# SQL Notes and Examples

* a transform oriented language - designed to use relations to transform inputs into required outputs
* first developed under the name SEQUEL at IBM San Jose research facilities as the data manipulation language for their prototype relational database, System R
* first commercial DBMS that supported SQL was Oracle in 1979
* renamed SQL in 1980
* the most common language for relational systems
* accepted by ANSI and ISO
* ANSI standards first published in 1986
* SQL-2011 is the current standard
* implemented on mainframes, minis and PC's

### Components:

1. A Data Definition Language (DDL) for defining the database structure
2. A Data Manipulation Language (DML) for retrieving and updating data
3. A Data Control Language (DCL) to control the database and to apply security to the tables and columns of the database
4. A Transaction Control Language (TCL) to manage database transactions

* SQL contains only these definitional and manipulative commands; it does not contain flow control commands (i.e. no if..then..else, do…while)

SQL is a relatively easy language to learn:

* it is a non-procedural language: you specify what information you require, rather than how to get it;
* it is essentially free-format, which means that parts of statements do not have to be typed at particular locations on the screen;
* the command structure consists of standard English words such as CREATE TABLE, INSERT, SELECT;
* it can be used by a range of users including Database Administrators (DBA), management personnel, application programmers and many other types of end-users.

DDL, DML, DCL and the database development process:

#### DDL

Define the database:

CREATE tables, indexes, views

Establish foreign keys

Drop or truncate tables

#### DML

Load the database:

INSERT data

UPDATE the database

Manipulate the database:

SELECT tables

#### DCL

Control the database:

GRANT, ADD, REVOKE

Physical Design

Implementation

Maintenance

#### TCL

Manage Transactions:

ROLLBACK, COMMIT, SAVEPOINT

Types of DML’s

1. based on relational Calculus

* based on predicate calculus
* proposed by Codd as Alpha language, but never implemented

1. SQL (transform oriented)
2. based on relational algebra

* theoretical way to manipulate a relational database
* operators act on relations to produce new relations (like addition acting on numbers to produce new numbers)
* retrieving data involves using relational operators to form new relations containing the desired data

1. Query by Example (QBE)

* developed by M.M. Zloof at IBM Yorktown Heights Research lab
* users enter their request by filling in portions of the displayed tables

# Relational Algebra

E. F. Codd suggest two theoretical relational languages to use with the relational model:

1. *Relational algebra*, a procedural language
2. *Relational calculus*, a nonprocedural language

Third-generation high-level compiler languages :

* can be used to manipulate data in a table
* can only work with **one** row at a time

Relational languages

* can work on the entire table or a group of rows
* without a looping structure.

can be embedded in other host languages for more processing capability

SQL is a ***nonprocedural relational language***.

## Defining a Database in SQL using SQL DDL

### Oracle DDL Statements:

**CREATE DATABASE**

**ALTER DATABASE**

**DROP DATABASE**

**CREATE DEFAULT**

**DROP DEFAULT**

**CREATE INDEX**

**CREATE PROCEDURE**

**ALTER PROCEDURE**

**DROP PROCEDURE**

**CREATE RULE**

**DROP RULE**

**CREATE TABLE**

**ALTER TABLE**

**DROP TABLE**

**CREATE TRIGGER**

**ALTER TRIGGER**

**DROP TRIGGER**

**CREATE VIEW**

### Creating Tables:

The Create Table statement

* creates a table
* creates all the table columns with appropriate datatype and size
* defines the constraints for each column

### Column Constraints:

* Constraints tell Oracle to apply certain rules to values being entered into a column and to reject records that don't satisfy the rules.
* Constraints include PRIMARY KEY, FOREIGN KEY, UNIQUE, NULL, NOT NULL, DEFAULT and CHECK COLUMN.

