SQL Data Definition and Manipulation

Data Definition in SQL

Creating Relations is achieved with the command **CREATE TABLE**

For example (here domains and constraints are Oracle-specific):

```
CREATE TABLE  Account
(    accountNo  NUMBER (5)   CONSTRAINT pk_account PRIMARY KEY,
     type  VARCHAR2(10)      CONSTRAINT nn_Type  NOT NULL
                             CONSTRAINT ck-Type
                                    CHECK Type IN ('Current', 'Deposit'),
     balance  NUMBER (20,2)  CONSTRAINT nn_Bal     NOT NULL,
     dateOpened   DATE,
     inBranch  NUMBER (2)    CONSTRAINT nn_InBr   NOT NULL
                             CONSTRAINT fk_Abranch
                                             REFERENCES Branch (branchNo)
);
```

The Most Important Domains in Oracle

Key Slide

CHAR(n)	fixed length strings, up to 2000 bytes
VARCHAR2(n)	variable length strings, max n, up to 4000 bytes
NUMBER(p, s)	numbers with precision p (i.e. number of significant digits) and scale s *precision. p - total number of digits, from 1 to 38* *scale s - number of decimal places, -84 to 127 * (-ve for rounding)*
NUMBER	floating point with precision 38
NUMBER(p)	integer with precision p and scale 0
DATE	dates in various formats *from Jan 1, 4712 BC to Dec 31, 9999 AD*
LONG	up to two gigabytes of character data
RAW(s)	binary data of up to 2000 bytes
LONG RAW	binary data of up 2 gigabytes
ROWID	unique identifier for a record

ANSI standard SQL has the following numeric types which are accepted:
- **NUMERIC(p,s), DECIMAL(p,s), DEC(p,s)**
- **INTEGER, INT** and **SMALLINT**
- **FLOAT, DOUBLE PRECISION** and **REAL**

Constraints in an Oracle Table Specification

The CONSTRAINT clause appears in the **CREATE TABLE** or the

ALTER TABLE command

Constraints come in two flavours:
- **column constraints** limit the values taken by a particular column
- **table constraints** limit the values taken by more than one column

Constraints can be **named** which are helpful in the following cases
- if you wish to alter the constraint in the future
- if you get an error referring to the constraint when entering data, you can be told which constraint is violated

If you do NOT give the constraint a unique name, you don't need the word 'CONSTRAINT' either

Column Constraints

Key Slide

Column constraints determine the values allowed in one column

They can be placed:
- with the column declaration
- or at the end of the column declarations, in the same way as table constraints (next slide)

Examples placed with the column declarations:

These two are identical in effect

```
type        VARCHAR2(10) CONSTRAINT nn_Type NOT NULL,
inBranch  NUMBER          CONSTRAINT fk_InBranch
                                   REFERENCES  branch(branchNo)
salary       NUMBER(8,2)   CONSTRAINT salCheck  (CHECK  salary > 0)
```

Example of a column constraint at the end of the column declarations (i.e. this is now a table constraint and the syntax is slightly different) :

```
CONSTRAINT fk_AccBr  FOREIGN KEY  (inBranch)
                                   REFERENCES  Branch(branchNo)
```

Table Constraints

Key Slide

Table constraints determine the values allowed in a whole row, and are used for composite primary or foreign keys, or for cross checks

They are placed at the end of the column declarations, but still inside the enclosing parentheses

```
CONSTRAINT pk_Owner PRIMARY KEY (accNo, custId)
CONSTRAINT fk_ApptLocation FOREIGN KEY ( HospID, BuildingName)
                            REFERENCES Building( HOSPID, Name )
```

– Note, this constraint is stronger than two separate foreign keys and a direct reference to Hospital is not necessary

Constraints can be **added or removed** using the **ALTER TABLE** command

For instance, in the Bank database, the Branch Manager cannot be constrained to be a foreign key referencing the Employee table, until the Employee table has been created

```
ALTER TABLE Branch
ADD (  CONSTRAINT fk_Manager FOREIGN KEY (manager)
                            REFERENCES Employee(ni#)    )
```

Multi-column Primary Keys

Key Slide

If a table has two (or more) columns in the primary key then you can only assert this with table constraints

– e.g. A table for the weak entity City in the US Geography database

• The key is city name and state name and this **cannot** be done with:

~~CREATE TABLE City~~
~~(name VARCHAR2(30) CONSTRAINT pk1_city PRIMARY KEY,~~
~~state VARCHAR2(30) CONSTRAINT pk1_city PRIMARY KEY,~~

• but must be done with:

```
CREATE TABLE City
(  name VARCHAR2(30),
   state VARCHAR2(30),
   .....
   CONSTRAINT pk_city PRIMARY KEY (name, state)
```

Multi-column Foreign Keys

Key Slide

Similarly, foreign keys into tables with multi-column primary keys must use table constraints

– e.g. A table for the many-to-many relationship city on river in the US Geography database:

• The foreign key to city **cannot** be done with:

~~CREATE TABLE On~~
~~(city VARCHAR2(30) CONSTRAINT fk_city~~
~~REFERENCES City(name),~~
~~state VARCHAR2(30) CONSTRAINT fk_state~~
~~REFERENCES City(state),~~

• since neither of the columns in City is guaranteed unique, but must be done with:

```
CREATE TABLE On
(  city VARCHAR2(30),
   state VARCHAR2(30),
   .....
   CONSTRAINT fk_on FOREIGN KEY (city, state)
                             REFERENCES City( name, state)
```

Altering a Table Specification

DBMSs vary in exactly what **alterations** are permitted.

