SQL Overview SQL tutorial gives unique learning on Structured Query Language and it helps to make practice on SQL commands which provides immediate results. SQL is a language of database, it includes database creation, deletion, fetching rows and modifying rows etc. SQL is an ANSI (American National Standards Institute) standard, but there are many different versions of the SQL language. What is SQL? SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database. SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language. Also, they are using different dialects, such as: • MS SQL Server using T-SQL, • Oracle using PL/SQL, • MS Access version of SQL is called JET SQL (native format) etc. Why SQL? • Allows users to access data in relational database management systems. • Allows users to describe the data. • Allows users to define the data in database and manipulate that data. • Allows to embed within other languages using SQL modules, libraries & pre-compilers. • Allows users to create and drop databases and tables. CHAPTER 1 TUTORIALS POINT Simply Easy Learning • Allows users to create view, stored procedure, functions in a database. • Allows users to set permissions on tables, procedures and views History: • 1970 -- Dr. E. F. "Ted" of IBM is known as the father of relational databases. He described a relational model for databases. • 1974 -- Structured Query Language appeared. • 1978 -- IBM worked to develop Codd's ideas and released a product named System/R. • 1986 -- IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software and its later becoming Oracle. SQL Process: When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task. There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc. Classic query engine handles all non-SQL queries, but SQL query engine won't handle logical files. Following is a simple diagram showing SQL Architecture: TUTORIALS POINT Simply Easy Learning SQL Commands: The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their nature: DDL - Data Definition Language: Command Description CREATE Creates a new table, a view of a table, or other object in database ALTER Modifies an existing database object, such as a table. DROP Deletes an entire table, a view of a table or other object in the database. DML - Data Manipulation Language: Command Description INSERT

CUSTOMERS table has already been created, then to add a NOT NULL constraint to SALARY column in Oracle and MySQL, you would write a statement similar to the following: ALTER TABLE CUSTOMERS MODIFY SALARY DECIMAL (18, 2) NOT NULL; TUTORIALS POINT Simply Easy Learning DEFAULT Constraint: The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value. Example: For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, SALARY column is set to 5000.00 by default, so in case INSERT INTO statement does not provide a value for this column. then by default this column would be set to 5000.00. CREATE TABLE CUSTOMERS(ID INT NOT NULL, NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL, ADDRESS CHAR (25) , SALARY DECIMAL (18, 2) DEFAULT 5000.00, PRIMARY KEY (ID)); If CUSTOMERS table has already been created, then to add a DFAULT constraint to SALARY column, you would write a statement similar to the following: ALTER TABLE CUSTOMERS MODIFY SALARY DECIMAL (18, 2) DEFAULT 5000.00; Drop Default Constraint: To drop a DEFAULT constraint, use the following SQL: ALTER TABLE CUSTOMERS ALTER COLUMN SALARY DROP DEFAULT; UNIQUE Constraint: The UNIQUE Constraint prevents two records from having identical values in a particular column. In the CUSTOMERS table, for example, you might want to prevent two or more people from having identical age. Example: For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, AGE column is set to UNIQUE, so that you can not have two records with same age: CREATE TABLE CUSTOMERS(ID INT NOT NULL, TUTORIALS POINT Simply Easy Learning NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL UNIQUE, ADDRESS CHAR (25) , SALARY DECIMAL (18, 2), PRIMARY KEY (ID)); If CUSTOMERS table has already been created, then to add a UNIQUE constraint to AGE column, you would write a statement similar to the following: ALTER TABLE CUSTOMERS MODIFY AGE INT NOT NULL UNIQUE. You can also use following syntax, which supports naming the constraint in multiple columns as well: ALTER TABLE CUSTOMERS ADD CONSTRAINT myUniqueConstraint UNIQUE(AGE, SALARY); DROP a UNIQUE Constraint: To drop a UNIQUE constraint, use the following SQL: ALTER TABLE CUSTOMERS DROP CONSTRAINT myUniqueConstraint; If you are using MySQL, then you can use the following syntax: ALTER TABLE CUSTOMERS DROP INDEX myUniqueConstraint; PRIMARY Key: A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values. A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a composite key. If a table has a primary key defined on any field(s), then you can not have two records having the same value of that field(s). Note: You would use these concepts while creating database tables. Create Primary Key: Here is the