Example:

To create the tables in the Chazy database (see data model on next page)

CREATE TABLE computer (

compid char(4) PRIMARY KEY NOT NULL,

mfgname varchar2(20) NOT NULL,

mfgmodel char(5) NOT NULL,

proctype varchar2(2) null );

CREATE TABLE employee (

empnum number(3) PRIMARY KEY,

empname varchar2 (20) NOT NULL,

empphone char(4) NULL );

CREATE TABLE package (

packid char(4) PRIMARY KEY,

packname varchar2(20),

packver number(4,2) NULL,

packtype varchar2(25) NOT NULL,

packcost number (6,2) NOT NULL );

CREATE TABLE pc (

tagnum number(5) PRIMARY KEY,

location varchar2(20) NOT NULL,

empnum number(3),

compid char(4),

FOREIGN KEY (empnum) REFERENCES employee (empnum),

FOREIGN KEY (compid) REFERENCES computer(compid)

);

CREATE TABLE software (

packid char(4) NOT NULL,

tagnum number(5) NOT NULL,

instdate date DEFAULT sysdate,

softcost number (6,2) NOT NULL,

PRIMARY KEY (packid, tagnum),

FOREIGN KEY (packid) REFERENCES package (packid),

FOREIGN KEY (tagnum) REFERENCES pc (tagnum)

);

# 

# The CHAZY Database

## Physical Data Model (Database Schema)

## Chazy Tables:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **COMPUTER** | | | |  | **EMPLOYEE** | | |
| **COMPID** | **MFGNAME** | **MFGMODEL** | **PROCTYPE** |  | **EMPNUM** | **EMPNAME** | **EMPPHONE** |
| B121  B221  C007  M759 | Bantam  Bantam  Cody  Lemmin | 48X  48D  DI  GRL | 486DX  486DX2  486DX  486SX |  | 124  567  611 | Alvarez, Ramon  Feinstein, Betty  Dinh, Melissa | 1212  8716  2963 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PC** | | | | |
| **TAGNUM** | **COMPID** | **EMPNUM** | **LOCATION** | |
| 32808  37691  57772  59836  77740 | M759  B121  C007  B221  M759 | 611  124  567  124  567 | Accounting  Sales  Info Systems  Home  Home |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PACKAGE** | | | | |
| **PACKID** | **PACKNAME** | **PACKVER** | **PACKTYPE** | **PACKCOST** |
| AC01  DB32  DB33  SS11  WP08  WP09 | Boise Accounting  Manta  Manta  Limitless View  Words & More  Freeware Processing | 3.00  1.50  2.10  5.30  2.00  4.27 | Accounting  Database  Database  Spreadsheet  Word Processing  Word Processing | 725.83  380.00  430.18  217.95  185.00  30.00 |

**SOFTWARE**

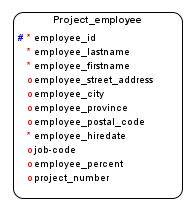
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PACKID** | **TAGNUM** | **INSTDATE** | **SOFTCOST** | |
| AC01  DB32  DB32  DB33  WP08  WP08  WP08  WP09  WP09 | | 32808  32808  37691  57772  32808  37691  57772  59836  77740 | 09/13/95  12/03/95  06/15/95  05/27/95  01/12/96  06/15/95  05/27/95  10/30/95  05/27/95 | 754.95  380.00  380.00  412.77  185.00  227.50  170.24  35.00  35.00 |

## Converting a Logical Data Model to a Relational Database

***Guidelines***:

1. Every entity becomes a table (relation).
2. Define entity integrity via the primary key keyword.
3. Define referential integrity via the foreign key keyword.
4. Use the check keyword to define data integrity.
5. Assign defaults where appropriate.

