Database Systems Laboratory 1

Introduction To SQL

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Introduction To SQL

RDBMS

- Relational database is a collection of related information that has been organized into tables. Each table contains rows and columns
- The tables are stored in the database in structures known as schemas
- Each row is called an entity, thus table as entity set
- Row is known as Tuple
- Columns are the properties called Attributes
- The facts describing an entity are known as data
- For an attribute, the set of permitted values is called domain of that attribute

Employee Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	09-NOV-81	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-SEP-81	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

Examples of RDBMS

- Oracle
- DB2
- Microsoft SQL-Server
- MySQL
- Microsoft Access
- Apache Derby
- Visual FoxPro
- OpenBase
- PostgreSQL
- SQLite
- Vertica
- IBM Informix
- Ingres
- IBM Lotus
- SQL Anywhere

Client/Server Database

- Client/Server databases run the DBMS as a process on the server and run a client database application on each client
- The client application sends a request for data over the network to the server
- When the server receives the client request, the DBMS retrieves data from the database, performs the required processing on the data, and sends only the requested data back to the client over the network

SQL

- SQL(Structured Query Language) is the standard language for relational database
- IBM developed the original version of SQL in early 1970s, with name Sequel
- In 1986, the ANSI & ISO published the SQL standard, SQL-86
- ANSI published extended standard, SQL-89 in 1989
- Next versions are SQL-92, SQL-1999, SQL-2003, SQL-2006, SQL-2008, SQL-2011
- SQL is the ideal database language to
  - create database and table structures
  - perform basic data management operation
  - perform complex queries to transform data into useful information

SQL Statements

Data Definition Language(DDL)

Defines the data structure that make up a database

- CREATE statement
- ALTER statement
- DROP statement
- RENAME statement
- TRUNCATE statement

Data Manipulation Language(DML)

Modifies the contents of tables

- INSERT statement
- UPDATE statement
- DELETE statement

SQL Statements...

Query Statement

Retrieves data from the database

- SELECT statement

Transaction Control Language(TCL)

Permanently records the changes made to the rows stored in a table or undoes those changes affected by DML statements

- COMMIT statement
- ROLLBACK statement
- SAVEPOINT statement

Data Control Language(DCL)

Gives and removes permissions on database structure

- GRANT statement
- REVOKE statement

SQL*PLUS

- It is a tool to manipulate data & perform queries against the database
- It enables you to conduct a conversation with the database
- Two versions of SQL*PLUS
  - Graphical version
    - Start->All Programs->Oracle Database 10g Express Edition->Go to Database home page(SQL->SQL Commands)
  - Command-Line version
    - Start->All Programs->Oracle Database 10g Express Edition->Run SQL Command Line(Connect)

Naming Conventions

- A table is an object that can store data in a database
- When you create a table, you must specify the table name, name of each column, data type of each column, and size of each column
- The table and column names can be up to 30 characters long
- Table or column name must begin with a letter
- The names are not case sensitive
- Spaces and hyphens are not allowed in a table or a column name; but $, _ and # are allowed

Data Types

Data type specifies the type of data that will be stored in the column. Data types also help to optimize storage space

- CHAR(n)
  - Stores fixed-length alphanumeric data in a column
  - Default and minimum size is one character
  - Maximum allowable size is 2000 characters (previously 255)
  - If a string of a smaller length is stored, it is padded with spaces at the end
- VARCHAR(n)/VARCHAR2(n)
  - Stores variable-length alphanumeric data in a column
  - Default and minimum size is one character
  - Maximum allowable size is 4000 characters (previously 2000)
  - If the data are smaller than the specified size, only the data value is stored; no padding is done

Data Types...

- DATE
  - Stores date and time values
  - The range of allowable dates is between January 1, 4712B.C. and December 31, 9999A.D.
  - The default date format is DD-MON-YY. The DD-MON-YYYY format also works
- NUMBER(precision, scale)
  - Stores floating point numbers as well as integer numbers. Precision is the total number of significant digits in the number; scale is the total number of digits to the right of the decimal point(if used). The precision can range from 1 to 38
  - If neither precision nor scale is specified, any number may be stored up to a precision of 38 digits
- INTEGER
  - Stores integer number

Data Types...

- NUMERIC(p,d)
  - Stores fixed-point number with user specified precision
  - Similar to NUMBER data type
- LONG
  - Stores variable length character strings containing up to 2GB
  - Similar to VARCHAR
  - There can be one LONG data type per table
- RAW
  - Stores binary data such as digitized picture or image
  - Maximum allowable size is 2000 Bytes (previously 255 Bytes)
- LONG RAW
  - It is the higher range of RAW
  - There can be one LONG data type per table
  - Maximum allowable size id 2GB

Data Types...

- LOB(Large Object)
  - Stores large volume of data
  - BLOB
    - Used for binary data such as graphics, video clips and audio files up to 4GB
  - CLOB
    - Used for character data up to 4GB
  - BFILE
    - Stores references to a binary file that is external to the database and is maintained by the operating system's file system

Table Creation

CREATE TABLE statement

CREATE statement is used for table creation. The syntax is:

CREATE TABLE table_name(column datatype, column datatype, ... column datatype);

For example, create a table for STUDENT (Roll, Name, Gender, Age, CGPA)

Solution: *CREATE TABLE STUDENT(Roll NUMBER(6), Name VARCHAR2(20), Gender CHAR(1), Age NUMBER(3), CGPA NUMBER(4,2));*

Viewing Table Structure

DESCRIBE statement

DESCRIBE statement is used for viewing table structure. The syntax is:

DESCRIBE table_name; or
DESC table_name;

For example: *DESCRIBE STUDENT;*
Solution:

Name	Null?	Type
Roll		NUMBER(6)
Name		VARCHAR2(20)
Gender		CHAR(1)
Age		NUMBER(3)
CGPA		NUMBER(4,2)

Record Insertion

INSERT statement

INSERT statement is used to insert a new row/record into a table. The syntax is:

INSERT INTO table_name (column1, column2,..) VALUES (value1, value2,..);

- Column names are optional
- Numeric data is not enclosed within quotes; while character and date values are enclosed within single quotes

Entering NULL values:

- *Implicit method*: Here, column name is omitted from the column list in the INSERT statement
- *Explicit method*: Here, the null value is used as a value for a numeric column, and an empty string (”) is used for date or character columns

Record Insertion...

Substitution variables

- Substitution variables enable you to create an interactive SQL script, which prompts you to enter a value for the substitution variable
- In command line version, & character is used before the substitution variable in the query; whereas : character is used in graphical version
- Substitution variables for character and date columns are enclosed within a pair of single quotation marks
- For more records, press /
- If an INSERT statement contains a value containing & character, it is treated as a substitution variable. In such cases, **SET DEFINE OFF;** and **SET DEFINE ON;** commands are used

Record Insertion...

- *INSERT INTO STUDENT(Roll, Name, Gender, Age, CGPA) VALUES (705129, 'Uday', 'M', 19, 9.2);*
- *INSERT INTO STUDENT VALUES (705129, 'Uday', 'M', 19, 9.2);*
- *INSERT INTO STUDENT(Roll, Name, CGPA) VALUES (705129, 'Uday', 9.2);*
- *INSERT INTO STUDENT VALUES (&Roll, '&Name', '&Gender', &Age, &CGPA);*
- *INSERT INTO STUDENT (Roll, Name, Gender, Age) VALUES(&Roll, '&Name', '&Gender', &Age);*

Record Insertion...

Customized Prompts

ACCEPT command is used for customized prompts. The syntax is:
ACCEPT variablename PROMPT 'prompt message'

- ACCEPT Roll PROMPT 'Please enter the Roll of Student:'
- ACCEPT Name PROMPT 'Please enter the Name of Student:'
- ACCEPT Gender PROMPT 'Please enter the Gender of Student:'
- ACCEPT Age PROMPT 'Please enter the Age of Student:'
- ACCEPT CGPA PROMPT 'Please enter the CGPA of Student:'

INSERT INTO STUDENT VALUES (&Roll, '&Name', '&Gender', &Age, &CGPA);

Once a variable is defined with substitution variable or ACCEPT, its value is known throughout that session

Inserting Time

Time insertion

Time can be inserted by using TO_DATE()

INSERT INTO STUDENT(dob) VALUES (TO_DATE('12-JAN-1990 10:34:45 P.M.', 'DD-MON-YYYY HH:MI:SS P.M.'));

- If only the date value is entered in a date-type column, the time value is set to the midnight (12:00A.M.)
- If only the time value is entered in a date-type column, the date is set to first of the current month

Querying Data

SELECT statement

SELECT statement is used to retrieve data from the underlying table. The syntax is:
SELECT column1,column2 FROM table_name;

If the user wants to see all the columns in a table, * can be used in place of columns

*SELECT * FROM STUDENT;*

Roll	Name	Gender	Age	CGPA
705129	Uday	M	19	9.2
705170	Ram	M	20	
705171	Kim	F	19	8.6
705172	Raji		20	7.5

NULL value means the value is unknown or doesn't exist

Querying Data...

