 – Transaction Isolation Levels

Level	Dirty Read	Non-Repeatable Read	Phantom Read	PostgreSQL	MySQL
READ UNCOMMITED	Occurs	Occurs	Occurs		Supports
READ COMMITED	Prevents	Occurs	Occurs	**Supports**	Supports
REPEATABLE READ	Prevents	Prevents	Occurs		**Supports**
SERIALIZABLE	Prevents	Prevents	Prevents	Supports	Supports

READ COMMITED is default setting for PostgreSQL and REPEATABLE READ for MySQL default setting.

6.1.3 Transaction Management with Data Control Language (DCL)

SQL supports database transaction with following statements such as BEGIN, COMMIT and ROLLBACK transactions:

- BEGIN, START TRANSACTION

BEGIN or START TRANSACTION starts a transaction block.

Syntax:

```
BEGIN;
Or
START TRANSACTION;
```

- COMMIT

COMMIT confirms the transaction. It means COMMIT permanently save any changes since BEGIN or START TRANSACTION command.

Syntax:

```
COMMIT;
```

- ROLLBACK

ROLLBAK cancels the transaction. It means that ROLLBACK restores all the changes since BEGIN or START TRANSACTION command

Syntax:

```
ROLLBACK;
```

Exercise 8. - Transaction Management practice with MySQL

Objective of this exercise is to practice basics of transaction control commands and DCL using MySQL.

1. BEGIN, COMMIT commands

Create a table and insert records for the preparation of this exercise

```
>mysql -u root -p -D s_ct_d
mysql> CREATE TABLE account(id INTEGER PRIMARY KEY, balance INTEGER)engine=Innodb;
mysql> INSERT INTO account VALUES(1001, 1000);
mysql> INSERT INTO account VALUES(1002, 2000);
```

Practice BEGIN and COMMIT with example of account transfer from 1001 to 1002

```
mysql> SELECT * FROM account;
mysql> BEGIN;
mysql> UPDATE account SET balance = balance - 200 WHERE id = 1001;
mysql> UPDATE account SET balance = balance + 200 WHERE id = 1002;
(Open another mysql Terminal to execute 'SELECT * FROM account;' What are the data in account table?)
mysql> COMMIT;
(Execute 'SELECT * FROM account;' from another mysql again to see what are the changes and discuss with your neighbors why?)
mysql> SELECT * FROM account;
```

I

ROLLBACK command

Practice ROLLBACK with example of account transfer from 1001 to 1002

```
mysql> SELECT * FROM account;
mysql> BEGIN;
mysql> UPDATE account SET balance = balance - 200 WHERE id = 1001;
mysql> UPDATE account SET balance = balance + 200 WHERE id = 1002;
mysql> SELECT * FROM account;
mysql> ROLLBACK;
mysql> SELECT * FROM account;
```

Check Transaction Isolation Level (REPERTABLE-READ as default)

Check MySQL default transaction isolation level with a variable called 'tx_isolation'

```
mysql> show variables like 'tx_isolation';
+---------------+----------------+
| Variable_name | Value          |
+---------------+----------------+
```

```
| tx_isolation  | REPEATABLE-READ |
+---------------+-----------------+
```

Change Transaction Isolation Level

Practice to change transaction isolation level using 'SET TRANSACTION' command

```
mysql> SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
mysql> show variables like 'tx_isolation';
+---------------+------------------+
| Variable_name | Value            |
+---------------+------------------+
| tx_isolation  | READ-UNCOMMITTED |
+---------------+------------------+

mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
mysql> show variables like 'tx_isolation';
+---------------+----------------+
| Variable_name | Value          |
+---------------+----------------+
| tx_isolation  | READ-COMMITTED |
+---------------+----------------+

mysql> SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
mysql> show variables like 'tx_isolation';
+---------------+-----------------+
| Variable_name | Value           |
+---------------+-----------------+
| tx_isolation  | REPEATABLE-READ |
+---------------+-----------------+

mysql> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
mysql> show variables like 'tx_isolation';
+---------------+--------------+
| Variable_name | Value        |
+---------------+--------------+
| tx_isolation  | SERIALIZABLE |
+---------------+--------------+
```

6.2 Concurrency Control

6.2.1 Record-level/Table-level Locks

- Record-level Lock

Record-level Lock locks selected records in the table using FOR UPDATE statement.

Syntax:

```
SELECT FOR UPDATE
```

- Table-level Lock

Table-Level lock locks entire table and mode of the table lock for PostgreSQL is defined as table below:

Syntax:

```
LOCK TABLE table_name [IN lock_mode MODE][NOWAIT]
```

Table 42 - PostgreSQL Table Lock Mode

Level	Lock Mode	SQL	Conflict with
1	**A**CCESS **S**HARE (AS)	SELECT, ANALYZE	AE (Access Exclusive)
2	**R**OW **S**HARE (RS)	SELECT FOR UPDATE	E, AE
3	**R**OW **E**XCLUSIVE (RE)	UPDATE, DELETE, INSERT	S, SRE, E, AE
4	**S**HARE **U**PDATE **E**XCLUSIVE (SUE)	VACUUM	SUE, S, SRE, E, AE
5	**S**HARE (S)	CREATE INDEX	RE, SUE, SRE, E, AE
6	**S**HARE **R**OW **E**XCLUSIVE (SRE)	LOCK	RE, SUE, S, SRE, E, AE
7	**E**XCLUSIVE (E)	LOCK	RS, RE, SUE, S, SRE, E, AE
8	**A**CCESS **E**XCLUSIVE (AE)	VACUUM FULL, ALTER TABLE, DROP TABLE REINDEX, CLUSTER	All

6.2.2 Shared/Exclusive Locks

Locking of tables also can be implemented by setting transaction isolation levels.