***Example:***



Create the **project\_employee** table. Note the following constraints:

* 1. the employee\_hiredate defaults to the current date
  2. the job\_code is between 500 and 600
  3. the project\_code is the primary key of the project table
  4. the employee\_province defaults to "QC"
  5. the postal code should be validated

CREATE TABLE Project\_employee

(

employee\_id NUMBER (5) NOT NULL PRIMARY KEY,

employee\_lastname VARCHAR2 (50) NOT NULL ,

employee\_firstname VARCHAR2 (20) NOT NULL ,

employee\_street\_address VARCHAR2 (40) ,

employee\_city VARCHAR2 (25) ,

employee\_province CHAR (2) DEFAULT 'QC',

employee\_postal\_code CHAR (6)

CHECK (EMPLOYEE\_POSTAL\_CODE LIKE '[A-Z][0-9][A-Z][0-9][A-Z][0-9]'),

employee\_hiredate DATE NOT NULL,

job\_code CHAR (3)

CHECK (JOB\_CODE BETWEEN '500' AND '600'),

employee\_percent NUMBER (4,2) ,

project\_number CHAR (3),

FOREIGN KEY (project\_number) REFERENCES PROJECT (project\_number)

) ;

SQL DML

* the basic form of an SQL command (also called an SQL query) is:

|  |
| --- |
| SELECT ... <columns>  FROM ... <tables>  WHERE ... <restrictions> |

* the columns we wish to see printed are listed after the word SELECT
* the tables in which these columns are found are listed after the word FROM
* the conditions defining which rows we wish to view are listed after the word WHERE

*Examples using CHAZY database*

1. To find the name of employee 124:

SELECT EMPNAME

FROM EMPLOYEE

WHERE EMPNUM = 124;

*Result:*

|  |
| --- |
| **EMPNAME** |
| Alvarez, Ramon |

2. To find the number and name of employee whose phone number is 2963:

SELECT EMPNUM, EMPNAME

FROM EMPLOYEE

WHERE EMPPHONE = 2963;

*Result:*

|  |  |
| --- | --- |
| **EMPNUM** | **EMPNAME** |
| 611 | Dinh, Melissa |

3. To list all columns in a table, use \* for columns:

SELECT \*

FROM EMPLOYEE;

*Result:*

|  |  |  |
| --- | --- | --- |
| **EMPNUM** | **EMPNAME** | **EMPPHONE** |
| 124 | Alvarez, Ramon | 1212 |
| 567 | Feinstein, Betty | 8716 |
| 611 | Dinh, Melissa | 2963 |

## Customizing Column Headings

In Oracle, there are two ways to display a column heading in a query that is different from the column name:

* "heading"
* column\_name AS [heading]

*Example:*

SELECT EMPNAME "Employee Name", EMPHONE AS "Phone Number"

FROM EMPLOYEE

*Result:*

|  |  |
| --- | --- |
| **Employee Name** | **Phone Number** |
| Alvarez, Ramon | 1212 |
| Feinstein, Betty | 8716 |
| Dinh, Melissa | 2963 |

Single-Table Queries

## The WHERE Clause

### Simple Conditions

|  |  |
| --- | --- |
| **Comparison Operator** | **Meaning** |
| =  <  >  <=  >=  != or <> or ^= | Equal to  Less than  Greater than  Less than or equal to  Greater than or equal to  Not equal to |

*Example:*

To list the package id and name of all the database packages:

SELECT PACKID, PACKNAME

FROM PACKAGE

WHERE ‘Database’;

*Result:*

|  |  |
| --- | --- |
| **PACKID** | **PACKNAME** |
| DB32 | Manta |
| DB33 | Manta |

### Logical Operators

- operators AND, OR, NOT

*Examples:*

1. To list all the database packages that cost more than $400:

SELECT PACKID, PACKNAME

FROM PACKAGE

WHERE PACKTYPE = ‘Database’

AND PACKCOST > 400;

*Result:*

|  |  |
| --- | --- |
| **PACKID** | **PACKNAME** |
| DB33 | Manta |

2. To list all the packages that are either a database or cost more than $400:

SELECT PACKNAME

FROM PACKAGE

WHERE PACKTYPE = ‘Database’

OR PACKCOST > 400;