SELECT Roll, Name, CGPA FROM STUDENT;

Roll	Name	CGPA
705129	Uday	9.2
705170	Ram	
705171	Kim	8.6
705172	Raji	7.5

Database Systems Laboratory 2

SQL Fundamentals

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

SQL Fundamentals

Student table

Roll	Name	City	Age	CGPA
101	Ram	Bhubaneswar	19	9.0
102	Hari	Bhubaneswar		6.7
103	Uday	Jharkhand	20	8.97
104	Vikas	Uttar Pradesh	19	8.5
105	Sweta	Ranchi	19	9.2
106	Yogesh	Rajastan	18	7.9
210	Smriti	Delhi	20	8.99
211	Sudam	Cuttack	21	8.6
212	Vikas	Kolkota	23	5.98
165	Manish		19	9.15

- The character data is displayed with left justification, while numeric data with right justification

SELECT Statement

SELECT statement

SELECT statement is used to retrieve data from the underlying table. The syntax is:
SELECT columns FROM tablename;

If the user wants to see all the columns in a table, * can be used in place of columns

SELECT Roll, CGPA FROM Student;

Roll	CGPA
101	9.0
102	6.7
103	8.97
104	8.5
105	9.2
106	7.9
210	8.99
211	8.6
212	5.98
165	9.15

SELECT Statement...

Displaying Distinct Rows

The DISTINCT keyword is used to suppress duplicate values.
The syntax is:
SELECT DISTINCT column FROM tablename;

SELECT DISTINCT City FROM Student;

City
Bhubaneswar
Jharkhand
Uttar Pradesh
Ranchi
Rajastan
Delhi
Cuttack
Kolkota

SELECT Statement...

Use of Arithmetic Expressions

The arithmetic expressions are used to display mathematically calculated data. The syntax is:
SELECT column, expression FROM tablename;

SELECT Name, Age, Age+3 FROM Student;

Name	Age	Age+3
Ram	19	22
Uday	20	23
Vikas	19	22
Sweta	19	22
Yogesh	18	21
Smriti	20	23
Sudam	21	24
Vikas	23	26
Manish	19	22

SELECT Statement...

Use of Alias

The column aliases are used to rename a table's columns for the purpose of a particular SQL query. The syntax is:
SELECT column1, column2 [AS] Alias FROM tablename;

SELECT Name, Age, Age+3 "Passing Age"FROM Student;

Name	Age	Passing Age
Ram	19	22
Uday	20	23
Vikas	19	22
Sweta	19	22
Yogesh	18	21
Smriti	20	23
Sudam	21	24
Vikas	23	26
Manish	19	22

SELECT Statement...

Concatenation

Concatenation joins a column or a character string to another column. The syntax is:
SELECT column1||' '||column2 [AS] ALIAS FROM tablename;

SELECT Name||' '||City FROM Student;
SELECT Name||' '||City AS "Address"FROM Student;

Name||' '||City	Address
Ram Bhubaneswar	Ram Bhubaneswar
Hari Bhubaneswar	Hari Bhubaneswar
Uday Jharkhand	Uday Jharkhand
Vikas Uttar Pradesh	Vikas Uttar Pradesh
Sweta Ranchi	Sweta Ranchi
Yogesh Rajastan	Yogesh Rajastan
Smriti Delhi	Smriti Delhi
Sudam Cuttack	Sudam Cuttack
Vikas Kolkota	Vikas Kolkota
Manish	Manish

SELECT Statement...

Displaying Time

Time of a date-type column can be displayed by using TO_CHAR(). The syntax is:

SELECT Roll, DOB FROM Student;
SELECT Roll, TO_CHAR(DOB, 'DD-MON-YYYY HH:MI:SS A.M.') FROM Student;

Selecting Specific Records

Specific records can be selected by using a WHERE clause with the SELECT statement. The syntax is:
SELECT columns FROM tablename WHERE *cond*^{n}**;**

*SELECT * FROM Student WHERE city= 'Bhubaneswar';*

Roll	Name	City	Age	CGPA
101	Ram	Bhubaneswar	19	9.0
102	Hari	Bhubaneswar		6.7

Operators used in WHERE condition

Relational Operators

=	ex: CGPA=9.0
>	ex: Age>20
<	ex: Age<20
>=	ex: Age>=20
<=	ex: Age<=20
<> or !=	ex: Name !='Hari'
ANY	ex: Age > ANY(20,23,19)
ALL	ex: Age > ALL(20,18)

Logical Operators

AND	ex: City='Bhubaneswar' AND Age=20
OR	ex: City ='Bhubaneswar' OR Age=20
NOT	ex: NOT(Age=20 OR Age=21)

AND has more precedence than OR

Operators used in WHERE condition...

LIKE Operator

LIKE operator uses wild cards for matching as:
%: represents zero or more characters
_: represents any one character

ex: Name LIKE 'S%'
ex: Name LIKE 'S__'
ex: Name LIKE '%i%'
ex: Name LIKE '_i%'

Special Operators

IN ex: City IN('Delhi','Cuttack','Ranchi')
BETWEEN ex: Age BETWEEN 20 AND 22
IS NULL ex: SELECT Name FROM Student WHERE Age is NULL;

Name	Age
Hari	

Sorting

ORDER BY clause using column name

ORDER BY clause is used to sort records in a table
SELECT columns FROM tablename [WHERE *cond*^{n}]
ORDER BY column [ASC/DESC];

*SELECT * FROM Student ORDER BY Age;*
*SELECT * FROM Student ORDER By CGPA, Age DESC;*
NULL values come at the end of the table in case of ORDER BY clause

ORDER BY clause using column number

Records can be sorted by using the column number
SELECT columns FROM tablename [WHERE *cond*^{n}]
ORDER BY columnno [ASC/DESC];
*SELECT * FROM Student ORDER BY 3;*

ALTER Statement

Stud (roll, name, age)

Column	NULL?	Datatype
ROLL		NUMBER(6)
NAME		VARCHAR2(20)
AGE		NUMBER(2)

Adding a New Column

ALTER TABLE tablename ADD(column definition);

ALTER TABLE Stud ADD (address number(20));

Column	NULL?	Datatype
ROLL		NUMBER(6)
NAME		VARCHAR2(20)
AGE		NUMBER(2)
ADDRESS		NUMBER(20)

ALTER Statement...

Modifying an Existing Column

ALTER TABLE tablename MODIFY(column definition);

ALTER TABLE Stud MODIFY(address varchar2(20));

Column	NULL?	Datatype
ROLL		NUMBER(6)
NAME		VARCHAR2(20)
AGE		NUMBER(2)
ADDRESS		VARCHAR2(20)

Dropping a Column

ALTER TABLE tablename DROP COLUMN columnname;

ALTER TABLE Stud DROP COLUMN address;

Column	NULL?	Datatype
ROLL		NUMBER(6)
NAME		VARCHAR2(20)
AGE		NUMBER(2)

ALTER Statement...

Renaming a Column

ALTER TABLE tablename RENAME COLUMN oldname to newname;

ALTER TABLE Stud RENAME COLUMN roll to id;

Column	NULL?	Datatype
ID		NUMBER(6)
NAME		VARCHAR2(20)
AGE		NUMBER(2)

UPDATE Statement

Roll	Name	Age	Branch
101	Vikas	19	
102	Soheb	20	
103	Gita	18	
104	Monalisa	19	
105	Ganesh	20	

UPDATE Statement

UPDATE tablename SET columnname=value [WHERE *cond*^{n}];

UPDATE Stud SET Branch='CSE' WHERE Roll=101;

Roll	Name	Age	Branch
101	Vikas	19	CSE
102	Soheb	20	
103	Gita	18	
104	Monalisa	19	
105	Ganesh	20	

UPDATE Statement...