6.2.3 Dead Locks

A dead lock is the situation that two transactions lock two database objects each other and as a result all the database activities are stopped. PostgreSQL detects dead lock automatically.

Timeout for dead lock can be configured deadlock_timeout parameter in postgresql.conf file.

Exercise 9. Concurrency Control practice with MySQL

Objective of this exercise is to practice to understand how transaction isolation level setting prevents undesirable phenomena for transaction.

Preparation (This practice is based on train seat reservation by concurrent user access)

Create following table and insert records:

```
CREATE TABLE seat (id integer PRIMARY KEY, reservedby text)engine=Innodb;
INSERT INTO seat VALUES(1, 'Lwin');
INSERT INTO seat VALUES(2, null);
INSERT INTO seat VALUES(3, 'Khine');
```

READ UNCOMMITED concurrency control

Run two mysql clients and execute following SQL to practice concurrent access

Table 43 - READ UNCOMMITED

Step	mysql A	mysql B
1	-- Set Isolation level and start transaction SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED; START TRANSACTION; -- Check availability of seat SELECT * FROM seat WHERE id = 2;	
2		-- Set Isolation level and start transaction SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED; START TRANSACTION; -- Check availability of seat SELECT * FROM seat WHERE id = 2;
3	-- Reserve a seat UPDATE seat SET reservedby = ‘Ono’ WHERE id = 2;	
4		-- Check availability of seat SELECT * FROM seat WHERE id = 2;
5	-- Commit Transaction COMMIT;	

Note: Discuss with your neighbor to understand what was happened (which undesirable phenomena for transaction was happened) in this transaction.

READ COMMITED concurrency control

Run two mysql clients and execute following SQL to practice concurrent access

Table 44 - READ COMMITED

Step	mysql A	mysql B
1	-- Set Isolation level and start transaction SET TRANSACTION ISOLATION LEVEL READ COMMITTED; START TRANSACTION; -- Check availability of seat SELECT * FROM seat WHERE id = 2;	
2		-- Set Isolation level and start transaction SET TRANSACTION ISOLATION LEVEL READ COMMITTED; START TRANSACTION; -- Check availability of seat SELECT * FROM seat WHERE id = 2;
3	-- Reserve a seat UPDATE seat SET reservedby = 'Ono' WHERE id = 2; -- Commit Transaction COMMIT;	
4		-- Reserve a seat UPDATE seat SET reservedby = 'Tamaki' WHERE id = 2; -- Commit Transaction COMMIT;

Note: Discuss with your neighbor to understand what was happened (which undesirable phenomena for transaction was happened) in this transaction.

SERIALIZABLE concurrency control

Run two mysql clients and execute following SQL to practice concurrent access

Table 45 - SERERIALIZABLE

Step	mysql A	mysql B
1	-- Set Isolation level and start transaction SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	

	START TRANSACTION; -- Check availability of seat SELECT * FROM seat;	
2		-- Set Isolation level and start transaction SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; START TRANSACTION; -- Check availability of seat; SELECT * FROM seat;
3	-- Reserve a seat UPDATE seat SET reservedby = 'Ono' WHERE id = 2; -- Commit Transaction COMMIT;	
4		-- Reserve a seat UPDATE seat SET reservedby = 'Tamaki' WHERE id = 2; -- Commit Transaction COMMIT;

Note: Discuss with your neighbor to understand what was happened (which undesirable phenomena for transaction was happened) in this transaction.

Record level lock using FOR UPDATE

Run two mysql clients and execute following SQL to practice concurrent access

Table 46 - FOR UPDATE

Step	mysql A	mysql B
1	-- Check availability of seat SELECT * FROM seat; -- Start transaction BEGIN; SELECT * FROM seat WHERE id = 2 FOR UPDATE;	
2		-- Set Isolation level and start transaction BEGIN; SELECT * FROM seat WHERE id = 2 FOR UPDATE;
3	-- Reserve a seat UPDATE seat SET reservedby = 'Ono'	

	WHERE id = 2; -- Commit Transaction COMMIT;	
4		-- Reserve a seat UPDATE seat SET reservedby = ‘Ono’ WHERE id = 2; -- Commit Transaction COMMIT;

Table level lock using LOCK TABLE and release those lock using UNLOCK TABLES

Run two mysql clients and execute following SQL to practice concurrent access

Table 47 - LOCK TABLE and UNLOCK TABLES

Step	mysql A	mysql B
1	-- Add a column for this practice ALTER TABLE seat ADD seat_id int; -- Check availability of seat SELECT * FROM seat; -- Start transaction with table level lock BEGIN; LOCK TABLE seat WRITE;	
2		-- Start transaction with table level lock BEGIN; LOCK TABLE seat WRITE; -- Table seat is blocked by another user
3	-- Insert a new record and commit SELECT * FROM seat; INSERT INTO seat VALUES(4, 'Sasahara', 2); COMMIT; UNLOCK TABLES;	
4		-- Now the table seat is locked by mysql B -- Seat has already taken by another user SELECT * FROM seat WHERE seat_id = 2; -- Some DML here COMMIT;

		UNLOCK TABLES;

7 Distributed Database Management System

A distributed database is a database that is under the control of a central database management system (DBMS) in which portions of the database are stored in multiple computers located in the same physical location, or dispersed over a network of interconnected computers.