Oracle permits the table definition to be changed by the **ALTER TABLE** command

Columns and integrity constraints can be added.

```
ALTER TABLE Employee
                    ADD CONSTRAINT  chksal  CHECK(salary > 0);

ALTER TABLE Employee ADD(title VARCHAR2(4));
```

Reducing Table Constraints

The definition of an existing column can only be altered if this change cannot cause loss of data.

- E.g. the length of a column can be increased but not decreased.

```
ALTER TABLE Employee MODIFY (title VARCHAR2(5));
```

It's not possible in Oracle to delete columns.

- You need to copy what's required from the original table, delete the original table, and rename the copied table.
  - Or delete and re-create the table.

Microsoft Access permits these alterations, but displays a dialog box asking the user to confirm their alteration.

View Relations

Key Slide

Views are relations **derived** from **base** relations

They allow access to other tables and views, but contain no data themselves

The data is only stored in the base table/relation

They can be used to

- **restrict access** to a predetermined set of rows and/or columns of a base table. e.g. hiding personal details
- **hide data complexity**. A view acts as a single table when actually several tables are used to construct the result. This is useful if you are frequently using the same group of tables in many queries. e.g. Projects and details of employees who work on them (including names)

Creating Views

Key Slide

Views are created with the command:

```
CREATE VIEW RelName
   (optionally column names i.e. aliases)
     AS the result of a query
```

e.g.

```
CREATE VIEW  MaleEmpView
   AS   SELECT  *  FROM  Employee
          WHERE gender = 'M'
               WITH CHECK OPTION;
```

Updating Through Views

Some systems allow you to explicitly update a view, indirectly changing the underlying table

- e.g. it would be possible to add new male employees by insertion into *MaleEmpView*

This is allowed if there is no ambiguity on what will happen, which is true if:

- the view is created from only one table
- there are no nested selects , no distinct, no group functions, no calculations

The following example is an SQL insert command which could be successfully performed:

```
INSERT INTO MaleEmpView
       VALUES ('111', 'Smith', 'John', ‘M', '1 Beech Walk', 'teacher');
```

Preventing Changes to Views

Inserts and updates performed through the View **must** result in rows that the query can select

– e.g. you should not be allowed to add females to employees through the view *MaleEmpView*

The WITH CHECK OPTION enforces this

```
INSERT INTO MaleEmpView VALUES ('111', 'Smith', 'Joan', 'F', '1
   Beech Walk', 'teacher');
```

This causes:

```
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

Removing and Changing Relations

A base relation or view can be **removed** with the command **DROP**

```
DROP TABLE Project;
DROP VIEW MaleEmpView;
```

Remember that in setting up a file made up of CREATE TABLE commands, it is sensible to start with a sequence of DROP commands to get rid of the previous versions during the early stages of setting up a database

– Although you don’t want to do this after having added the data.

Data Manipulation in SQL

Adding New Tuples

– There are three ways of doing this, only two being available in Oracle SQL:
  - bulk loading – done in *SQLLoader*, not in SQL in Oracle
  - inserting new values one tuple at a time
  - inserting query results

Bulk Loading and Dumping of Data

Clearly, inserting values one tuple at a time is slow, so there is a mechanism for loading many tuples at a time - this is called **bulk loading**

Some DBMS have a **COPY TABLE** command for copying data from/to files

In Oracle the *SQLLoader* loads **empty** tables from data in text files

There are Import and Export utilities for backing up data and moving it between Oracle databases

Inserting Tuples

Key Slide

The INSERT statement does this and has three forms:

1. Inserting all the values:

```
INSERT INTO Employee VALUES (‘GC1234’, ‘Richard’, ……….)
```

2. Inserting some of the values:

```
INSERT INTO Employee (ni#,lName,fName, gender)
                              VALUES (‘GC1234’,‘Richard’,’Cooper’,’M’);
```

If columns are omitted, the column default will be assigned.

3. Inserting the results of a query:

```
INSERT INTO Males
                    SELECT * FROM Employees WHERE gender ='M‘;
```

N.B. This copies the data, like copy table, unlike making a view relation.

this table must already exist!

Deletion

Key Slide

To delete tuples satisfying a condition from some relation:

```
DELETE FROM <relation>   WHERE <condition>;
```

e.g.