UPDATE Stud SET Branch='CSE';

Roll	Name	Age	Branch
101	Vikas	19	CSE
102	Soheb	20	CSE
103	Gita	18	CSE
104	Monalisa	19	CSE
105	Ganesh	20	CSE

DROP Statement

DROP command

DROP TABLE tablename;

DROP TABLE Stud;

TRUNCATE Statement

TRUNCATE command

TRUNCATE TABLE tablename;

TRUNCATE TABLE Stud;

DELETE Statement

DELETE command

DELETE FROM tablename [WHERE *cond*^{n}];

DELETE FROM Stud WHERE Roll=101;

DELETE FROM Stud;

RENAME Statement

RENAME command

RENAME oldname TO newname;

RENAME Stud TO Student;

Viewing User Tables

Viewing all user Objects

```
SELECT * FROM TAB;
```

Viewing all user Tables

```
SELECT table_name FROM user_tables;
```

Creating Table from another Table

Creating Table from another Table

CREATE TABLE tablename(column1,column2) AS SELECT column1,column2 FROM tablename;

CREATE TABLE Person(Roll, Name, Age) AS SELECT Roll, Name, Age FROM Stud;

The SQL statement populates the target table with data from the source table

Inserting data into a Table from another Table

Inserting data into a Table from another Table

INSERT INTO tablename SELECT column1, column2 FROM tablename[WHERE *cond*^{n}];

INSERT INTO Person SELECT Roll, Name, Age FROM Stud WHERE Roll=101;

INSERT INTO Person SELECT Roll, Name, Age FROM Stud;

ROWID

- Each row has a unique ROWID
- It is an 18-bit number and represented as a base-64 number
- It contains the physical address of a row in a database

In case user has inputted same records more than one time, ROWID is used to distinguish each record

For example, consider Customer table

Cid	CName	Address
1	Akash	BBS
2	Amir	BBS
2	Amir	BBS
3	Ashok	CTC

ROWID...

If the user wants to delete the duplicate copies of the same record, then ROWID is used

SELECT ROWID, Cid FROM Customer;

ROWID	Cid
AAAF4YAABAAAHCKAAA	1
AAAF4YAABAAAHCKAAB	2
AAAF4YAABAAAHCKAAC	2
AAAF4YAABAAAHCKAAD	3

DELETE FROM Customer WHERE ROWID= 'AAAF4YAABAAAHCKAAC';

Database Systems Laboratory 3

Constraints

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Constraints

Constraints

SQL Constraints

Constraints enforce rules on tables. Constraints can be imposed to the database tables either with the CREATE or ALTER command. Whenever a DML operation is to be performed on a table, the specified constraint must be satisfied for the operation to succeed

Naming a Constraint

A constraint can be identified by an internal or user-defined name. For a user's account, each constraint name must be unique. The standard convention for naming constraint is : **<table name>_<column name>_<constraint type>**

The abbreviation for different constraint types are: *pk* for PRIMARY Key, *fk* for FOREIGN Key, *uk* for UNIQUE, *chk* or *ck* for CHECK and *nn* for NOT NULL constraint
If you do not name a constraint, then the server will generate a name for it by using *SYS_Cn* format

Constraints...

Defining a Constraint

Constraint can be defined in either of the two ways:

- **Column level**:
  - A column- level constraint references a single column and is defined along with the definition of the column
  - This type of constraint is applied to the current column only
  - **column datatype [CONSTRAINT constraint_name] constraint_type**
- **Table level**:
  - A table- level constraint references one or more columns and is defined separately from the definitions of the columns
  - Except the NOT NULL constraint, all other constraints can be defined at the table level
  - **[CONSTRAINT constraint_name] constraint_type (column,..)**

Normally, simple keys are defined at the column level and composite keys are defined at the table level

NOT NULL Constraint

NOT NULL Constraint

It ensures that the column has a value and the value is not a NULL value
It prevents a column from accepting NULL values. The syntax is:
columnname datatype(size) NOT NULL or

columnname datatype(size) CONSTRAINT constraintname NOT NULL
It can only be applied at column level

name VARCHAR(20) CONSTRAINT student_name_nn NOT NULL

NOT NULL Constraint...

Let the structure of ITEM_MASTER table is:

Column	Type	Size
Item_no	NUMBER	4
Name	VARCHAR2	20
Qty_on_hand	NUMBER	5
Category	CHAR	1
Unit_measure	CHAR	4
Reorder_Lvl	NUMBER	5
Reorder_qty	NUMBER	5
Rate	NUMBER	8,2

CREATE TABLE ITEM_MASTER(Item_no NUMBER(4), Name VARCHAR2(20), Qty_on_hand NUMBER(5), Category CHAR(1), Unit_measure CHAR(4), Reorder_Lvl NUMBER(5), Reorder_qty NUMBER(5), Rate NUMBER(8,2));

Let the Item_no,Reorder_lvl, Reorder_qty and Rate columns are NOT NULL

NOT NULL Constraint...

CREATE TABLE ITEM_MASTER(Item_no NUMBER(4) NOT NULL, Name VARCHAR2(20), Qty_on_hand NUMBER(5), Category CHAR(1), Unit_measure CHAR(4), Reorder_Lvl NUMBER(5) NOT NULL, Reorder_qty NUMBER(5) NOT NULL, Rate NUMBER(8,2) NOT NULL);

CREATE TABLE ITEM_MASTER(Item_no NUMBER(4) CONSTRAINT c1 NOT NULL, Name VARCHAR2(20), Qty_on_hand NUMBER(5), Category CHAR(1), Unit_measure CHAR(4), Reorder_Lvl NUMBER(5) CONSTRAINT c2 NOT NULL, Reorder_qty NUMBER(5) CONSTRAINT c3 NOT NULL, Rate NUMBER(8,2) CONSTRAINT c4 NOT NULL);

Dropping NOT NULL Constraint

A NOT NULL constraint can be dropped by executing **ALTER TABLE tablename DROP CONSTRAINT constraintname;**

ALTER TABLE ITEM_MASTER DROP CONSTRAINT c4;

Unique Constraint

Unique Constraint

It ensures every value in a column or set of columns be unique
The unique constraint allows NULL values. The syntax is:

Column level: **Columnname datatype(size) UNIQUE** or

Columnname datatype(size) CONSTRAINT constraintname UNIQUE

Table level: **CONSTRAINT constraintname UNIQUE(columns)**

mob_no NUMBER(10) CONSTRAINT student_mob_uk UNIQUE

CONSTRAINT student_mob_uk UNIQUE(mob_no)

Unique Constraint...

Let the Name column in ITEM_MASTER table is unique:

CREATE TABLE ITEM_MASTER(Item_no NUMBER(4) NOT NULL, Name VARCHAR2(20) UNIQUE, Qty_on_hand NUMBER(5), Category CHAR(1), Unit_measure CHAR(4), Reorder_Lvl NUMBER(5) NOT NULL, Reorder_qty NUMBER(5) NOT NULL, Rate NUMBER(8,2) NOT NULL);

CREATE TABLE ITEM_MASTER(Item_no NUMBER(4) NOT NULL, Name VARCHAR2(20) , Qty_on_hand NUMBER(5), Category CHAR(1), Unit_measure CHAR(4), Reorder_Lvl NUMBER(5) NOT NULL, Reorder_qty NUMBER(5) NOT NULL, Rate NUMBER(8,2) NOT NULL), CONSTRAINT ce3 UNIQUE(Name) ;

Unique Constraint...

The composite unique key constraint can be defined only at the table level by specifying column names separated by a comma within parentheses

CONSTRAINT student_name_city_uk UNIQUE(name, city)

CREATE TABLE ITEM_MASTER(Item_no NUMBER(4) NOT NULL, Name VARCHAR2(20) , Qty_on_hand NUMBER(5), Category CHAR(1), Unit_measure CHAR(4), Reorder_Lvl NUMBER(5) NOT NULL, Reorder_qty NUMBER(5) NOT NULL, Rate NUMBER(8,2) NOT NULL), CONSTRAINT ce4 UNIQUE(Item_no,Name) ;

Dealing with UNIQUE Constraint in an existing table

The syntax for adding unique constraint is:
ALTER TABLE tablename ADD CONSTRAINT constraintname UNIQUE(columns);

ALTER TABLE ITEM_MASTER ADD CONSTRAINT C4 UNIQUE(Name);

The syntax for dropping unique constraint is:
ALTER TABLE tablename DROP CONSTRAINT constraintname;

ALTER TABLE ITEM_MASTER DROP CONSTRAINT C4;

Primary Key Constraint

Primary Key Constraint

Primary key constraint is also known as the entity integrity constraint
A table can have at most one primary key constraint
PRIMARY key is equivalent to the combination of NOT NULL constraint and UNIQUE constraint

Column level: **Columnname datatype(size) PRIMARY KEY** or

Columnname datatype(size) CONSTRAINT constraintname PRIMARY KEY

Table level: **CONSTRAINT constraintname PRIMARY KEY(columns)**

roll number(6) CONSTRAINT student_roll_pk PRIMARY KEY

CONSTRAINT student_roll_pk PRIMARY KEY(roll)

Primary Key Constraint...