7.1 Distributed Database Features

A distributed database system requires functional characteristics that the database end users believe that they are working with a centralized DBMS (virtual view of data set); all complexities of a distributed database should be hidden, in other words transparent from them and provides following features:

7.1.1 Distribution Transparency

Users must be able to interact with the system as if it was one logical system; the users do not need to know that the data are partitioned or the data can be replicated at several sites or different locations. This concept applies to the systems performance and methods of access amongst other things as well. In order to maintain the same level of performance, system scale-up or scale-out such as database load balancing will be implemented.

7.1.2 Transaction Transparency

Each transaction must maintain database integrity across multiple databases. Transactions must also be divided into sub transactions, each sub transaction affecting one database system. This transparency will be implemented by distributed concurrency control or two-phase commit protocol.

7.1.3 Failure Transparency

System must be bale to continue to operate in the event of a node failure. Event of the failure will be picked up by another node in the network. This transparency will be implemented by database clustering or replication technologies.

7.2 Database Replication

Database replication refers to the storage of data copies at a multiple sites served by either physical storages or a computer network. Fragment copies can be stored at several sites to serve specific information requirements. Because the existence of fragment copies can enhance data availability and response time, data copies can help to reduce communication and total query costs.

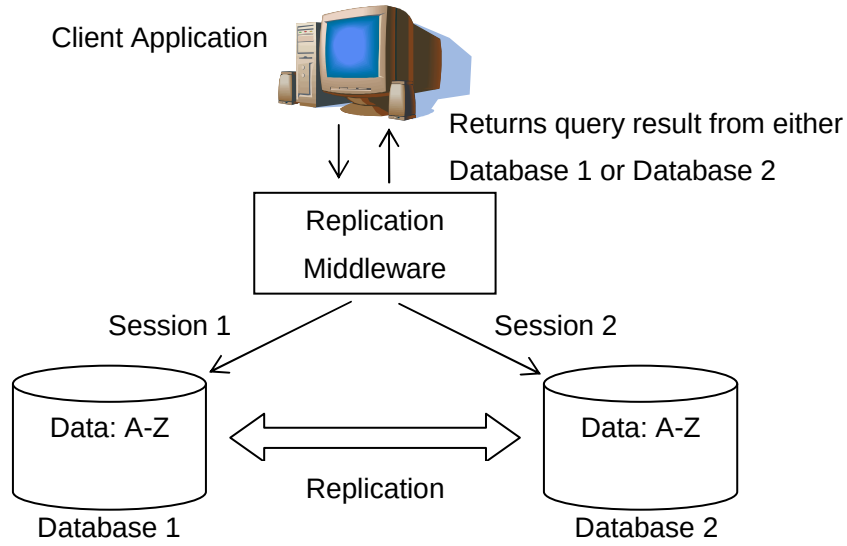


Figure 55 - Database Replication

7.3 Database Partitioning

7.3.1 Data Fragmentation

Data fragmentation allows breaking a single database object such as table into two or more segment of fragment. Each fragment can be stored at any site or storage either local or over a network for the sake of better database performance especially for enterprise scale of databases. This is because the distribution of physical disk access will reduce the bottleneck of performances. There are two types of fragmentations such as:

- Horizontal fragmentation

Horizontal fragmentation refers to the division of a table into fragments of rows. Each fragment is stored at a different node, and each fragment has unique rows.

- Vertical fragmentation

Vertical fragmentation refers to the division of a table into attribute subsets. Each subset (fragment) is stored at a different node, and each fragment has unique columns.

7.3.2 Parallel Query

A parallel query is a database query based on the technologies of horizontal fragmentation to partitioning to distinct rows into several tables as illustrated in two figures below:

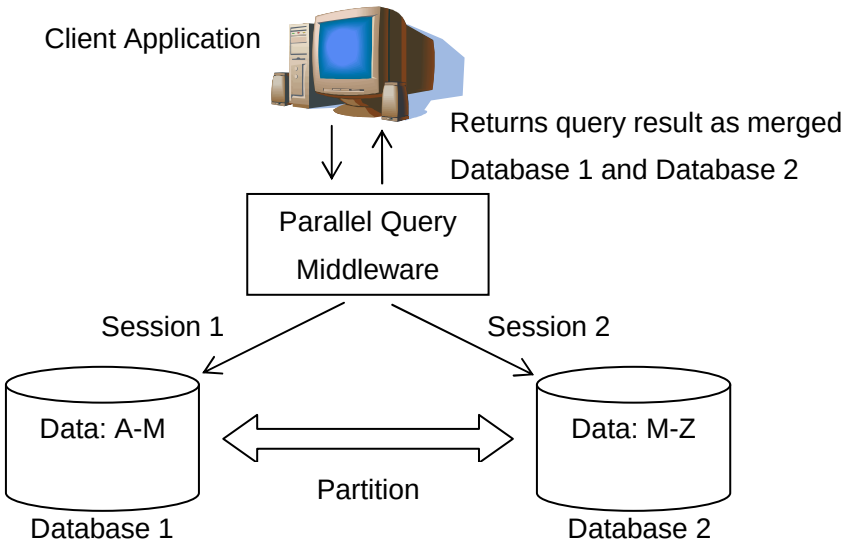


Figure 56 - Parallel Query

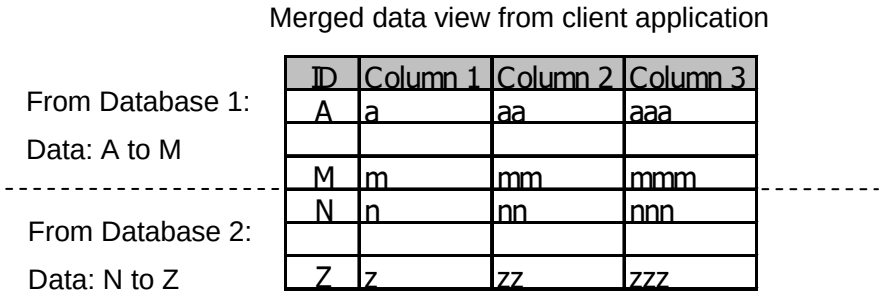


Figure 57 - Horizontal Fragmentation by Parallel Query

7.4 Introduction to MySQL Database Replication

7.4.1 What is MySQL Database Replication?

Database replication is a way of copying database objects and keeping data synchronized (synchronous or asynchronous) in multiple databases over the network. The database replication can be done between master server and slave server(s). It can be used for enterprise applications in order to implement load balancing, implement redundancy of the database system for high-availability and high-reliability. The replication can also be used for database backup and recovery as well, if the slave server is configured as stand-by database server when master server acts as active database server (Active/Stand-by configuration).

MySQL supports for one-way, asynchronous database replication which consists of the master server and the slave servers. This is in contrast to the synchronous replication which is a characteristic of pgpool-II for PostgreSQL which we have learned in "Fundamental Database".

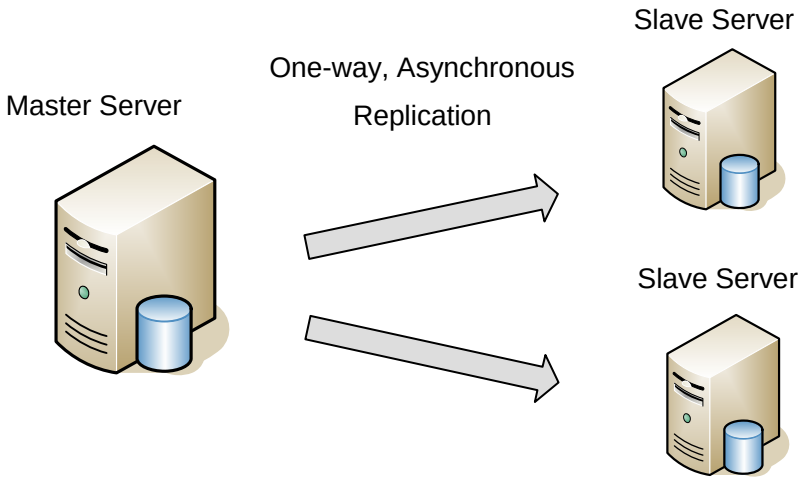


Figure 58 - MySQL One-way, Asynchronous Replication Overview

MySQL replication will be implemented as following steps (as shown in figure above):

1. Slave server to connect Master server using master.info and relay-log.info files
2. Upon successful connection, there are three threads will be created as below:
   - Master Server Thread: Binlog Dump Thread
   - Slave Server Threads: I/O Thread and SQL Thread
3. Master Server to transfer the binary log to the relay log in Slave Server.
4. Slave Server to load the relay log onto Slave Server Database to update (replicate):

Please read MySQL Reference Manual “Chapter 6. Replication” for more details.

For your information, pgpool-II implements synchronous database replication to update two PostgreSQL servers at same timing with Two-Phase Commitment technology as can be seen below. The synchronous replication is usually slower than the asynchronous replication since two nodes must be updated at same time:

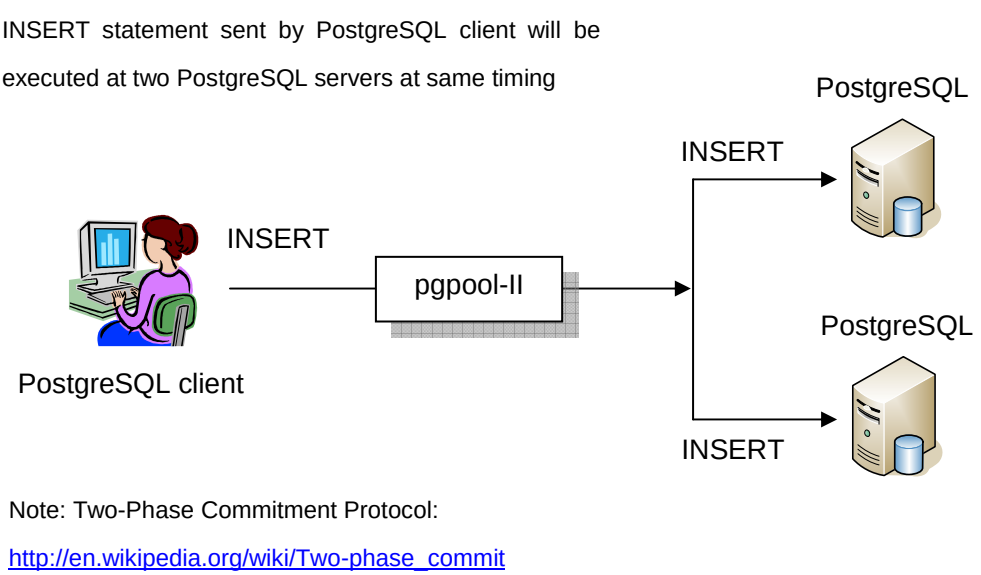


Figure 59 - PostgreSQL with pgpool-II Synchronous Replication Overview

7.4.2 How to Configure MySQL Database Replication?

As for the configuration of MySQL database replication, both the master and the slave servers must be configured. Each server must have unique server-id to identify over the network and binary log need to be activated in order to transfer from the master server to the slave. Followings are step-by-step configuration for the replication:

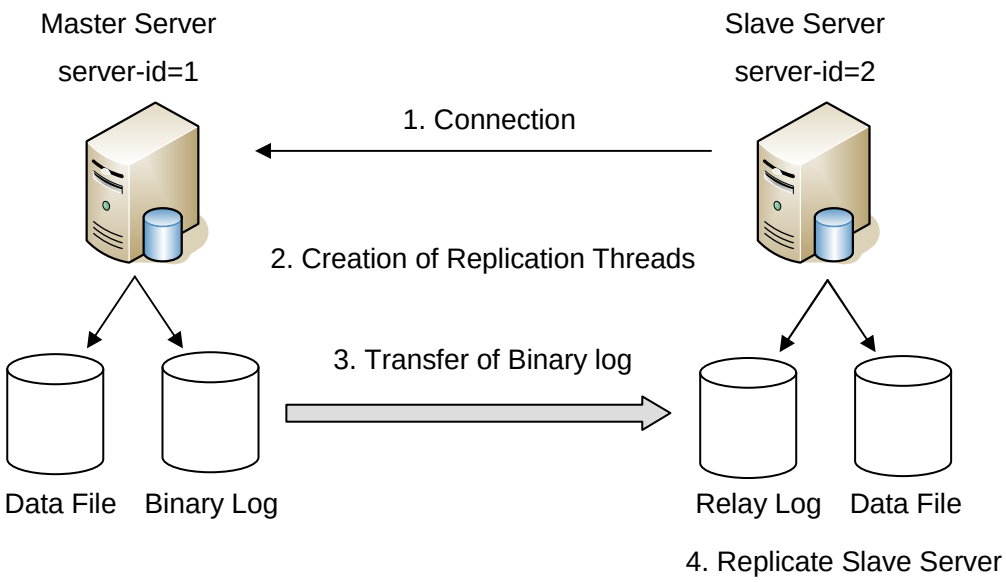


Figure 60 - MySQL Replication Procedure

Configure master server configuration file (/etc/my.cnf)

```
# Master server
[mysqld]
server-id=1
log-bin
```

Both master and slave must have unique server-id

Enable binary log for master server

Create and grant permission for replication user account for the slave server

```
# Master server
mysql> GRANT REPLICATION ON *.* TO replication_user@slave_server IDENTIFIED
BY 'replication_user_password
```

Note:

replication_user : Replication user account name
slave_server : Slave server IP address or hostname
replication_user_password : Password for the replication user

Configure slave server configuration file (/etc/my.cnf)

```
# Slave server

[mysqld]
server-id=2
```

```
log-bin

master-host=master_server
master-user=replication_user
master-password=replication_user_password
mater-port=3306
```

Export the master server database and import it to slave server database

```
# Master server (Export database)
linux terminal> mysqldump –u root –p database_name >db.sql

# Slave server (Create and import database)
linux terminal> mysqladmin –u root –p CREATE database_name
linux terminal> mysql –u root –p database_name < db.sql
```

Restart both the master and slave servers, then start replication

```
# Shutdown and Restart the master MySQL server with mysql tools
# Shutdown and Restart the slave MySQL server with mysql tools
# Start replication from the slave server
# Slave Server
mysql> START SLAVE;

# Monitor master server status at the master server
# Master Server
mysql> SHOW MASTER STATUS;
+----------------+----------+--------------+------------------+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+----------------+----------+--------------+------------------+
| XXX-bin.000003|       98 |              |                  |
+----------------+----------+--------------+------------------+

# Monitor slave server status at the slave server
# Slave Server
mysql> SHOW SLAVE STATUS;
****** 1.row ********
,,,
Slave_IO_Running: Yes
Slave_SQL _Running: Yes
```

,,,

Table below is a list of some of the commonly used parameters for the MySQL replication:

Table 47 - MySQL Replication Parameters

Category	Parameter	Description
Replication	read-only	Disable update from other than slave thread and super user
	replicate-do-table	The slave thread to restrict replication to the specified table
	replicate-ignore-table	The slave thread to ignore replication to the specified table
	replicate-do-db	The slave thread to restrict replication to the specified database
	replicate-ignore-db	The slave thread to ignore replication to the specified database
Communication	master-host	Master server hostname or IP address
	master-port	Master server port number
	master-user	Connection user name to master server
	master-password	Password for connection user to master server
Log	max_relay-log_size	Maximum size of relay log file
	relay-log	Relay log file name