*Result:*

|  |
| --- |
| PACKNAME |
| Boise Accounting |
| Manta |
| Manta |

3. To list all the packages that are not databases:

SELECT PACKNAME

FROM PACKAGE

WHERE NOT (PACKTYPE = ‘Database’);

*Result:*

|  |
| --- |
| PACKNAME |
| Boise Accounting |
| Limitless View |
| Words & More |
| Freeware Processing |

### 

### Use of BETWEEN

Instead of coding:

SELECT PACKID, PACKNAME, PACKCOST

FROM PACKAGE

WHERE PACKCOST >= 200

AND PACKCOST <= 400

to list all the packages whose cost is between 200 and 400, you can code:

SELECT PACKID, PACKNAME, PACKCOST

FROM PACKAGE

WHERE PACKCOST BETWEEN 200 AND 400;

*Result:*

|  |  |  |
| --- | --- | --- |
| PACKID | PACKNAME | PACKCOST |
| DB32 | Manta | 380.00 |
| SS11 | Limitless View | 217.95 |

### Use of IN

*Example:*

To list all packages that are databases or spreadsheets or word processing:

SELECT PACKID, PACKNAME, PACKTYPE

FROM PACKAGE

WHERE PACKTYPE IN

(‘Database’, ‘Spreadsheet’, ‘Word Processing’);

*Result:*

|  |  |  |
| --- | --- | --- |
| PACKID | PACKNAME | PACKTYPE |
| DB32 | Manta | Database |
| DB33 | Manta | Database |
| SS11 | Limitless View | Spreadsheet |
| WP08 | Words & More | Word Processing |
| WP09 | Freeware Processing | Word Processing |

### Use of LIKE

- provides for wildcard (%) in comparison

*Examples:*

To list all packages that contain an ampersand in their name

SELECT PACKID, PACKNAME

FROM PACKAGE

WHERE PACKNAME LIKE ‘%&%’;

*Result:*

|  |  |
| --- | --- |
| PACKID | PACKNAME |
| WP08 | Words & More |

### Use of Computed Columns

- allows you to show the results of a calculation in a query

*Examples:*

1. To list the package id, name and 90% of the package cost for all packages:

SELECT PACKID, PACKNAME,

(.90 \* PACKCOST) “Reduced Price”

FROM PACKAGE

*Result:*

|  |  |  |
| --- | --- | --- |
| PACKID | PACKNAME | Reduced Price |
| AC01 | Boise Accounting | 653.247 |
| DB32 | Manta | 342 |
| DB33 | Manta | 387.162 |
| SS11 | Limitless View | 196.155 |
| WP08 | Words & More | 175.5 |
| WP09 | Freeware Processing | 36 |

2. To list the package id, name and 90% of the package cost for all packages where 90% of the cost is less than or equal 200:

SELECT PACKID, PACKNAME,

(.90 \* PACKCOST) “Reduced Price”

FROM PACKAGE

WHERE (.90 \* PACKCOST) <= 200;

*Result:*

|  |  |  |
| --- | --- | --- |
| PACKID | PACKNAME | Reduced Price |
| SS11 | Limitless View | 196.155 |
| WP08 | Words & More | 175.5 |
| WP09 | Freeware Processing | 36 |

### NULLS

*Example:*

To select the number and name of all employees who do not have a phone number:

SELECT EMPNUM, EMPNAME

FROM EMPLOYEE

WHERE EMPPHONE IS NULL;

*Result:*

|  |  |
| --- | --- |
| **EMPNUM** | **EMPNAME** |
|  |  |

## Sorting

* order of rows is immaterial (from definition of a relational database)
* to display rows in a particular sequence, use the ORDER BY clause

*Examples:*

To list the employee number, name and phone number in employee name sequence:

SELECT EMPNUM, EMPNAME, EMPPHONE

FROM EMPLOYEE

ORDER BY EMPNAME;