Let the Item_no column in ITEM_MASTER table is primary key:

CREATE TABLE ITEM_MASTER(Item_no NUMBER(4) PRIMARY KEY, Name VARCHAR2(20) UNIQUE, Qty_on_hand NUMBER(5), Category CHAR(1), Unit_measure CHAR(4), Reorder_Lvl NUMBER(5) NOT NULL, Reorder_qty NUMBER(5) NOT NULL, Rate NUMBER(8,2) NOT NULL);

CREATE TABLE ITEM_MASTER(Item_no NUMBER(4), Name VARCHAR2(20) UNIQUE, Qty_on_hand NUMBER(5), Category CHAR(1), Unit_measure CHAR(4), Reorder_Lvl NUMBER(5) NOT NULL, Reorder_qty NUMBER(5) NOT NULL, Rate NUMBER(8,2) NOT NULL, CONSTRAINT C7 PRIMARY KEY(Item_no));

Dealing with Primary Key Constraint in an existing table

The syntax for adding Primary key constraint is:
ALTER TABLE tablename ADD CONSTRAINT constraintname PRIMARY KEY(columns);

ALTER TABLE ITEM_MASTER ADD CONSTRAINT C5 PRIMARY KEY(Item_no);

The syntax for dropping Primary key constraint is:
ALTER TABLE tablename DROP PRIMARY KEY [CASCADE];

ALTER TABLE ITEM_MASTER DROP PRIMARY KEY; or

ALTER TABLE ITEM_MASTER DROP CONSTRAINT C5;

Foreign Key Constraint

Foreign Key Constraint

It is also known as the referential integrity constraint. It establishes a relationship with the primary key of the same or another table. Foreign key and the referenced primary key columns need not have the same name, but the data type, size and domain must match

Column level: **Columnname datatype(size) [CONSTRAINT constraintname] REFERENCES tablename(columns)** or **Columnname datatype(size) [CONSTRAINT constraintname] REFERENCES tablename**

Table level: **CONSTRAINT constraintname FOREIGN KEY(columns) REFERENCES tablename(columns)**

fid VARCHAR(6) CONSTRAINT student_fid_fk REFERENCES faculty(fid)

CONSTRAINT student_fid_fk FOREIGN KEY(fid) REFERENCES faculty(fid)

Foreign Key Constraint...

ON DELETE CASCADE

This option can be added to allow deletion of a record in the parent table and deletion of the dependent records in the child table implicitly

Coulmn level: **Columnname datatype(size) [CONSTRAINT constraintname] REFERENCES tablename(columns) [ON DELETE CASCADE]**

Table level: **CONSTRAINT constraintname FOREIGN KEY(columns) REFERENCES tablename(columns) [ON DELETE CASCADE]**

fid VARCHAR(6) CONSTRAINT student_fid_fk REFERENCES faculty(fid) ON DELETE CASCADE

CONSTRAINT student_fid_fk FOREIGN KEY(fid) REFERENCES faculty(fid) ON DELETE CASCADE

Foreign Key Constraint...

Let ITEM_TRANS be the table where It_no references to the Item_no column in ITEM_MASTER table

Column	Type	Size
It_no	NUMBER	4
Trans_date	DATE	
qty	NUMBER	5

CREATE TABLE ITEM_TRANS(It_no NUMBER(4) REFERENCES ITEM_MASTER(Item_no), trans_date DATE, qty NUMBER(5));

Dealing with Foreign Key Constraint in an existing table

The syntax for adding Foreign key constraint is:
ALTER TABLE tablename ADD CONSTRAINT constraintname FOREIGN KEY(columns) REFERENCES tablename(columns);

ALTER TABLE ITEM_TRANS ADD CONSTRAINT C7 FOREIGN KEY(Item_no) REFERENCES ITEM_MASTER(Item_no);

The syntax for dropping Foreign key constraint is:
ALTER TABLE tablename DROP CONSTRAINT constraintname;

ALTER TABLE ITEM_TRANS DROP CONSTRAINT C7;

Check Constraint

Check Constraint

It defines a condition that every row must satisfy. There can be more than one CHECK constraint on a column

Column level: **Columnname datatype(size) CONSTRAINT constraintname CHECK(condition)**

Table level: **CONSTRAINT constraintname CHECK(condition)**

age NUMBER(2) CONSTRAINT student_age_chk CHECK((age>=15) AND (age<=50))

CONSTRAINT student_age_chk CHECK((age>=15) AND (age<=50))

name VARCHAR(20) CONSTRAINT student_name_nn CHECK(name is NOT NULL)

Check Constraint...

CREATE TABLE ITEM_MASTER(Item_no NUMBER(4) PRIMARY KEY, Name VARCHAR2(20) UNIQUE, Qty_on_hand NUMBER(5), Category CHAR(1) CHECK(Category in('A', 'B', 'C'), Unit_measure CHAR(4), Reorder_Lvl NUMBER(5) NOT NULL, Reorder_qty NUMBER(5) NOT NULL, Rate NUMBER(8,2) NOT NULL);

CREATE TABLE ITEM_MASTER(Item_no NUMBER(4) PRIMARY KEY, Name VARCHAR2(20) UNIQUE, Qty_on_hand NUMBER(5), Category CHAR(1) NOT NULL, Unit_measure CHAR(4), Reorder_Lvl NUMBER(5) NOT NULL, Reorder_qty NUMBER(5) NOT NULL, Rate NUMBER(8,2) NOT NULL, CHECK((Category='A' AND Rate<=1000) OR (Category='B' AND Rate<=4500) OR (Category='C' AND Rate>=4500)));

Dealing with Check Constraint in an existing table

The syntax for adding Check constraint is:
ALTER TABLE tablename ADD CONSTRAINT constraintname CHECK (condition);

ALTER TABLE ITEM_TRANS ADD CONSTRAINT C8 CHECK(Category in('A', 'B', 'C'));

The syntax for dropping Check constraint is:
ALTER TABLE tablename DROP CONSTRAINT constraintname;

ALTER TABLE ITEM_TRANS DROP CONSTRAINT C8;

DEFAULT Value

DEFAULT Value

It ensures that a particular column will always have a value when a new record is inserted. The default value gets overwritten if a user enters another value. The default value is used if a NULL value is inserted. The DEFAULT value is defined in the column level. The syntax is:

Columnname datatype(size) DEFAULT value

CREATE TABLE ITEM_MASTER(Item_no NUMBER(4) PRIMARY KEY, Name VARCHAR2(20) UNIQUE, Qty_on_hand NUMBER(5) DEFAULT 100, Category CHAR(1), Unit_measure CHAR(4), Reorder_Lvl NUMBER(5) NOT NULL, Reorder_qty NUMBER(5) NOT NULL, Rate NUMBER(8,2) NOT NULL);

DEFAULT Value...

If a column level constraint is defined on the column with a default value, then the default value must precede the constraint. The syntax is:

Columnname datatype(size) DEFAULT value constraint definition

Qty_on_hand NUMBER(5) DEFAULT 100 CHECK (Qty_on_hand>=100),

Viewing USER Constraints

Viewing USER Constraints

User can view all the constraints by excuting
SELECT * FROM USER_CONSTRAINTS;

If the user wants to view all the constraints applied to a single table, the syntax is:
SELECT * FROM USER_CONSTRAINTS WHERE TABLE_NAME= tablename;

*SELECT * FROM USER_CONSTRAINTS WHERE TABLE_NAME='ITEM_MASTER';*

Database Systems Laboratory 4

Built-in Functions

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Built-in Functions

DUAL Table

DUAL Table

DUAL table is a small worktable, which consists of only one column **DUMMY** and a single row with value **X** of VARCHAR2 type

This table is owned by user *SYS* and is available to all users

It is used for Arithmetic calculations and Date retrieval

*SELECT 2*5 FROM DUAL;*

SELECT SYSDATE FROM DUAL;

Employee Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	09-NOV-81	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-SEP-81	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

Built-in Functions

Built-in Functions

The built-in functions provide a powerful tool for the enhancement of a basic query. They serve the purpose of manipulating data items and returning a result. Functions are of two types:

- ***Single-row or Scalar functions***:
  - They work on columns from each row and return one result per row
- ***Group functions or Aggregate functions***:
  - They manipulate data in a group of rows and return single result

Group Functions

The group or aggregate functions perform an operation on a group of rows and return one result. The different aggregate functions are:

COUNT([DISTINCT] column)

This function counts the number of rows without considering NULL values

SELECT COUNT(MGR) FROM EMP;
SELECT COUNT(DISTINCT MGR) FROM EMP;

COUNT(MGR)	COUNT(DISTINCT MGR)
13	6

COUNT(*)

It counts the number of rows including NULL values

SELECT COUNT(*) FROM EMP;

COUNT(*)
14

Group Functions...