Note: MySQL Log Files, Configuration File (/etc/my.cnf) and variables (parameters) will be discussed in “Database Design and Administration” and “Database Programming”

7.5 Introduction to MySQL Data Partition

7.5.1 What is MySQL Data Partition?

Data partitioning is a way of distribute portions of individual tables across multiple files according to a set of rules. In effect, different portions of a table are stored as separate tables in different locations. MySQL 5.1 supports *horizontal partitioning*. That is, different rows of a table may be assigned to different partitions. MySQL 5.1 does not support *vertical partitioning*, in which different columns of a table are assigned to different partitions. The user-selected rule by which the division of data is accomplished is known as a *partitioning function*, which in MySQL can be the modulus, simple matching against a set of ranges or value lists, an internal hashing function, or a linear hashing function. The function is selected according to the partitioning type specified by the user, and takes as its parameter the value a user-supplied expression. This

expression can be either an integer column, or a function acting on one or more column values and returning an integer. The value of this expression is passed to the partitioning function, which returns an integer value representing the number of the partition in which that particular record should be stored.

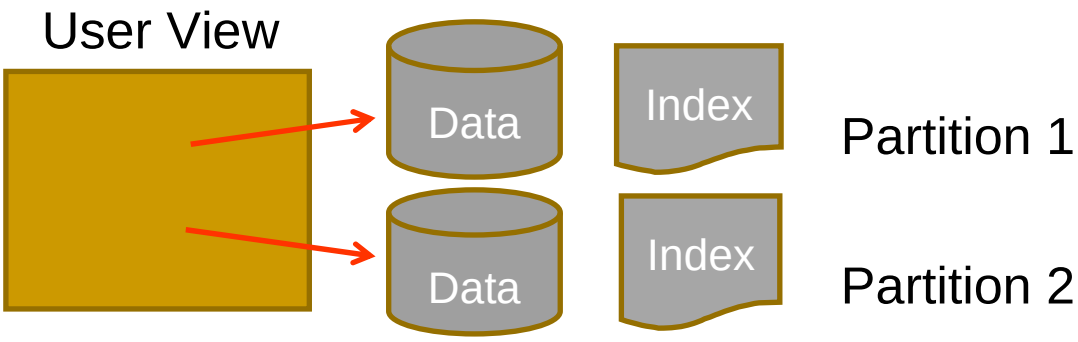


Figure 61 - MySQL Data Parititioning Overview

To determine My SQL Server supports partitioning

```
mysql> SHOW VARIABLES LIKE '%partition%';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| have_partitioning | YES   |
+-------------------+-------+
1 row in set (0.00 sec)
```

To check the status of partition

```
mysql> SHOW PLUGINS;
+------------+----------+----------------+---------+---------+
| Name       | Status   | Type           | Library | License |
+------------+----------+----------------+---------+---------+
| binlog     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| partition  | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| ARCHIVE    | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| BLACKHOLE  | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| CSV        | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| FEDERATED  | DISABLED | STORAGE ENGINE | NULL    | GPL     |
| MEMORY     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| InnoDB     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| MRG_MYISAM | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| MyISAM     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| ndbcluster | DISABLED | STORAGE ENGINE | NULL    | GPL     |
+------------+----------+----------------+---------+---------+
11 rows in set (0.00 sec)
```

Ty

- RANGE Partitioning

A table that is partitioned by range is partitioned in such a way that each partition contains rows for which the partitioning expression value lies within a given range.

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN (21)
);
```

LIST partitioning in MySQL is similar to range partitioning in many ways. As in partitioning by RANGE, each partition must be explicitly defined. The chief difference is that, in list partitioning, each partition is defined and selected based on the membership of a column value in one of a s set of value lists, rather than in one of a set of contiguous ranges of values.

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY LIST(store_id) (
    PARTITION pNorth VALUES IN (3,5,6,9,17),
    PARTITION pEast VALUES IN (1,2,10,11,19,20),
    PARTITION pWest VALUES IN (4,12,13,14,18),
    PARTITION pCentral VALUES IN (7,8,15,16)
);
```

- HASH Partitioning

Partitioning by HASH is used primarily to ensure an even distribution of data among a predetermined number of partitions. With range of list partitioning, you specify explicitly into which partition a given column value or set of column values is to be stored. With hash partitioning, you need only specify a column value or expression based on a column value to be hashed and the number of partitions into which the partitioned table is to be divided.

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH( YEAR(hired) )
PARTITIONS 4;
```

- KEY Partitioning

Partitioning by key is similar to partitioning by hash, except that where hash partitioning employs a user-defined expression, the hash function for key partitioning is supplied by the MySQL server.

```
CREATE TABLE k1 (
    id INT NOT NULL primary key,
    name VARCHAR(20)
 )
PARTITION BY KEY()
PARTITIONS 2;
```

- Subpartitioning

Subpartitions is also known as composite partitioning. It is possible to subpartition tables that are partitioned by RANGE or LIST. Subpartitions may use either HASH or KEY partitioning. Each partition must have the same number of subpartitions.