*Result:*

|  |  |  |
| --- | --- | --- |
| EMPNUM | EMPNAME | EMPPHONE |
| 124 | Alvarez, Ramon | 1212 |
| 611 | Dinh, Melissa | 2963 |
| 567 | Feinstein, Betty | 8716 |

To list the package, name, type, and cost in descending cost order within type:

SELECT PACKID, PACKNAME, PACKTYPE, PACKCOST

FROM PACKAGE

ORDER BY PACKTYPE, PACKCOST DESC;

*Result:*

|  |  |  |  |
| --- | --- | --- | --- |
| PACKID | PACKNAME | PACKTYPE | PACKCOST |
| AC01 | Boise Accounting | Accounting | $725.83 |
| DB33 | Manta | Database | $430.18 |
| DB32 | Manta | Database | $380.00 |
| SS11 | Limitless View | Spreadsheet | $217.95 |
| WP08 | Words & More | Word Processing | $195.00 |
| WP09 | Freeware Processing | Word Processing | $40.00 |

## Built-In Functions

|  |  |
| --- | --- |
| **Built-in Function** | **Meaning** |
| COUNT | Count of the number of rows satisfying the WHERE clause |
| SUM | Sum of the values in a column for all rows satisfying the WHERE clause (column must be numeric) |
| AVG | Average of the values in a column for all rows satisfying the WHERE clause (column must be numeric) |
| MAX | Largest value in a column for all rows satisfying the WHERE clause (If column is numeric, will be largest number. If not, will be highest entry based on collating sequence.) |
| MIN | Smallest value in a column for all rows satisfying the WHERE clause (If column is numeric, will be smallest number. If not, will be highest entry based on collating sequence.) |

*Examples:*

1. To list the number of databases in the package table:

SELECT COUNT(\*) “Number of Databases”

FROM PACKAGE

WHERE PACKTYPE = ‘Database’;

*Result:*

|  |
| --- |
| Number of Databases |
| 2 |

2. To list the number of packages and the total cost of all the packages in the package table:

SELECT COUNT(PACKiD) "Number of Packages",

TO\_CHAR(SUM(PACKCOST),’$99,999.99’) "Total Cost"

FROM PACKAGE

*Result:*

|  |  |
| --- | --- |
| Number of Packages | Total Cost |
| 6 | $1,988.96 |

3. To list the number of packages and the average cost of all the packages in the package table:

SELECT COUNT(PACKID) "Number of Packages",

TO\_CHAR(AVG(PACKCOST),’$99,999.99’) "Average Cost"

FROM PACKAGE

*Result:*

|  |  |
| --- | --- |
| Number of Packages | Average Cost |
| 6 | $331.49 |

4. To list the number of packages and the maximum cost of all the packages in the package table:

SELECT COUNT(PACKID) AS "Number of Packages",

TO\_CHAR(MAX(PACKCOST),’$99,999.99’) AS "Maximum Cost"

FROM PACKAGE

*Result:*

|  |  |
| --- | --- |
| Number of Packages | Maximum Cost |
| 6 | $725.83 |

5. To list the number of packages and the minimum cost of all the packages in the package table:

SELECT COUNT(PACKID) "Number of Packages",

TO\_CHAR(MIN(PACKCOST),’$99,999.99’) "Minimum Cost"

FROM PACKAGE

*Result:*

|  |  |
| --- | --- |
| Number of Packages | Minimum Cost |
| 6 | $40.00 |

## Use of DISTINCT

*Examples:*

1. List the employee numbers of all the employees who have PC's

SELECT EMPNUM

FROM PC

*Result:*

|  |
| --- |
| EMPNUM |
| 611 |
| 124 |
| 567 |
| 124 |
| 567 |

*Notice that employee numbers 124 and 567 are listed twice because they each have two PC's. To list each number only once, use the DISTINCT keyword:*