SUM([DISTINCT] column)

It finds the sum of all values in a column ignoring the NULL values

SELECT SUM(SAL) FROM EMP;

SUM(SAL)
29055

AVG([DISTINCT] column)

It finds the average of all values in a column ignoring the NULL values

SELECT AVG(SAL) FROM EMP;

AVG(SAL)
2075.35

Group Functions...

MAX([DISTINCT] column)

It finds the maximum value in the column ignoring the NULL values

SELECT MAX(SAL) FROM EMP;

MAX(SAL)
5000

MIN([DISTINCT] column)

It finds the minimum value in the column ignoring the NULL values

SELECT MIN(SAL) FROM EMP;

MIN(SAL)
800

Scalar Functions

Scalar Functions

These functions act on one value at a time. There are various types of scalar functions:

- ***Date functions***: These functions take a date value or date-type column as argument and return date-type data
- ***Numeric functions***: These functions take a number or number-type column as argument and return a numeric value
- ***Character functions***: These functions take a character string or character-type column as argument and return a character or numeric value
- ***Conversion functions***: These functions are used to convert value from one data type to another
- ***Misc. functions***: These functions perform some specific tasks

Date Functions

The date values are stored internally with day, month, year, hour, minute and second information. The different date functions are:

SYSDATE

It is the pseudo column that returns the system's current date

SELECT SYSDATE FROM DUAL;

SYSDATE
21-JAN-13

ADD_MONTHS(date, n)

It adds calendar months to a date

SELECT ADD_MONTHS(HIREDATE, 4) FROM EMP WHERE EMP_NO=7369;

ADD_MONTHS(HIREDATE,4)
17-APR-81

Date Functions...

LAST_DAY(date)

It returns the last day of the month

SELECT LAST_DAY(SYSDATE) FROM DUAL;

LAST_DAY(SYSDATE)
31-JAN-13

MONTHS_BETWEEN(date1, date2)

It finds the number of months between two dates

SELECT MONTHS_BETWEEN(SYSDATE,'23-JAN-89') FROM DUAL;

MONTHS_BETWEEN(SYSDATE,'23-JAN-89')
287.90

Date Functions...

NEXT_DAY(date, 'day')

It finds the next occurrence of a day from the given date

SELECT NEXT_DAY(SYSDATE, 'MONDAY') FROM DUAL;

NEXT_DAY(SYSDATE, 'MONDAY')
28-JAN-13

EXTRACT(YEAR/MONTH/DAY FROM date)

This extracts the year, month, or day from a date value

SELECT EXTRACT(MONTH FROM SYSDATE) FROM DUAL;

EXTRACT(MONTH FROM SYSDATE)
1

SELECT EXTRACT(YEAR FROM SYSDATE) FROM DUAL;

EXTRACT(YEAR FROM SYSDATE)
2013

Numeric Functions

These functions take numeric values and return a numeric value. The different functions in this category are:

ABS(n)

It returns the absolute value of n

SELECT ABS(5), ABS(-100) FROM DUAL;

ABS(5)	ABS(-100)
5	100

CEIL(n)

This returns the smallest integer greater than or equal to the given value

SELECT CEIL(-5.2), CEIL(5.7) FROM DUAL;

CEIL(-5.2)	CEIL(5.7)
-5	6

Numeric Functions...

FLOOR(n)

This returns the largest integer less than or equal to the given value

SELECT FLOOR(-5.2), FLOOR(5.7) FROM DUAL;

FLOOR(-5.2)	FLOOR(5.7)
-6	5

EXP(n)

It returns the exponent e raised to power n

SELECT EXP(5) FROM DUAL;

EXP(5)
148.413159

Numeric Functions...

LN(n)

It returns the natural logarithm of n

SELECT LN(2) FROM DUAL;

LN(2)
0.693147181

LOG(b, n)

It returns $log_b n$ value

SELECT LOG(4,10) FROM DUAL;

LOG(4,10)
1.66096405

MOD(n, m)

It returns the integer remainder of n/m

SELECT MOD(15,4) FROM DUAL;

MOD(15,4)
3

Numeric Functions...

POWER(m, n)

It returns m raised to power n

SELECT POWER(4,3) FROM DUAL;

POWER(4,3)
64

SQRT(n)

It returns the square root of the number n

SELECT SQRT(25) FROM DUAL;

SQRT(25)
5

SELECT SQRT(-25) FROM DUAL;

ORA-01428: argument '-25' is out of range

Numeric Functions...

ROUND(m, [n])

It returns m, rounded to n places to the right of a decimal point

SELECT ROUND(15.19,1), ROUND(15.19) FROM DUAL;

ROUND(15.19,1)	ROUND(15.19)
15.2	15

TRUNC(m, n)

It returns the truncated value of m up to n positions

SELECT TRUNC(15.19,1) FROM DUAL;

TRUNC(15.19,1)
15.1

Numeric Functions...

SIGN(n)

It returns the sign of number n: -1 for negative, 0 for zero, 1 for positive

SELECT SIGN(-8.5) FROM DUAL;

SIGN(-8.5)
-1

SIN(n)

It returns sine of n, where n is in radian

SELECT SIN(60), SIN(1.047167) FROM DUAL;

SIN(60)	SIN(1.047167)
-0.3048106	0.8660

Other trigonometric functions are: *COS(n), TAN(n), SINH(n), COSH(n), TANH(n)*

Character Functions

These functions work on character values. The different types of character functions are:

CHR(n)

It returns the ASCII character corresponding to the integer n

SELECT CHR(70) FROM DUAL;

CHR(70)
F

CONCAT(s1, s2)

It joins the first string to the second string. It is similar to the || operator

SELECT CONCAT('RAM','KRISHNA'), 'RAM'||'KRISHNA' FROM DUAL;

CONCAT('RAM','KRISHNA')	'RAM'||'KRISHNA'
RAMKRISHNA	RAMKRISHNA

Character Functions...

LPAD(s, n, c)

It pads the string s with the character c to the left to a total width of n

SELECT LPAD('ORACLE',10,'') FROM DUAL;*

LPAD('ORACLE',10,'*')
****ORACLE

RPAD(s, n, c)

It pads the string s with the character c to the right to a total width of n

SELECT RPAD('ORACLE',10,'') FROM DUAL;*

RPAD('ORACLE',10,'*')
ORACLE****

Character Functions...

INITCAP(s)

It returns the string with capitalization of the first letter in each word

SELECT INITCAP('HELLO') FROM DUAL;

INITCAP('HELLO')
Hello

SELECT INITCAP(E_NAME) FROM EMP;

LOWER(s)

It converts each letter to lowercase

SELECT LOWER('HELLO') FROM DUAL;

LOWER('HELLO')
hello

Character Functions...

UPPER(s)

It converts each letter to uppercase

SELECT UPPER('HeLLo') FROM DUAL;

UPPER('HeLLo')
HELLO

LTRIM(s, c)

It trims the string s from the left when the characters specified, c, is present in s

SELECT LTRIM(E_NAME,'S') FROM EMP;

RTRIM(s, c)

It trims the string s from the right when the characters specified, c, is present in s

SELECT RTRIM(E_NAME,'I') FROM EMP;

Character Functions...

REPLACE(s, s1, s2)

It returns the string s with the replacement of s2 in place of s1

SELECT REPLACE('ORACLE','RAC','V') FROM DUAL;

REPLACE('ORACLE','RAC','V')
OVLE

SUBSTR(s, n, m)

It returns a substring, starting at character position n, and returns m number of characters

SELECT SUBSTR('DATABASE',3,2) FROM DUAL;

SUBSTR('DATABASE',3,2)
TA

Character Functions...