```
CREATE TABLE sp (id INT, purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990) (
            SUBPARTITION s0,
            SUBPARTITION s1
        ),
        PARTITION p1 VALUES LESS THAN (2000) (
            SUBPARTITION s2,
            SUBPARTITION s3
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE (
            SUBPARTITION s4,
            SUBPARTITION s5
        )
    );
```

Exercise 10.1 - Database Replication practice with MySQL

The configuration of this practice is as depicted below. One of the simplest replications, one-way and asynchronous replication practice will be carried out. Before you start this practice, please check and note your PC's IP address using such as "ifconfig" command.

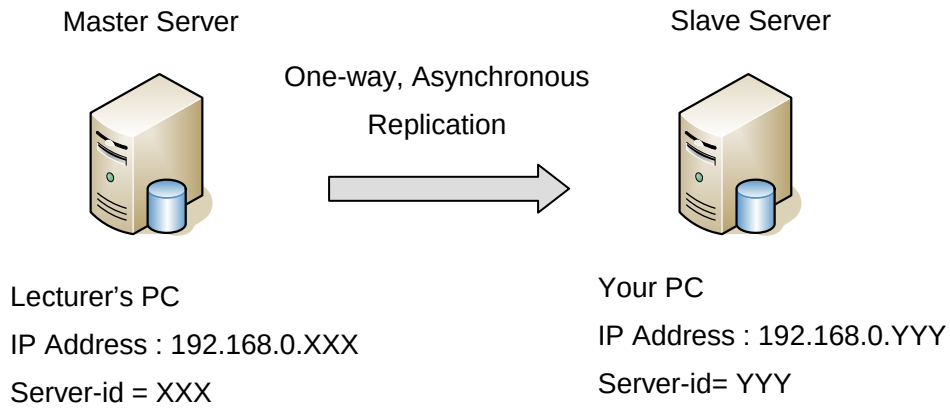


Figure 62 - MySQL Database Replication configuration

2. Preparation (Open SUSE Firewall port 3306 for MySQL Remote Access)

Please check SUSE Firewall setting before start the practice. Port 3306 must be opened for MySQL for this practice.

- YaST -> Select "Security and Users" -> Select "Firewall" -> Click "Allowed Services".
- Select "MySQL Remote Service" and Click "Add".

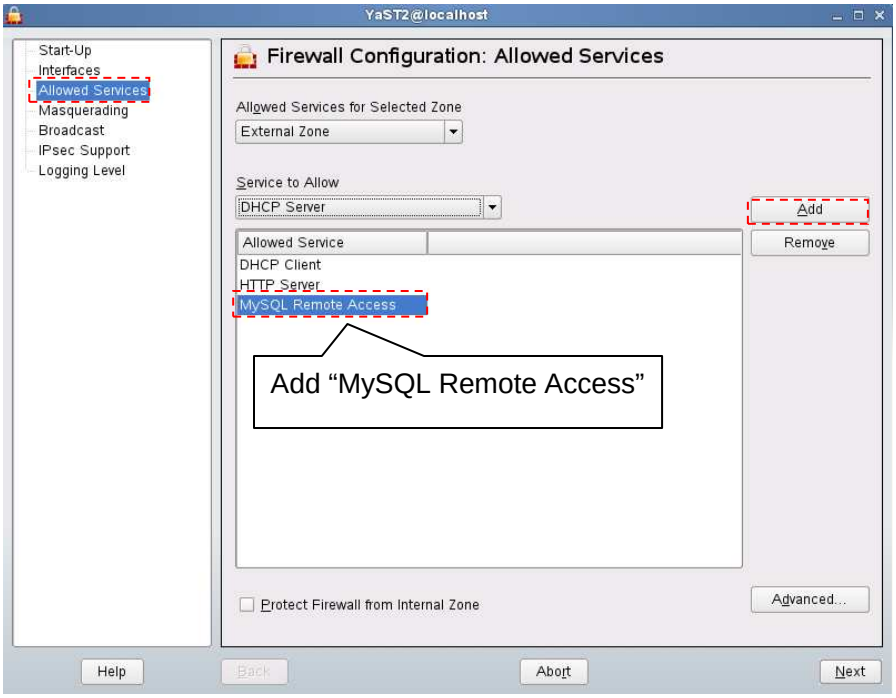


Figure 63 - SUSE Firewall Configuration for MySQL Remote Access

3. Master Server Configuration

- Amend MySQL configuration file (/etc/my.cnf) as follows. Enable binary log and set server-id (This has been done already, so you can skip this time):

```
[mysqld]
log-bin
server-id=1
```

- Remote logon to the master server with mysql command:

```
Linux terminal> mysql –u root –p –h 192.168.0.XXX
Password:****
mysql>
```

Note: You need to have **root@192.168.0.YYY** account in the master server for above remote logon. If you do not have the account, please log on to the master server locally.

- Grant access privilege for user of slave server to access master server as follows:

mysql> GRANT **REPLICATION SLAVE** ON *.* TO repl@192.168.0.**YYY** IDENTIFIED BY 'repl';

Note: **YYY** is your PC's IP address.

mysql> SELECT * FROM **INFORMATION_SCHEMA**.USER_PRIVILEGES WHERE GRANTEE LIKE ```repl%';

Note: SQL above can check status of GRANT.

4. Slave Server Configuration

There are two options to configure the slave server such as amend MySQL configuration file (/ect/my.cnf) or use CHANGE MASTER command as shown in below:

- Option 1: Amend MySQL configuration file (/etc/my.cnf) as follows (To specify the master server information):

 # Slave server (/etc/my.cnf)

[mysqld]
log-bin
server-id=YYY

master-host=192.168.0.XXX
master-user=repl
master-password=repl
master-port=3306

Note:
XXX is last octet of Master server IP address.
YYY can be any unique number. In this practice we will use last octet of your IP address (1 to 254).

- Option2: Use CHANGE MASTER to configure from mysql client as follows:

 # Slave server (/etc/my.cnf)

[mysqld]

log-bin

server-id=YYY

Note:

Add log-bin and server-id parameters only in /etc/my.cnf.

YYY can be any unique number. In this practice we will use last octet of your IP address (1 to 254).

 # Slave server

mysql> STOP SLAVE;

mysql> **CHANGE MASTER** TO MASTER_HOST='192.168.0.XXX', MASTER_PORT=3306, MASTER_USER='repl', MASTER_PASSWORD='repl', MASTER_LOG_FILE='S1-L-bin.000015';

mysql> START SLAVE;

mysql> SHOW SLAVE STATUS\G;

Note: As for MASTER_LOG_FILE, please check the master server status using SHOW MASTER STATUS.

5. Start Replication

- Restart Master MySQL server (This has been done already, so you can skip this time).
- Restart Slave MySQL server.
- Start replication from Slave MySQL server as follows:

 # Slave Server

mysql> START SLAVE;

- Monitor Replication for both Master and Slave servers as follows:

 # Master server

Find out Binlog Dump thread is running

mysql> **SHOW PROCESSLIST\G**;

Find out status of the master server

mysql> **SHOW MASTER STATUS**;

```
+---------------+----------+--------------+------------------+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+---------------+----------+--------------+------------------+
| XXX-bin.000003|       98 |              |                  |
+---------------+----------+--------------+------------------+
```

 # Slave server

mysql> **SHOW SLAVE STATUS\G**;

****** 1.row ********

,,,

Slave_IO_Running: **Yes**

Slave_SQL _Running: **Yes**

,,,

These two status must be “Yes”

Note: \G is mysql option to display result vertically.

Note: If Slave_IO_Running or Slave_SQL _Running status is “No”, there might be something wrong with slave configuration.

6. Test Replication

- Execute some DDL, DML in Master Server:

 # Master Server

```
mysql> CREATE DATABASE test_repl;
mysql> USE test_repl;
mysql> CREATE TABLE test(id INT);
mysql> INSERT INTO test VALUES(9999);
```

- Check replication of above DDL, DML in Slave Server:

```
# Slave Server
# Check database is exits or not
mysql> SHOW DATABASES;
mysql> USE test_repl;

# Check INSERT statement is replicated or not
mysql> SELECT * FROM test;

+--------+
| id     |
+--------+
|   9999 |
+--------+
1 rows in set (0.17 sec)
```

7. Stop Replication

Stop Replication from Slave server.

```
# Slave Server
mysql> STOP SLAVE;
mysql> SHOW SLAVE STATUS\G;
```

Exercise 10-2. - Data Partitioning practice with MySQL

SQL Partition Quiz

- **Partition by Range**

```
mysql>create table students(
              id int not null,
              fname varchar(30),
              lname varchar(30),
              birthday date,
              grade tinyint not null
       )
       partition by RANGE(year(birthday))(
              partition p0 VALUES LESS THAN (1980),
              partition p1 VALUES LESS THAN (1990),
              partition p2 VALUES LESS THAN MAXVALUE
       );
mysql>explain partitions select * from students;
```

- **Partition by List**

```
mysql>create table students(
              id int not null,
              fname varchar(30),
              lname varchar(30),
              birthday date,
              grade tinyint not null
       )
       partition by LIST(month(birthday))(
              partition spring VALUES IN (3,4,5),
              partition summer VALUES IN (6,7,8),
              partition fall VALUES IN (9,10,11),
              partition winter VALUES IN (12,1,2)
       );
mysql>explain partitions select * from students;
```

- **Partition by Hash**

```
mysql>create table students(
              id int not null,
              fname varchar(30),
              lname varchar(30),
              birthday date,
              grade tinyint not null
       )
       partition by HASH(day(birthday))
              partitions 30;
mysql>explain partitions select * from students;
```

- **Partition by Key**

```
mysql> create table students(
              id int not null,
              fname varchar(30),
              lname varchar(30),
              birthday date,
              grade tinyint not null,
              UNIQUE KEY (id)
       )
       partition by KEY()
              partitions 10;
mysql>explain partitions select * from students;
```

- **Subpartition**

```
mysql> create table students(
              id int not null,
              fname varchar(30),
              lname varchar(30),
              birthday date,
              grade tinyint not null,
              PRIMARY KEY (id,birthday)
)
PARTITION BY LIST( MONTH(birthday) )
   SUBPARTITION BY KEY(id) (
       PARTITION spring VALUES IN(3,4,5) (
          SUBPARTITION s0,
          SUBPARTITION s1
       ),
       PARTITION summer VALUES IN(6,7,8) (
          SUBPARTITION s2,
          SUBPARTITION s3
       ),
       PARTITION fall VALUES IN (9,10,11) (
          SUBPARTITION s4,
          SUBPARTITION s5
       ),
        PARTITION winter VALUES IN (12,1,2) (
          SUBPARTITION s6,
          SUBPARTITION s7
       )
   );
Mysql> explain partitions select * from students;
```

Tables and Figures

Figures

Tables

Indexes

Keywords