syntax to define ID attribute as a primary key in a CUSTOMERS table. TUTORIALS POINT Simply Easy Learning CREATE TABLE CUSTOMERS(ID INT NOT NULL, NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL, ADDRESS CHAR (25) , SALARY DECIMAL (18, 2), PRIMARY KEY (ID)); To create a PRIMARY KEY constraint on the "ID" column when CUSTOMERS table already exists, use the following SQL syntax: ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID); NOTE: If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created). For defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax: CREATE TABLE CUSTOMERS(ID INT NOT NULL, NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL, ADDRESS CHAR (25) , SALARY DECIMAL (18, 2), PRIMARY KEY (ID, NAME)); To create a PRIMARY KEY constraint on the "ID" and "NAMES" columns when CUSTOMERS table already exists, use the following SQL syntax: ALTER TABLE CUSTOMERS ADD CONSTRAINT PK_CUSTID PRIMARY KEY (ID, NAME); Delete Primary Key: You can clear the primary key constraints from the table, Use Syntax: ALTER TABLE CUSTOMERS DROP PRIMARY KEY ; TUTORIALS POINT Simply Easy Learning FOREIGN Key: A foreign key is a key used to link two tables together. This is sometimes called a referencing key. Foreign Key is a column or a combination of columns whose values match a Primary Key in a

nonprimary fields is between the data. For example, in the below table, street name, city, and state are unbreakably bound to the zip code. CREATE TABLE CUSTOMERS(CUST_ID INT NOT NULL, CUST_NAME VARCHAR (20) NOT NULL, DOB DATE, STREET VARCHAR(200), TUTORIALS POINT Simply Easy Learning CITY VARCHAR(100), STATE VARCHAR(100), ZIP VARCHAR(12), EMAIL_ID VARCHAR(256), PRIMARY KEY (CUST_ID)); The dependency between zip code and address is called a transitive dependency. To comply with third normal form, all you need to do is move the Street, City, and State fields into their own table, which you can call the Zip Code table: CREATE TABLE ADDRESS(ZIP VARCHAR(12), STREET VARCHAR(200), CITY VARCHAR(100), STATE VARCHAR(100), PRIMARY KEY (ZIP)); Next, alter the CUSTOMERS table as follows: CREATE TABLE CUSTOMERS(CUST_ID INT NOT NULL, CUST_NAME VARCHAR (20) NOT NULL, DOB DATE, ZIP VARCHAR(12), EMAIL_ID VARCHAR(256), PRIMARY KEY (CUST_ID)); The advantages of removing transitive dependencies are mainly twofold. First, the amount of data duplication is reduced and therefore your database becomes smaller. The second advantage is data integrity. When duplicated data changes, there's a big risk of updating only some of the data, especially if it's spread out in a number of different places in the database. For example, if address and zip code data were stored in three or four different tables, then any changes in zip codes would need to ripple out to every record in those three or four tables. TUTORIALS POINT Simply Easy Learning SQL RDBMS Databases There are many popular RDBMS available to work with. This tutorial gives a brief overview of few most popular RDBMS. This would help you to compare their basic features. MySQL MySQL is an open source SQL database, which is developed by Swedish company MySQL AB. MySQL is pronounced "my ess-que-ell," in contrast with SQL, pronounced "sequel." MySQL is supporting many different platforms including Microsoft Windows, the major Linux distributions, UNIX, and Mac OS X. MySQL has free and paid versions, depending on its usage (non-commercial/commercial) and features. MySQL comes with a very fast, multi-threaded, multi-user, and robust SQL database server. History: • Development of MySQL by Michael Widenius & David Axmark beginning in 1994. • First internal release on 23 May 1995. • Windows version was released on 8 January 1998 for Windows 95 and NT. • Version 3.23: beta from June 2000, production release January 2001. • Version 4.0: beta from August 2002, production release March 2003 (unions). • Version 4.01: beta from August 2003, Jyoti adopts MySQL for database tracking. • Version 5.0: beta from June 2004, production release October 2004. • Version 5.0: beta from March 2005, production release October 2005. • Sun Microsystems acquired MySQL AB on 26 February 2008. • Version 5.1: production release 27 November 2008. CHAPTER 3 TUTORIALS POINT Simply Easy Learning Features: • High Performance. • High Availability. • Scalability and Flexibility Run anything. • Robust Transactional Support. • Web and Data Warehouse Strengths. • Strong Data Protection. • Comprehensive Application Development. • Management Ease. • Open Source Freedom and 24 x 7 Support. • Lowest Total Cost of Ownership. MS SQL Server MS SQL Server is a Relational Database Management System developed by Microsoft Inc. Its primary query languages are: • T-SQL. • ANSI SQL. History: • 1987 - Sybase releases SQL Server for UNIX. • 1988 - Microsoft, Sybase, and Aston-Tate port SQL Server to OS/2. • 1989 - Microsoft, Sybase, and Aston-Tate release SQL Server 1.0 for OS/2. • 1990 - SQL Server 1.1 is released with support for Windows 3.0 clients. • Aston-Tate drops out of SQL Server development. • 2000 - Microsoft releases SQL Server 2000. • 2001 - Microsoft releases XML for SQL Server Web Release 1 (download). • 2002 - Microsoft releases SQLXML 2.0 (renamed from XML for SQL Server). • 2002 - Microsoft releases SQLXML 3.0. TUTORIALS POINT Simply Easy Learning • 2005 - Microsoft releases SQL Server 2005 on November 7th, 2005. Features: • High Performance. • High Availability. • Database mirroring. • Database snapshots. • CLR integration. • Service Broker. • DDL triggers. • Ranking functions. • Row version-based isolation levels. • XML integration. • TRY...CATCH. • Database Mail. ORACLE It is a very large and multi-user database management system. Oracle is a relational database management