```
DELETE FROM Employee WHERE Dnum = 5;
```

To delete all the tuples just leave out the WHERE condition

Note – this does not drop the table!

The Semantics of Deletion

Consider the following:

```
DELETE FROM Employee E1 WHERE EXISTS
    (SELECT * FROM Employee E2
            WHERE E1.Dnum = E2.Dnum AND E1.NI# <> E2.NI#)
```

– which deletes an employee if there are any other employees in the same department (who are not them)
– if every department has at least two employees, does this delete everybody?
– i.e. do we delete the last person from a department, since there should only be that person left if we've got rid of everyone?

Yes we do delete everybody, because the semantics is this:

– deletion proceeds in two stages:
  1. Mark all tuples for which the WHERE condition is satisfied
  2. Delete the marked tuples

Updates

Key Slide

To change certain attributes in certain tuples of a relation:

```
UPDATE <relation>
        SET <list of attribute assignments>
                WHERE <condition on tuples>;
```

e.g. to change one employee's department:

```
UPDATE Employee
        SET Dnum = 6
                WHERE NI# = 1234
```

or to move all department 5 employees to department 6

```
UPDATE Employee
        SET Dnum = 6
                WHERE Dnum = 5
```

Creating Tables as Copies of Data

Key Slide

This could be useful during development, or as a staging post in complex query / report or to accelerate queries by avoiding joins

The format is:

```
CREATE TABLE <relation>   (optionally column names)
             AS <the result of a query>.
```

For example to create a table of just employees:

```
CREATE TABLE MaleEmpBaseTable
             AS    SELECT  *  FROM Employee  WHERE sex= 'M';
```

NOT NULL constraints are copied over. Other constraints need adding

IMPORTANT *: This creates a* ***copy*** *as compared with a* ***view*** *which is merely a reference to other data.*

Use with care – as it introduces redundancy

Dates & Times in Oracle

Although there are functions in Oracle for entering and extracting date and time information, they are not straightforward. Look in the Help if you need more than is here

file:\\fs4/oracle10g_docs/server.101/b10759/functions001.htm#sthref1032

The default format is DD-MON-YY e.g.01-JAN-01
- Use four figures for dates in the last century
- However, the output default is two figures!

Most uses of dates require the use of functions to cast to and from strings:
Example: to find employees born in 1985, use the To_Char function which takes two parameters
1) the attribute with a DATE domain
2) the format that the character output is required in

```
SELECT * FROM  Employee
        WHERE TO_CHAR(DateOfBirth, ‘YYYY’) = ‘1985’;
```

Date Examples

Date comparison
```
SELECT *  FROM Account
  WHERE dateOpened < '01-Jan-1997';
```

Add/subtract number of days from a date (NB **SYSDATE** =‘today’)
```
SELECT  *  FROM Account
  WHERE dateOpened > SYSDATE - 180;
```

Add/subtract number of months to a date
```
SELECT  *  FROM  Account
  WHERE bDate > Add_Months(SYSDATE, - 3);
```

This command returns accounts opened on a particular day of week.
```
SELECT accountno,dateOpened,inBranch
  FROM  Account
   WHERE To_Char(dateOpened,'DY') = 'TUE'
```

To_Date and To_Char

Key Slide

To_Date and *To_Char* cast dates to and from strings

To_Char takes a date and time value and a format and turns the date into a string using the format
- The format can decorate the string with sub-strings such as “AD”, “AM”, punctuation such as “:”
- It also determines which parts of the date or time value is returned – days, months, hours, etc.
- and what format is used –e.g. 12 hour or 24-hour clock
  - e.g. `SELECT TO_CHAR(mydate, 'DD-MON-YYYY HH24:MI:SSxFF')`
  - gives `01-DEC-1999 10:00:00`

To_Date is the converse, e.g.:
```
SELECT TO_DATE( 'January 15, 1989, 11:00 A.M.', 'Month dd, YYYY,
   HH:MI A.M.', 'NLS_DATE_LANGUAGE = American')
```

Time Examples

Times are perhaps easiest stored using 2 digit integers

Otherwise use the DATE datatype. Insert the values by using the To_Date function which needs as parameters
- the time as a string e.g. 17:30 and the format of this string e.g. ‘HH24:MI’

```
INSERT INTO Times  VALUES (TO_DATE('17:30','HH24:MI'));
```

To get a time out again, use the TO_CHAR function to convert the time to a character format
```
SELECT  TO_CHAR(mytime, 'HH24') AS hr FROM  Times;
```
I have renamed the output column in this example.
```
SELECT TO_CHAR(mytime, 'HH24:MI') AS hm FROM Times;
```

Oracle will automatically convert this format to a number if is involved in an arithmetic expression
```
SELECT  (TO_CHAR(mytime, 'HH24')+ 2)  AS TwoLater FROM Times;
```