LENGTH(s)

It returns the number of characters present in the string s

SELECT LENGTH('ORACLE') FROM DUAL;

LENGTH('ORACLE')
6

SOUNDEX(s)

It compares words that are spell differently, but sound alike

SELECT E_NAME FROM EMP WHERE SOUNDEX(E_NAME)=SOUNDEX('KEEING');

E_NAME
KING

Conversion Functions

These functions convert data from one data type to another. The different conversion functions are:

TO_NUMBER(char [,format])

It converts a character value with valid digits to a number using the given format

SELECT SUM(SAL) FROM EMP;
SELECT SUM(TO_NUMBER(SAL)) FROM EMP;

TO_DATE(char [,format])

It converts a character value to date value based on the format provided

SELECT TO_DATE('January 7, 1988','month dd, yyyy') FROM DUAL;

TO_DATE('January 7, 1988','month dd, yyyy')
07-JAN-88

Conversion Functions...

TO_CHAR(number [,format])

It converts a number to a VARCHAR value based on the format provided. *0* is used for compulsory purpose and *9* is used for optional purpose

SELECT TO_CHAR(17145,'$999,999') FROM DUAL;

TO_CHAR(17145,'$999,999')
$17,145

SELECT TO_CHAR(17145,'$000,000') FROM DUAL;

TO_CHAR(17145,'$000,000')
$017,145

Conversion Functions...

TO_CHAR(date [,format])

It converts a date to a VARCHAR value based on the format provided

SELECT TO_CHAR(HIREDATE,'MONTH DD, YYYY') FROM EMP WHERE EMP_NO=7566;

TO_CHAR(HIREDATE,'MONTH DD, YYYY')
APRIL 02, 1981

Use of *TH* in Date formatting

It converts a date to a VARCHAR value based on *TH* format

SELECT HIREDATE, TO_CHAR(HIREDATE,'DDTH-MON-YY') FROM EMP WHERE DEPT_NO=10;

HIREDATE	TO_CHAR(HIREDATE,'DDTH-MON-YY')
09-JUN-81	09TH-JUN-81
17-NOV-81	17TH-NOV-81
23-JAN-82	23RD-JAN-82

Conversion Functions...

Use of *SP* in Date formatting

It converts a date to a VARCHAR value with the spelling

SELECT TO_CHAR(HIREDATE,'DDSP-MON-YY') FROM EMP WHERE DEPT_NO=10;

TO_CHAR(HIREDATE,'DDSP-MON-YY')
NINE-JUN-81
SEVENTEEN-NOV-81
TWENTY-THREE-JAN-82

Use of *SPTH* in Date formatting

It converts a date to a VARCHAR value with the spelling and *TH* format

SELECT TO_CHAR(HIREDATE,'DDSPTH-MON-YY') FROM EMP WHERE DEPT_NO=10;

TO_CHAR(HIREDATE,'DDSPTH-MON-YY')
NINTH-JUN-81
SEVENTEENTH-NOV-81
TWENTY-THIRD-JAN-82

Misc. Functions

Two important functions to deal with NULL value are:

NVL(column, value)

It converts a NULL value to an actual value supplied as an argument. For numerical values, it accepts 0; whereas for character values, it accepts a fixed string

SELECT E_NAME, NVL(COMMISSION, 0) COMMISSIONFROM EMP;
SELECT E_NAME, SALARY+NVL(COMMISSION, 0) Total SalaryFROM EMP;

NVL2(column, notnullvalue, nullvalue)

It checks for NULL as well as not NULL values. If the column has a not NULL value, the second parameter is displayed. If the column has a NULL value, the third parameter is displayed

SELECT E_NAME, NVL2(COMMISSION, 'YES', 'NO') FROM EMP;

Database Systems Laboratory 5

Grouping, DCL & TCL

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Grouping, DCL & TCL

Grouping Data

GROUP BY clause

GROUP BY clause is used for grouping data. The column appearing in SELECT statement must also appear in GROUP BY clause. The syntax of this is:
SELECT column, groupfunction(column) FROM tablename GROUP BY column;

SELECT DEPT_NO FROM EMP GROUP BY DEPT_NO;

DEPT_NO
30
20
10

SELECT DEPT_NO, AVG(SAL) FROM EMP GROUP BY DEPT_NO;

DEPT_NO	AVG(SAL)
30	1566.66666666666666666666666666666667
20	2175
10	2926.66666666666666666666666666666667

Grouping Data...

HAVING Clause

HAVING clause can restrict groups. The syntax of this is:

SELECT column, groupfunction(column) FROM tablename GROUP BY column [HAVING groupcondition];

SELECT DEPT_NO, AVG(SAL) FROM EMP GROUP BY DEPT_NO HAVING AVG(SAL)>2000;

DEPT_NO	AVG(SAL)
20	2175
10	2926.66666666666666666666666666666667

Grouping Data...

WHERE and GROUP BY Clause

The data can be restricted before the grouping by using WHERE clause. The syntax of this is:

SELECT column, groupfunction(column) FROM tablename [WHERE condition] GROUP BY column;

SELECT DEPT_NO, AVG(SAL) FROM EMP WHERE SAL>2000 GROUP BY DEPT_NO;

DEPT_NO	AVG(SAL)
30	2850
20	2991.6666666666666666666666666666666667
10	3740

Grouping Data...

WHERE, GROUP BY and HAVING Clause

WHERE clause is used to filter the unwanted records, while the HAVING clause is used to filter the unwanted groups. The syntax for it is:

SELECT column, groupfunction(column) FROM tablename [WHERE condition] GROUP BY column [HAVING groupcondition];

SELECT DEPT_NO, AVG(SAL) FROM EMP WHERE SAL>2000 GROUP BY DEPT_NO HAVING AVG(SAL)<3000;

DEPT_NO	AVG(SAL)
30	2850
20	2991.6666666666666666666666666666666667

Data Control Language

An object privilege specifies what a user can do with a database object, such as a table or a view. The different privileges for table are: ALTER, INSERT, UPDATE, DELETE, and SELECT

Granting Privileges

A user can grant privileges on objects from own schema to other users or roles by using GRANT command. The syntax of providing a privilege is:
GRANT privileges/ ALL ON objectname TO username/ PUBLIC [WITH GRANT OPTION];
WITH GRANT OPTION clause allows the grantee to grant privileges to other users and roles

GRANT SELECT, INSERT ON Employee TO Mita;

GRANT SELECT, INSERT ON Employee TO Mita WITH GRANT OPTION;

Data Control Language...

*SELECT * FROM SYSTEM.Employee;*

GRANT SELECT ON SYSTEM.Employee TO Mithun;

Revoking Privileges

If a user granted privileges by a WITH GRANT OPTION to another user and that second user passed on those privileges, the REVOKE statement takes privileges not only from the grantee but also from the users granted privileges by the grantee. The general syntax is:
REVOKE privileges/ALL ON objectname FROM username/ PUBLIC;

REVOKE INSERT ON Employee FROM Mita;

REVOKE ALL ON Employee FROM Mita;

Transaction Control Language

A transaction consists of a sequence of query and update statements. Transaction Control Language gives you flexibility to undo transactions or write transactions to the disk. Transactions provide consistency in case of a system failure

Committing a transaction

COMMIT statement is used to end the current transaction and makes permanent any changes made during transaction. The general syntax is:
COMMIT;

Roll backing the operations

ROLLBACK statement is used to discard parts or all of the work the user has done in the current transaction. The syntax for this is:
ROLLBACK;

Transaction Control Language...

Roll backing the operations to a particular position

SAVEPOINT allows the user to create logical marking in the whole transaction, so that the system will discard all the changes up to a point. The syntax for creating a SAVEPOINT is:
SAVEPOINT savepointname;

The syntax for roll backing to a particular savepoint is:
ROLLBACK TO SAVEPOINT savepointname;

SQL*PLUS Commands

SAVE filename.sql [REPLACE/ APPEND]

It saves current buffer contents to a file

GET filename.sql

It retrieves previously saved file to the buffer

START filename.sql

It runs a previously saved command from file

@filename.sql

Same as START command

The filename in the file-related commands requires entire file path

SQL*PLUS Commands...