GETDATE | +-----------------------+ | 2009-10-22 12:07:18.140 | +-----------------------+ 1 row in set (0.00 sec) TUTORIALS POINT Simply Easy Learning SQL CREATE Database The SQL CREATE DATABASE statement is used to create new SQL database. Syntax: Basic syntax of CREATE DATABASE statement is as follows: CREATE DATABASE DatabaseName; Always database name should be unique within the RDBMS. Example: If you want to create new database , then CREATE DATABASE statement would be as follows: SQL> CREATE DATABASE testDB; Make sure you have admin privilege before creating any database. Once a database is created, you can check it in the list of databases as follows: SQL> SHOW DATABASES; +------------------+ | Database | +------------------+ | information_schema | | AMROOD | | TUTORIALSPOINT | | mysql | | orig | | test | | testDB | +------------------+ 7 rows in set (0.00 sec) CHAPTER 8 TUTORIALS POINT Simply Easy Learning DROP or DELETE Database The SQL DROP DATABASE statement is used to drop an existing database in SQL schema. Syntax: Basic syntax of DROP DATABASE statement is as follows: DROP DATABASE DatabaseName; Always database name should be unique within the RDBMS. Example: If you want to delete an existing database , then DROP DATABASE statement would be as follows: SQL> DROP DATABASE testDB; NOTE: Be careful before using this operation because by deleting an existing database would result in loss of complete information stored in the database. Make sure you have admin privilege before dropping any database. Once a database is dropped, you can check it. SQL> in the list of databases as follows:SHOW DATABASES; +------------------+ | Database | +------------------+ | information_schema | | AMROOD | | TUTORIALSPOINT | | mysql | | orig | | test | +------------------+ 6 rows in set (0.00 sec) CHAPTER 9 TUTORIALS POINT Simply Easy Learning SQL SELECT Database When you have multiple databases in your SQL Schema, then before starting your operation, you would need to select a database where all the operations would be performed. The SQL USE statement is used to select any existing database in SQL schema. Syntax: Basic syntax of USE statement is as follows: USE DatabaseName; Always database name should be unique within the RDBMS. Example: You can check available databases as follows: SQL> SHOW DATABASES; +------------------+ | Database | +------------------+ | information_schema | | AMROOD | | TUTORIALSPOINT | | mysql | | orig | | test | +------------------+ 6 rows in set (0.00 sec) Now, if you want to work with AMROOD database, then you can execute the following SQL command and start working with AMROOD database: SQL> USE AMROOD; CHAPTER 10 TUTORIALS POINT Simply Easy Learning SQL CREATE Table Creating a basic table involves naming the table and defining its columns and each column's data type. The SQL CREATE TABLE statement is used to create a new table. Syntax: Basic syntax of CREATE TABLE statement is as follows: CREATE TABLE table_name(column1 datatype, column2 datatype, column3 datatype, ... columnN datatype, PRIMARY KEY(one or more columns)); CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement. Then in brackets comes the list defining each column in the table and what sort of data type it is. The syntax becomes clearer with an example below. A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement. You can check complete details at Create Table Using another Table. Create Table Using another Table A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement. The new table has the same column definitions. All columns or specific columns can be selected. When you create a new table using existing table, new table would be populated using existing values in the old table. Syntax: The basic syntax for creating a table from another table is as follows: CHAPTER 11 TUTORIALS POINT Simply Easy Learning CREATE TABLE NEW_TABLE_NAME AS SELECT [column1, column2...columnN] FROM EXISTING_TABLE_NAME [WHERE] Here, column1, column2...are the fields of existing table and same would be used to create fields of new table. Example: Following is an example, which would create a table SALARY using CUSTOMERS table and having fields customer ID and customer SALARY: SQL> CREATE TABLE SALARY AS SELECT ID, SALARY