SELECT DISTINCT EMPNUM

FROM PC

*Result:*

|  |
| --- |
| EMPNUM |
| 124 |
| 567 |
| 611 |

2. Count the number of employees who have PC's:

SELECT COUNT(EMPNUM) AS "Number of Employees",

FROM PC

*Result:*

|  |
| --- |
| Number of Employees |
| 5 |

*Notice all the records are counted even though employee numbers 124 and 567 occur twice. To count each number only once, use the DISTINCT keyword in the COUNT expression:*

SELECT COUNT(DISTINCT EMPNUM) AS "Number of Employees"

FROM PC

*Result:*

|  |
| --- |
| Number of Employees |
| 3 |

## Nesting Queries

* places one query inside another
* the inner query is called a **subquery** and is executed first

*Example:*

To list all packages that cost more than the average cost of all database packages:

SELECT PACKID, PACKNAME

FROM PACKAGE

WHERE (PACKCOST >

(SELECT AVG(PACKCOST)

FROM PACKAGE

WHERE PACKTYPE = ‘Database’));

*Result:*

|  |  |
| --- | --- |
| PACKID | PACKNAME |
| AC01 | Boise Accounting |
| DB33 | Manta |

*Operation:*

1. Creates a temporary table to contain result of subquery:

|  |
| --- |
| AVG(PACKCOST) |
| 405.09 |

2. Executes the query:

|  |  |
| --- | --- |
| PACKID | PACKNAME |
| AC01 | Boise Accounting |
| DB33 | Manta |

## Grouping

* allows you to do control-break totals

*Example:*

To list the TAGNUM and total cost of all software installed on the computer with that TAGNUM for each TAGNUM in the SOFTWARE table:

SELECT TAGNUM,

TO\_CHAR(SUM(SOFTCOST),"$9,999.99") "Total Software Cost"

FROM SOFTWARE

GROUP BY TAGNUM

ORDER BY TAGNUM;

*Result:*

|  |  |
| --- | --- |
| TAGNUM | Total Software Cost |
| 32808 | $939.95 |
| 37691 | $227.50 |
| 57772 | $583.01 |
| 59836 | $35.00 |
| 77740 | $35.00 |

## 

## Using Having

* WHERE limits the rows to be listed
* the HAVING clause can limit the *groups* that are included

*Examples:*

* + 1. Count the number of packages of a given type:

SELECT PACKTYPE, COUNT(PACKID) "Number of Packages"

FROM PACKAGE

GROUP BY PACKTYPE;

*Result:*

|  |  |
| --- | --- |
| PACKTYPE | Number of Packages |
| Accounting | 1 |
| Database | 2 |
| Spreadsheet | 1 |
| Word Processing | 2 |

* + 1. List only those packages that have a count > 1

SELECT PACKTYPE, COUNT(PACKID) AS "Number of Packages"

FROM PACKAGE

GROUP BY PACKTYPE

HAVING COUNT(PACKID) > 1;

*Result:*

|  |  |
| --- | --- |
| PACKTYPE | Number of Packages |
| Database | 2 |
| Word Processing | 2 |

* + 1. To list the package type and number of packages of the type that cost more than $150:

SELECT PACKTYPE, COUNT(PACKID) AS "Number of Packages"

FROM PACKAGE

WHERE PACKCOST > 150

GROUP BY PACKTYPE

*Result:*

|  |  |
| --- | --- |
| PACKTYPE | Number of Packages |
| Accounting | 1 |
| Database | 2 |
| Spreadsheet | 1 |
| Word Processing | 1 |

* + 1. To list the package type and number of packages of the type that cost more than $150 for which there is more than one package:

SELECT PACKTYPE, COUNT(PACKID) AS "Number of Packages"

FROM PACKAGE

WHERE PACKCOST > 150

GROUP BY PACKTYPE

HAVING COUNT(PACKID) > 1;

*Result:*

|  |  |
| --- | --- |
| PACKTYPE | Number of Packages |
| Database | 2 |