EDIT

It opens the current buffer named *afiedt.buf*

EDIT filename.sql

It allows to edit a saved file

PASSWORD

It allows to change the password

CREATE USER username IDENTIFIED BY password

It creates a new user with respective password

EXIT

It leaves SQL *PLUS environment & commits current transaction

Database Systems Laboratory 6

Join & Set Operators

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Join & Set Operators

Cross Join

Cross Join

Cross join is used to combine information from any two relations

SELECT Department.deptno, location, ename, eid FROM Employee, Department;

SELECT d.deptno, d.location, e.ename, e.eid FROM Employee e, Department d;

*SELECT * FROM Employee CROSS JOIN Department;*

Join

Join

When the required data are present in more than one table, related tables are joined using a join condition. The join condition combines a row in one table with a row in another table based on the same values in the common columns

Tables are joined on columns that have the same datatype and data width in the tables. Join operation joins two relations by merging those tuples from two relations that satisfy a given condition

In broad level, Joins are of three categories:

- Inner Join
- Outer Join
- Self Join

Inner Join

In the Inner Join, tuples with NULL valued join attributes do not appear in the result. Tuples with NULL values in the join attributes are also eliminated. The different types of inner join are:

Theta Join

Theta join is a join with a specified condition involving a column from each table. The syntax of theta join is:
SELECT columns FROM tables WHERE join_condition;

When columns are retrieved from more than one table, the use of a table name qualifier in front of the column name tells the server to retrieve that column from the specified table

SELECT e.eid, e.ename, e.sal, s.bonus FROM Employee e, Salgrades s WHERE e.comm>s.bonus;

Inner Join...

Equi Join

Equi join is a special kind of theta join where the join condition is based on the equality operation

SELECT empno, ename, Employee.deptno, dname FROM Employee, Department WHERE Employee.deptno = Department.deptno;

Natural Join

Natural join is possible between two tables only when they contain at least one common attribute. It is just like the equi join with the elimination of the common attributes. The syntax of natural join is:
SELECT columns FROM table1 NATURAL JOIN table2;

SELECT ename, sal, deptno FROM Employee NATURAL JOIN Department;

Outer Join

Outer join is an extension of the natural join operation to deal with the missing information. The different types of outer join are:

Left Outer Join

Left outer join preserves all rows in left table even though there is no matching tuples present in the right table. The syntax of left outer join is:
SELECT columns FROM table1 LEFT OUTER JOIN table2 USING(column); or
SELECT columns FROM table1 LEFT OUTER JOIN table2 ON table1.column= table2.column;

*SELECT * FROM Employee LEFT OUTER JOIN Department USING(deptno);*
*SELECT * FROM Employee LEFT OUTER JOIN Department ON Employee.deptno= Department.deptno;*
SELECT deptno, location, ename * FROM Employee, Department WHERE Employee.deptno= Department.deptno(+);

Outer Join...

Right Outer Join

Right outer join preserves all rows in right table even though there is no matching tuples present in left table. The syntax of right outer join is:
SELECT columns FROM table1 RIGHT OUTER JOIN table2 USING(column); or
SELECT columns FROM table1 RIGHT OUTER JOIN table2 ON table1.column= table2.column;

*SELECT * FROM Employee RIGHT OUTER JOIN Department USING(deptno);*
*SELECT * FROM Employee RIGHT OUTER JOIN Department ON Employee.deptno= Department.deptno;*
SELECT deptno, location, ename * FROM Employee, Department WHERE Employee.deptno(+)= Department.deptno;

Outer Join...

Full Outer Join

Full outer join preserves all records in both tables. The syntax of full outer join is:
SELECT columns FROM table1 FULL OUTER JOIN table2 USING(column); or

SELECT columns FROM table1 FULL OUTER JOIN table2 ON table1.column= table2.column;

*SELECT * FROM Employee FULL OUTER JOIN Department USING(deptno);*

*SELECT * FROM Employee FULL OUTER JOIN Department ON Employee.deptno= Department.deptno;*

Self Join

Self Join

Self join is similar to the theta join. It joins a relation to itself by a condition. When a table is joined to itself, two copies of the same table are used. The syntax for this is:
SELECT columns FROM table T1, table T2 WHERE T1.column operator T2.column;

SELECT e.ename AS ëmployee", m.ename AS managerFROM Employee m, Employee e WHERE e.mgr=m.empno;

Set Operators

To perform the set operations such as UNION, DIFFERENCE and INTERSECTION, the relations need to be union compatible for the result to be a valid relation. The different set operators are:

UNION

Union is used to combine data from two relations

SELECT name, job FROM Employee WHERE dept=20 UNION
SELECT name, job FROM Employee WHERE dept=30;

Set Operators...

DIFFERENCE

Difference is used to identify the rows that are in one relation and not in another

SELECT name, job FROM Employee WHERE dept=20 MINUS SELECT name, job FROM Employee WHERE dept=30;

INTERSECTION

Intersection is used to identify the rows that are common to two relations

SELECT name, job FROM Employee WHERE dept=20 INTERSECT SELECT name, job FROM Employee WHERE dept=30;

Database Systems
Laboratory 7
Sub-Query

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Sub-Query

Sub Query

Sub Query

Subquery is a SELECT statement that is nested within another SELECT statement, which returns intermediate results

It is useful when you need to select rows from a table with a condition that depends on the data in another table. Here, inner query evaluates first, generates values that are tested in the condition of the outer query. Subqueries are of two types:

- ***Single-row subquery***: It is a subquery that returns only one row of data
- ***Multiple-row subquery***: It is a subquery that returns more than one row of data

The subtypes of subqueries are:

- Multiple column Subqueries
- Correlated subqueries

Single-row Subquery

Single-row Subquery

Single-row subquery returns zero or one row to the outer SQL statement. The general syntax of this is:
SELECT columns FROM table1 WHERE column operator (SELECT column FROM table2 WHERE condition);

Let, *Student (roll,name,age, deptno)* and *Department (deptno, dname, campus)* be two schemas. If the user wants to find the roll and name of students belongs to 'CSE' department, the query is:

SELECT roll, name FROM Student WHERE deptno=(SELECT deptno FROM Department WHERE dname='CSE');

Single-row Subquery...

Single-row Subquery used in HAVING clause

Subquery can be used in place of the value in the HAVING clause

Q. Find the department and the departmental average age which are less than the maximum departmental age

SELECT deptno, AVG(age) FROM Student GROUP BY deptno HAVING AVG(age)<(SELECT MAX(AVG(age)) FROM Student GROUP BY deptno);

Single-row Subquery...

Single-row Subquery used in FROM clause

The inner SELECT statement in the FROM clause is used as the data source for the outer SELECT statement. This is also called as Inline View

SELECT age FROM (SELECT age FROM Student WHERE age<20);

A subquery may not use the ORDER BY clause, but an inline view may

SELECT age FROM (SELECT age FROM Student ORDER BY age DESC) WHERE ROWNUM<=2;

Multiple-row Subquery

Multiple-row Subquery

Multiple-row subquery returns more than one row. It uses either IN, ALL or ANY operator

SELECT roll, name FROM Student WHERE age< ANY(SELECT age FROM Customer);

SELECT roll, name FROM Student WHERE age< IN(SELECT age FROM Customer);

SELECT roll, name FROM Student WHERE age< ALL(SELECT age FROM Customer);

Multiple-column Subquery

Multiple-column Subquery

Multiple-column subquery returns more than one column as result

SELECT roll,deptno,age FROM Student WHERE (deptno,age) IN (SELECT deptno,MIN(age) FROM Student Group BY deptno);

Correlated Subquery

Correlated Subquery

In a correlated subquery, the inner query can reference columns from the outer query. The inner query is executed once for each row in the outer query

SELECT roll, deptno, name, age FROM Student s1 WHERE age>(SELECT AVG(age) FROM Student s2 WHERE s1.deptno = s2.deptno);

*SELECT * FROM Catalog WHERE Catalog.bookid IN(SELECT bookid FROM Order_Details WHERE Catalog.bookid= Order_Details.bookid);*

Nested Subquery

Nested Subquery

You can nest subqueries to a depth of 255

SELECT empid, empname, salary, hiredate FROM Employee WHERE empid IN(SELECT empid FROM Employee WHERE mgrid = (SELECT empid FROM Employee WHERE empname = 'Paresh'));

SELECT deptno,AVG(age) FROM Student GROUP BY deptno HAVING AVG(age)<(SELECT MAX(AVG(age)) FROM Student WHERE deptno IN (SELECT deptno FROM Staff WHERE qty>10) GROUP BY deptno);