comparisons with different operators in the same SQL statement. The AND Operator: The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause. Syntax: The basic syntax of AND operator with WHERE clause is as follows: SELECT column1, column2, columnN FROM table_name WHERE [condition1] AND [condition2]...AND [conditionN]; You can combine N number of conditions using AND operator. For an action to be taken by the SQL statement, whether it be a transaction or query, all conditions separated by the AND must be TRUE. Example: Consider the CUSTOMERS table having the following records: +----+---------+-----+---------+---------+ | ID | NAME | AGE | ADDRESS | SALARY | +----+---------+-----+---------+---------+ | 1 | Ramesh | 32 | Ahmedabad | 2000.00 | | 2 | Khilan | 25 | Delhi | 1500.00 | | 3 | kaushik | 23 | Kota | 2000.00 | | 4 | Chaitali | 25 | Mumbai | 6500.00 | | 5 | Hardik | 27 | Bhopal | 8500.00 | | 6 | Komal | 22 | MP | 4500.00 | | 7 | Muffy | 24 | Indore | 10000.00 | +----+---------+-----+---------+---------+ Following is an example, which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is greater than 2000 AND age is less tan 25 years: CHAPTER 16 TUTORIALS POINT Simply Easy Learning SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY > 2000 AND age < 25; This would produce the following result: +----+-------+---------+ | ID | NAME | SALARY | +----+-------+---------+ | 6 | Komal | 4500.00 | | 7 | Muffy | 10000.00 | +----+-------+---------+ The OR Operator: The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause. Syntax: The basic syntax of OR operator with WHERE clause is as follows: SELECT column1, column2, columnN FROM table_name WHERE [condition1] OR [condition2]...OR [conditionN] You can combine N number of conditions using OR operator. For an action to be taken by the SQL statement, whether it be a transaction or query, only any ONE of the conditions separated by the OR must be TRUE. Example: Consider the CUSTOMERS table having the following records: +----+---------+-----+---------+---------+ | ID | NAME | AGE | ADDRESS | SALARY | +----+---------+-----+---------+---------+ | 1 | Ramesh | 32 | Ahmedabad | 2000.00 | | 2 | Khilan | 25 | Delhi | 1500.00 | | 3 | kaushik | 23 | Kota | 2000.00 | | 4 | Chaitali | 25 | Mumbai | 6500.00 | | 5 | Hardik | 27 | Bhopal | 8500.00 | | 6 | Komal | 22 | MP | 4500.00 | | 7 | Muffy | 24 | Indore | 10000.00 | +----+---------+-----+---------+---------+ Following is an example, which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is greater than 2000 OR age is less tan 25 years: SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY > 2000 OR age < 25; This would produce the following result: +----+---------+---------+ | ID | NAME | SALARY | +----+---------+---------+ | 3 | kaushik | 2000.00 | | 4 | Chaitali | 6500.00 | TUTORIALS POINT Simply Easy Learning | 5 | Hardik | 8500.00 | | 6 | Komal | 4500.00 | | 7 | Muffy | 10000.00 | +----+---------+---------+ TUTORIALS POINT Simply Easy Learning SQL UPDATE Query The SQL UPDATE Query is used to modify the existing records in a table. You can use WHERE clause with UPDATE query to update selected rows, otherwise all the rows would be affected. Syntax: The basic syntax of UPDATE query with WHERE clause is as follows: UPDATE table_name SET column1 = value1, column2 = value2...., columnN = valueN WHERE [condition]; You can combine N number of conditions using AND or OR operators. Example: Consider the CUSTOMERS table having the following records: +----+---------+-----+---------+---------+ | ID | NAME | AGE | ADDRESS | SALARY | +----+---------+-----+---------+---------+ | 1 | Ramesh | 32 | Ahmedabad | 2000.00 | | 2 | Khilan | 25 | Delhi | 1500.00 | | 3 | kaushik | 23 | Kota | 2000.00 | | 4 | Chaitali | 25 | Mumbai | 6500.00 | | 5 | Hardik | 27 | Bhopal | 8500.00 | | 6 | Komal | 22 | MP | 4500.00 | | 7 | Muffy | 24 | Indore | 10000.00 | +----+---------+-----+---------+---------+ Following is an example, which would update ADDRESS for a customer whose ID is 6: SQL> UPDATE CUSTOMERS SET ADDRESS = 'Pune' WHERE ID = 6; Now, CUSTOMERS table would have the following records: CHAPTER 17 TUTORIALS POINT Simply Easy Learning +----+---------+-----+---------+---------+ | ID | NAME | AGE | ADDRESS | SALARY | +----+---------+-----+---------+---------+ | 1 | Ramesh | 32 | Ahmedabad | 2000.00 | | 2 | Khilan | 25 | Delhi | 1500.00 | | 3 | kaushik | 23 | Kota | 2000.00 | | 4 | Chaitali | 25 | Mumbai | 6500.00 | | 5 | Hardik | 27 | Bhopal | 8500.00 | | 6 |

certain conditions. • INDEX: Use to create and retrieve data from the database very quickly. NOT NULL Constraint: By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such constraint on this column specifying that NULL is now not allowed for that column. A NULL is not the same as no data, rather, it represents unknown data. Example: For example, the following SQL creates a new table called CUSTOMERS and adds five columns, three of which, ID and NAME and AGE, specify not to accept NULLs: CREATE TABLE CUSTOMERS(ID INT NOT NULL, NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL, CHAPTER 25 TUTORIALS POINT Simply Easy Learning ADDRESS CHAR (25) , SALARY DECIMAL (18, 2), PRIMARY KEY (ID)); If CUSTOMERS table has already been created, then to add a NOT NULL constraint to SALARY column in Oracle and MySQL, you would write a statement similar to the following: ALTER TABLE CUSTOMERS MODIFY SALARY DECIMAL (18, 2) NOT NULL; DEFAULT Constraint: The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value. Example: For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, SALARY column is set to 5000.00 by default, so in case INSERT INTO statement does not provide a value for this column, then by default this column would be set to 5000.00. CREATE TABLE CUSTOMERS(ID INT NOT NULL, NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL, ADDRESS CHAR (25) , SALARY DECIMAL (18, 2) DEFAULT 5000.00, PRIMARY KEY (ID)); If CUSTOMERS table has already been created, then to add a DFAULT constraint to SALARY column, you would write a statement similar to the following: ALTER TABLE CUSTOMERS MODIFY SALARY DECIMAL (18, 2) DEFAULT 5000.00; Drop Default Constraint: To drop a DEFAULT constraint, use the following SQL: ALTER TABLE CUSTOMERS ALTER COLUMN SALARY DROP DEFAULT; TUTORIALS POINT Simply Easy Learning UNIQUE Constraint: The UNIQUE Constraint prevents two records from having identical values in a particular column. In the CUSTOMERS table, for example, you might want to prevent two or more people from having identical age. Example: For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, AGE column is set to UNIQUE, so that you can not have two records with same age: CREATE TABLE CUSTOMERS(ID INT NOT NULL, NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL UNIQUE, ADDRESS CHAR (25) , SALARY DECIMAL (18, 2), PRIMARY KEY (ID)); If CUSTOMERS table has already been created, then to add a UNIQUE constraint to AGE column, you would write a statement similar to the following: ALTER TABLE CUSTOMERS MODIFY AGE INT NOT NULL UNIQUE; You can also use the following syntax, which supports naming the constraint in multiple columns as well: ALTER TABLE CUSTOMERS ADD CONSTRAINT myUniqueConstraint UNIQUE(AGE, SALARY); DROP a UNIQUE Constraint: To drop a UNIQUE constraint, use the following SQL: ALTER TABLE CUSTOMERS DROP CONSTRAINT myUniqueConstraint; If you are using MySQL, then you can use the following syntax: ALTER TABLE CUSTOMERS DROP INDEX myUniqueConstraint; TUTORIALS POINT Simply Easy Learning PRIMARY Key: A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values. A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a composite key. If a table has a primary key defined on any field(s), then you can not have two records having the same value of that field(s). Note: You would use these concepts while creating database tables. Create Primary Key: Here is the syntax to define ID attribute as a primary key in a CUSTOMERS table. CREATE TABLE CUSTOMERS(ID INT NOT NULL, NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL, ADDRESS CHAR (25) , SALARY DECIMAL (18, 2), PRIMARY KEY (ID)); To create a PRIMARY KEY constraint on the "ID" column when CUSTOMERS table already exists, use the following SQL syntax: ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID); NOTE: If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created). For defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

example, to drop the primary key constraint in the EMPLOYEES table, you can use the following command: ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES_PK; Some implementations may provide shortcuts for dropping certain constraints. For example, to drop the primary key constraint for a table in Oracle, you can use the following command: ALTER TABLE EMPLOYEES DROP PRIMARY KEY; Some implementations allow you to disable constraints. Instead of permanently dropping a constraint from the database, you may want to temporarily disable the constraint, and then enable it later. Integrity Constraints: Integrity constraints are used to ensure accuracy and consistency of data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity. There are many types of integrity constraints that play a role in referential integrity (RI). These constraints include Primary Key, Foreign Key, Unique Constraints and other constraints mentioned above. TUTORIALS POINT Simply Easy Learning SQL Joins The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each. Consider the following two tables, (a) CUSTOMERS table is as follows: +----+----------+-----+-----------+---------+ | ID | NAME | AGE | ADDRESS | SALARY | +----+----------+-----+-----------+---------+ | 1 | Ramesh | 32 | Ahmedabad | 2000.00 | | 2 | Khilan | 25 | Delhi | 1500.00 | | 3 | kaushik | 23 | Kota | 2000.00 | | 4 | Chaitali | 25 | Mumbai | 6500.00 | | 5 | Hardik | 27 | Bhopal | 8500.00 | | 6 | Komal | 22 | MP | 4500.00 | | 7 | Muffy | 24 | Indore | 10000.00 | +----+----------+-----+-----------+---------+ (b) Another table is ORDERS as follows: +-----+---------------------+-------------+--------+ |OID | DATE | CUSTOMER_ID | AMOUNT | +-----+---------------------+-------------+--------+ | 102 | 2009-10-08 00:00:00 | 3 | 3000 | | 100 | 2009-10-08 00:00:00 | 3 | 1500 | | 101 | 2009-11-20 00:00:00 | 2 | 1560 | | 103 | 2008-05-20 00:00:00 | 4 | 2060 | +-----+---------------------+-------------+--------+ Now, let us join these two tables in our SELECT statement as follows: SQL> SELECT ID, NAME, AGE, AMOUNT FROM CUSTOMERS, ORDERS WHERE CUSTOMERS.ID = ORDERS.CUSTOMER_ID; This would produce the following result: +----+----------+-----+--------+ | ID | NAME | AGE | AMOUNT | +----+----------+-----+--------+ | 3 | kaushik | 23 | 3000 | | 3 | kaushik | 23 | 1500 | | 2 | Khilan | 25 | 1560 | CHAPTER 26 TUTORIALS POINT Simply Easy Learning | 4 | Chaitali | 25 | 2060 | +----+----------+-----+--------+ Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, , <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol. SQL Join Types: There are different types of joins available in SQL: • INNER JOIN: returns rows when there is a match in both tables. • LEFT JOIN: returns all rows from the left table, even if there are no matches in the right table. • RIGHT JOIN: returns all rows from the right table, even if there are no matches in the left table. • FULL JOIN: returns rows when there is a match in one of the tables. • SELF JOIN: is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement. • CARTESIAN JOIN: returns the Cartesian product of the sets of records from the two or more joined tables. INNER JOIN The most frequently used and important of the joins is the INNER JOIN. They are also referred to as an EQUIJOIN. The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row. Syntax: The basic syntax of INNER JOIN is as follows: SELECT table1.column1, table2.column2... FROM table1 INNER JOIN table2 ON table1.common_filed = table2.common_field; Example: Consider the following two tables, (a) CUSTOMERS table is as follows: +----+----------+-----+-----------+---------+ | ID | NAME | AGE | ADDRESS | SALARY | +----+----------+-----+-----------+---------+ | 1 | Ramesh | 32 | Ahmedabad | 2000.00 | | 2 | Khilan | 25 | Delhi | 1500.00 | | 3 | kaushik | 23 | Kota | 2000.00 | | 4 | Chaitali | 25 | Mumbai | 6500.00 | | 5 | Hardik | 27 | Bhopal | 8500.00 | | 6 | Komal | 22 | MP | 4500.00 | | 7 | Muffy | 24 | Indore | 10000.00 | +----+----------+-----+-----------+---------+

+------+----------------------+ | name | AVG(daily_typing_pages) | +------+----------------------+ | Jack | 135.0000 | | Jill | 220.0000 | TUTORIALS POINT Simply Easy Learning | John | 250.0000 | | Ram | 220.0000 | | Zara | 325.0000 | +------+----------------------+ 5 rows in set (0.20 sec) SQL SUM Function SQL SUM function is used to find out the sum of a field in various records. To under stand SUM function, consider an employee_tbl table, which is having the following records: SQL> SELECT * FROM employee_tbl; +------+------+-----------+-----------------+ | id | name | work_date | daily_typing_pages | +------+------+-----------+-----------------+ | 1 | John | 2007-01-24 | 250 | | 2 | Ram | 2007-05-27 | 220 | | 3 | Jack | 2007-05-06 | 170 | | 3 | Jack | 2007-04-06 | 100 | | 4 | Jill | 2007-04-06 | 220 | | 5 | Zara | 2007-06-06 | 300 | | 5 | Zara | 2007-02-06 | 350 | +------+------+-----------+-----------------+ 7 rows in set (0.00 sec) Now suppose based on the above table you want to calculate total of all the dialy_typing_pages, then you can do so by using the following command: SQL> SELECT SUM(daily_typing_pages) -> FROM employee_tbl; +----------------------+ | SUM(daily_typing_pages) | +----------------------+ | 1610 | +----------------------+ 1 row in set (0.00 sec) You can take sum of various records set using GROUP BY clause. Following example will sum up all the records related to a single person and you will have total typed pages by every person. SQL> SELECT name, SUM(daily_typing_pages) -> FROM employee_tbl GROUP BY name; +------+----------------------+ | name | SUM(daily_typing_pages) | +------+----------------------+ | Jack | 270 | | Jill | 220 | | John | 250 | | Ram | 220 | | Zara | 650 | +------+----------------------+ 5 rows in set (0.17 sec) SQL SQRT Function SQL SQRT function is used to find out the square root of any number. You can Use SELECT statement to find out squre root of any number as follows: TUTORIALS POINT Simply Easy Learning SQL> select SQRT(16); +----------+ | SQRT(16) | +----------+ | 4.000000 | +----------+ 1 row in set (0.00 sec) You are seeing float value here because internally SQL will manipulate square root in float data type. You can use SQRT function to find out square root of various records as well. To understand SQRTfunction in more detail, consider an employee_tbl table, which is having the following records: SQL> SELECT * FROM employee_tbl; +------+------+-----------+-----------------+ | id | name | work_date | daily_typing_pages | +------+------+-----------+-----------------+ | 1 | John | 2007-01-24 | 250 | | 2 | Ram | 2007-05-27 | 220 | | 3 | Jack | 2007-05-06 | 170 | | 3 | Jack | 2007-04-06 | 100 | | 4 | Jill | 2007-04-06 | 220 | | 5 | Zara | 2007-06-06 | 300 | | 5 | Zara | 2007-02-06 | 350 | +------+------+-----------+-----------------+ 7 rows in set (0.00 sec) Now suppose based on the above table you want to calculate square root of all the dialy_typing_pages, then you can do so by using the following command: SQL> SELECT name, SQRT(daily_typing_pages) -> FROM employee_tbl; +------+----------------------+ | name | SQRT(daily_typing_pages) | +------+----------------------+ | John | 15.811388 | | Ram | 14.832397 | | Jack | 13.038405 | | Jack | 10.000000 | | Jill | 14.832397 | | Zara | 17.320508 | | Zara | 18.708287 | +------+----------------------+ 7 rows in set (0.00 sec) SQL RAND Function SQL has a RAND function that can be invoked to produce random numbers between 0 and 1: SQL> SELECT RAND( ), RAND( ), RAND( ); +----------------+----------------+----------------+ | RAND( ) | RAND( ) | RAND( ) | +----------------+----------------+----------------+ | 0.45464584925645 | 0.1824410643265 | 0.54826780459682 | +----------------+----------------+----------------+ 1 row in set (0.00 sec) When invoked with an integer argument, RAND( ) uses that value to seed the random number generator. Each time you seed the generator with a given value, RAND( ) will produce a repeatable series of numbers: TUTORIALS POINT Simply Easy Learning SQL> SELECT RAND(1), RAND( ), RAND( ); +----------------+----------------+----------------+ | RAND(1) | RAND( ) | RAND( ) | +----------------+----------------+----------------+ | 0.18109050223705 | 0.75023211143001 | 0.20788908117254 | +----------------+----------------+----------------+ 1 row in set (0.00 sec) You can use ORDER BY RAND() to randomize a set of rows or values as follows: To understand ORDER BY RAND() function, consider an employee_tbl table, which is having the following records: SQL> SELECT * FROM employee_tbl; +------+------+-----------+-----------------+ | id | name | work_date | daily_typing_pages | +------+------+-----------+-----------------+ | 1 | John | 2007-01-24 | 250 | | 2 | Ram | 2007-05-27 | 220 | | 3 | Jack |

LEFT('foobarbar', 5); +----------+ | LEFT('foobarbar', 5) | +----------+ | fooba | +----------+1 row in set (0.00 sec) LENGTH(str) Returns the length of the string str measured in bytes. A multi-byte character counts as multiple bytes. This means that for a string containing five two-byte characters, LENGTH() returns 10, whereas CHAR_LENGTH() returns 5. SQL> SELECT LENGTH('text'); +----------+ | LENGTH('text') | +----------+ | 4 | +----------+1 row in set (0.00 sec) LOAD_FILE(file_name) Reads the file and returns the file contents as a string. To use this function, the file must be located on the server host, you must specify the full pathname to the file, and you must have the FILE privilege. The file must be readable by all and its size less than max_allowed_packet bytes. If the file does not exist or cannot be read because one of the preceding conditions is not satisfied, the function returns NULL. As of SQL 5.0.19, the character_set_filesystem system variable controls interpretation of filenames that are given as literal strings. SQL> UPDATE table_test ->SET blob_col=LOAD_FILE('/tmp/picture') ->WHERE id=1; ..... TUTORIALS POINT Simply Easy Learning LOCATE(substr,str), LOCATE(substr,str,pos) The first syntax returns the position of the first occurrence of substring substr in string str. The second syntax returns the position of the first occurrence of substring substr in string str, starting at position pos. Returns 0 if substr is not in str. SQL> SELECT LOCATE('bar', 'foobarbar'); +----------+ | LOCATE('bar', 'foobarbar') | +----------+ | 4 | +----------+1 row in set (0.00 sec) LOWER(str) Returns the string str with all characters changed to lowercase according to the current character set mapping. SQL> SELECT LOWER('QUADRATICALLY'); +----------+ | LOWER('QUADRATICALLY') | +----------+ | quadratically | +----------+1 row in set (0.00 sec) LPAD(str,len,padstr) Returns the string str, left-padded with the string padstr to a length of len characters. If str is longer than len, the return value is shortened to len characters. SQL> SELECT LPAD('hi',4,'??'); +----------+ | LPAD('hi',4,'??') | +----------+ | ??hi | +----------+1 row in set (0.00 sec) LTRIM(str) Returns the string str with leading space characters removed. SQL> SELECT LTRIM(' barbar'); +----------+ | LTRIM(' barbar') | +----------+ | barbar | +----------+1 row in set (0.00 sec) MAKE_SET(bits,str1,str2,...) Returns a set value (a string containing substrings separated by ,. characters) consisting of the strings that have the corresponding bit in bits set. str1 corresponds to bit 0, str2 to bit 1, and so on. NULL values in str1, str2, ... are not appended to the result. TUTORIALS POINT Simply Easy Learning SQL> SELECT MAKE_SET(1,'a','b','c'); +----------+ | MAKE_SET(1,'a','b','c') | +----------+ | a | +----------+1 row in set (0.00 sec) MID(str,pos,len) MID(str,pos,len) is a synonym for SUBSTRING(str,pos,len). OCT(N) Returns a string representation of the octal value of N, where N is a longlong (BIGINT) number. This is equivalent to CONV(N,10,8). Returns NULL if N is NULL. SQL> SELECT OCT(12); +----------+ | OCT(12) | +----------+ | 14 | +----------+1 row in set (0.00 sec) OCTET_LENGTH(str) OCTET_LENGTH() is a synonym for LENGTH(). ORD(str) If the leftmost character of the string str is a multi-byte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula: (1st byte code) + (2nd byte code . 