UPDATE with Subquery

UPDATE with Subquery

The column values can be updated with the help of a subquery statement. The general syntax is:
UPDATE tablename SET columnname = value WHERE columnnane operator (SELECT subquery);

The value in the update statement can also be calculated by using a subquery. The general syntax for this is:
UPDATE tablename SET columnname = (SELECT subquery) [WHERE condition];

Update the age of the student having roll no. 4 to the average age of all the students

UPDATE Student SET age=(SELECT AVG(age) FROM Student) WHERE roll=4;

DELETE with Subquery

DELETE with Subquery

A row or rows from a table can be deleted based on a value returned by a subquery. The general syntax is:
DELETE FROM tablename WHERE columnname = (SELECT subquery);

Delete all the student records whose ages are greater than the average age of the students

DELETE FROM Student WHERE age>(SELECT AVG(age) FROM Student);

TOP- N Analysis

TOP- N Analysis

TOP-N queries are used to sort rows in a table and then to find the first- N largest or first- N smallest values

TOP-N query uses an order by clause. The sorted rows are numbered with a pseudocolumn named ROWNUM

SELECT roll, name FROM (SELECT roll, name FROM student ORDER BY roll DESC) WHERE ROWNUM<= 4;

Database Systems
Laboratory 8
View, Sequence, Synonym & Index

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

View, Sequence, Synonym & Index

View

View

View is an object which gives the user a logical view of data from an underlying table or tables

You can restrict what users can view by allowing them to see only a few columns from a table

When a view is created from more than one table, the user can view data from the view without using join conditions and complex conditions

Views also hide the names of the underlying tables

View is stored as a SELECT statement in the Data Dictionary
View contains no data of its own Any updation of rows in the table will automatically reflect in the views

A query fired on a view will run slower than a query fired on a base table

View...

View Types

Views are of two types:

- ***Simple view***: It is based on one table. It allows data manipulation
- ***Complex view***: It is based on one or more tables. It doesn't allow data manipulation

View Creation

The syntax for creating a view is:
CREATE [OR REPLACE] VIEW viewname AS SELECT statement;

*CREATE VIEW Debts AS SELECT * FROM BILLS;*

View...

Creating View from another View

Similar to the view creation from tables, a view can be created from a previously created view

*CREATE VIEW Credit_dbt AS SELECT * FROM Debts WHERE account_id=4;*

Displaying the content of a view

User can display the content from a view as:
SELECT * FROM viewname;

*SELECT * FROM Debts;*

Inserting into table through view

User can insert records into the underlying table through the view as:
INSERT INTO viewname VALUES(val1, val2..);

INSERT INTO Debts VALUES(val1,val2,..);

View...

Describing a view structure

View structure can be described as:
DESCRIBE viewname;

DESCRIBE Debts;

Updating table data through view

User can update the records of a table through the view as:
UPDATE viewname SET columnname=newvalue[WHERE *cond*^{n}**];**

*UPDATE Debts SET PRICE=PRICE*1.1 WHERE PUBLISHER='MGH';*

Deleting records from table through view

User can delete records from table through view as:
DELETE FROM viewname [WHERE *cond*^{n}**];**

DELETE FROM Debts WHERE condn;

View...

Creating a View WITH CHECK OPTION Constraint

This constraint applies to the WHERE clause condition in the subquery. It allows insertion and updation of rows based on the condition that satisfies the view

*CREATE VIEW Asd AS SELECT * FROM Products WHERE price < 15;*

*CREATE VIEW Psd AS SELECT * FROM Products WHERE price < 15 WITH CHECK OPTION CONSTRAINT con1;*

Creating a View WITH READ ONLY Constraint

This option is used to make sure that the data in the underlying table are not changed through the view

*CREATE VIEW Cheap_products_view3 AS SELECT * FROM Products WHERE price < 15 WITH READ ONLY CONSTRAINT cheap_products_view3_read_only;*

View...

Viewing all the user views

All user created views can be displayed as:
SELECT * FROM USER_VIEWS;

Removing a View

A view can be removed as:
DROP VIEW viewname;

DROP VIEW Debts;

View...

Altering a View

When the underlying table is altered, the view becomes invalid. Thus, the view requires the recompilation as:
ALTER VIEW viewname COMPILE;

ALTER VIEW Debts COMPILE;

ALTER VIEW statement lets you add or remove constraints to or from a view

ALTER VIEW Psd DROP CONSTRAINT con1;

View...

Use of GROUP BY clause

GROUP By clause can be used with view creation

CREATE OR REPLACE VIEW Vn (empno, noincr, amount) AS SELECT emp_no, COUNT(*), SUM(amt) FROM INCR GROUP BY emp_no;

CREATE VIEW Pr AS SELECT product_type_id, AVG(price) average_price FROM Products WHERE price < 15 GROUP BY product_type_id HAVING AVG(price) > 13;

Complex View

Complex View

It is based on one or more tables. It doesn't allow data manipulation

In complex view, you can use only SELECT statement

CREATE VIEW Vw AS SELECT P.name, PT.type,P.price FROM PRODUCTS P NATURAL JOIN PRODUCTTYPE PT;

Advantages of Views

Advantages of Views

Some of the major advantages of using views are:

- Views allow in setting up different security levels for the same base table, thus protecting certain data from people who do not have proper authority
- The views allow the same data to be seen by different users in different ways at the same time
- Views can be used to hide complex queries

Sequence

Sequence

Sequence is used to generate a sequence of numbers. The value generated can have a maximum of 38 digits

The minimum information required to generate numbers using a sequence are:

- The starting number
- The maximum number
- The increment value

The syntax for creating a sequence is:
CREATE SEQUENCE seqname INCREMENT BY n START WITH s MAXVALUE m1 \ NOMAXVALUE MINVALUE m2 \ NOMINVALUE [CYCLE \ NOCYCLE] [CACHE c \ NOCACHE];

Sequence...

CREATE SEQUENCE sq INCREMENT BY 1 START WITH 100 MAXVALUE 999 NOCACHE;

CURRVAL & NEXTVAL pseudocolumns

NEXTVAL column returns the next available number in the sequence

CURRVAL column gives the current sequence value

NEXTVAL must be used at least once to get the value from CURRVAL

SELECT sq.CURRVAL FROM DUAL;
SELECT sq.NEXTVAL FROM DUAL;
INSERT INTO emp(eid) VALUES (sq.NEXTVAL);

Sequence...

Viewing the details of a user sequences

SELECT sequence_name, last_number, max_value, min_value, increment_by FROM USER_SEQUENCES;

Modifying a Sequence

Modification of a sequence does not allow you to change the START WITH option. Similarly, the maximum value cannot be set to a number less than the current number
ALTER SEQUENCE seqname INCREMENT BY n MAXVALUE m1 \ NOMAXVALUE MINVALUE m2 \ NOMINVALUE [CYCLE \ NOCYCLE] [CACHE c \ NOCACHE];

Dropping a Sequence

A sequence can be dropped as:
DROP SEQUENCE sequencename;

DROP SEQUENCE sq;

Synonym

Synonym

Synonyms are used to create alternate names for tables, views, sequences ... etc. The syntax for this is:
CREATE [PUBLIC] SYNONYM synname FOR objectname;

CREATE SYNONYM Emp FOR Employee;

CREATE SYNONYM Cstd FOR Customer_Details;
*SELECT * FROM Cstd;*

Viewing the details of a user synonyms

SELECT synonym_name, table_name, table_owner FROM USER_SYNONYMS;

Dropping a Synonym

A Synonym can be dropped as:
DROP SYNONYM synonymname;

DROP SYNONYM Emp;

Index

Index is used for faster retrieval of rows from a table. It can be used implicitly or explicitly. Mainly, index is of two types:

Simple Index

It is created on a single column. The syntax is:
CREATE INDEX indexname ON tablename(column);

CREATE INDEX idx ON Student (cgpa);

Complex Index

It is created on more than one column. The syntax is:
CREATE INDEX indexname ON tablename(columns);

CREATE INDEX ids ON Student (first, last);

Index...

Viewing the details of a user-defined index

SELECT index_name, table_name FROM USER_INDEXES;

SELECT index_name, table_name FROM USER_INDEXES WHERE table_name= 'Student';

Rebuilding an Index

When a table goes through changes, it is advisable to rebuild indexes based on that table. The syntax is:
ALTER INDEX indexname REBUILD;

ALTER INDEX ids REBUILD;