256) + (3rd byte code . 2562) ... If the leftmost character is not a multi-byte character, ORD() returns the same value as the ASCII() function. SQL> SELECT ORD('2'); +----------+ | ORD('2') | +----------+ | 50 | +----------+1 row in set (0.00 sec) POSITION(substr IN str) POSITION(substr IN str) is a synonym for LOCATE(substr,str). TUTORIALS

POINT Simply Easy Learning QUOTE(str) Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotes and with each instance of single quote (’ ‘ ‘), backslash (‘\’), ASCII NUL, and Control-Z preceded by a backslash. If the argument is NULL, the return value is the word ’NULL‘ without enclosing single quotes. SQL> SELECT QUOTE('Don\'t!'); +--+ | QUOTE('Don\'t!') | +--+ | 'Don\'t!' | +--+ 1 row in set (0.00 sec) NOTE: Please check if your installation has any bug with this function then don't use this function. expr REGEXP pattern This function performs a pattern match of expr against pattern. Returns 1 if expr matches pat; otherwise it returns 0. If either expr or pat is NULL, the result is NULL. REGEXP is not case sensitive, except when used with binary strings. SQL> SELECT 'ABCDEF' REGEXP 'A%C%%'; +---+ | 'ABCDEF' REGEXP 'A%C%%' | +---+ | 0 | +---+ 1 row in set (0.00 sec) Another example is: SQL> SELECT 'ABCDE' REGEXP '.*'; +---+ | 'ABCDE' REGEXP '.*' | +---+ | 1 | +---+ 1 row in set (0.00 sec) Let's see one more example: SQL> SELECT 'new*\n*line' REGEXP 'new*.*line'; +---+ | 'new*\n*line' REGEXP 'new*.*line' | +---+ | 1 | +---+ 1 row in set (0.00 sec) REPEAT(str,count) Returns a string consisting of the string str repeated count times. If count is less than 1, returns an empty string. Returns NULL if str or count are NULL. SQL> SELECT REPEAT('SQL', 3); +--+ TUTORIALS POINT Simply Easy Learning | REPEAT('SQL', 3) | +--+ | SQLSQLSQL | +--+ 1 row in set (0.00 sec) REPLACE(str,from_str,to_str) Returns the string str with all occurrences of the string from_str replaced by the string to_str. REPLACE() performs a case-sensitive match when searching for from_str. SQL> SELECT REPLACE('www.mysql.com', 'w', 'Ww'); +---+ | REPLACE('www.mysql.com', 'w', 'Ww') | +---+ | WwWwWw.mysql.com | +---+ 1 row in set (0.00 sec) REVERSE(str) Returns the string str with the order of the characters reversed. SQL> SELECT REVERSE('abcd'); +---+ | REVERSE('abcd') | +---+ | dcba | +---+ 1 row in set (0.00 sec) RIGHT(str,len) Returns the rightmost len characters from the string str, or NULL if any argument is NULL. SQL> SELECT RIGHT('foobarbar', 4); +--+ | RIGHT('foobarbar', 4) | +--+ | rbar | +--+ 1 row in set (0.00 sec) RPAD(str,len,padstr) Returns the string str, right-padded with the string padstr to a length of len characters. If str is longer than len, the return value is shortened to len characters. SQL> SELECT RPAD('hi',5,'?'); +---+ | RPAD('hi',5,'?') | +---+ | hi??? | +---+ 1 row in set (0.00 sec) TUTORIALS POINT Simply Easy Learning RTRIM(str) Returns the string str with trailing space characters removed. SQL> SELECT RTRIM('barbar '); +---+ | RTRIM('barbar ') | +---+ | barbar | +---+ 1 row in set (0.00 sec) SOUNDEX(str) Returns a soundex string from str. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the SOUNDEX() function returns an arbitrarily long string. You can use SUBSTRING() on the result to get a standard soundex string. All non-alphabetic characters in str are ignored. All international alphabetic characters outside the A-Z range are treated as vowels. SQL> SELECT SOUNDEX('Hello'); +---+ | SOUNDEX('Hello') | +---+ | H400 | +---+ 1 row in set (0.00 sec) expr1 SOUNDS LIKE expr2 This is

synonym for UPPER(). TUTORIALS POINT Simply Easy Learning UNHEX(str) Performs the inverse operation of HEX(str). That is, it interprets each pair of hexadecimal digits in the argument as a number and converts it to the character represented by the number. The resulting characters are returned as a binary string. SQL> SELECT UNHEX('4D7953514C'); +----------+ | UNHEX('4D7953514C') | +----------+ | SQL | +----------+1 row in set (0.00 sec) The characters in the argument string must be legal hexadecimal digits: '0' .. '9', 'A' .. 'F', 'a' .. 'f'. If UNHEX() encounters any non-hexadecimal digits in the argument, it returns NULL. UPPER(str) Returns the string str with all characters changed to uppercase according to the current character set mapping. SQL> SELECT UPPER('Allah-hus-samad'); +----------+ | UPPER('Allah-hus-samad') | +----------+ | ALLAH-HUS-SAMAD | +----------+1 row in set (0.00 sec)