



Government Girls' Polytechnic, Bilaspur

Name of the Lab: **DBMS Lab**

Practical : **DBMS-II Lab**

Class: **6th Semester (CSE)**

Teachers Assessment: 20 End Semester Examination:50

Experiment no.1

Objective :To Study design of E-R Diagram.

Hardware and system software requirement: HDD 40 GB, 128 RAM and Window xp.

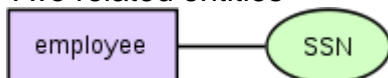
Software requirement: SQL, Window xp.

Theory: An entity-relationship model (ERM) is an abstract and conceptual representation of data. Entity-relationship modeling is a database modeling method, used to produce a type of conceptual schema or semantic data model of a system, often a relational database, and its requirements in a top-down fashion. Diagrams created by this process are called entity-relationship diagrams, ER diagrams, or ERDs.

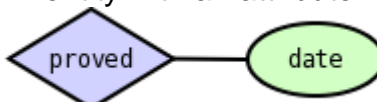
The building blocks: entities, relationships, and attributes



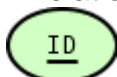
Two related entities



An entity with an attribute



A relationship with an attribute



Primary key

An entity may be defined as a thing which is recognized as being capable of an independent existence and which can be uniquely identified. An entity is an abstraction from the complexities of some domain. When we speak of an entity we normally speak of some aspect of the real world which can be distinguished from other aspects of the real world.

An entity may be a physical object such as a house or a car, an event such as a house sale or a car service, or a concept such as a customer transaction or order. Although the term entity is the one most commonly used, following Chen we should really distinguish between an entity and an entity-type. An entity-type is a category. An entity, strictly speaking, is an instance of a given entity-type. There are usually many instances of an entity-type. Because the term entity-type is somewhat cumbersome, most people tend to use the term entity as a synonym for this term.

Entities can be thought of as nouns. Examples: a computer, an employee, a song, a mathematical theorem.

Entity-Relationship Diagram

Definition: An entity-relationship (ER) diagram is a specialized graphic that illustrates the interrelationships between entities in a database. ER diagrams often use symbols to represent three different types of information. Boxes are commonly used to represent entities. Diamonds are normally used to represent relationships and ovals are used to represent attributes.

Also Known As: ER Diagram, E-R Diagram, entity-relationship model

Examples:

Consider the example of a database that contains information on the residents of a city. The ER diagram shown in the image above contains two entities -- people and cities. There is a single "Lives In" relationship. In our example, due to space constraints, there is only one attribute associated with each entity. People have names and cities have populations. In a real-world example, each one of these would likely have many different attribute

EXPERIMENT NO 2

Objective : To study of creation the tables with primary key, foreign key, normal, unique.

Hardware and system software requirement: HDD 40 GB, 128 RAM and Window xp.

software requirement: SQL, Window xp.

Theory: Keys

key

any subset of a relation is called key.

super key

a key is called super key if it is sufficient to identify a unique tuple of a relation.

candidate key

a minimal super key is called candidate key i.e no proper subset of a candidate key is super key.

primary key

a candidate key chosen as a principal to identify a unique tuple of a relation.

foreign key

a key of a relation which is a primary key of some other relation in the relational schema.

Primary Key Definition

Definition: The primary key of a relational table uniquely identifies each record in the table. It can either be a normal attribute that is guaranteed to be unique (such as Social Security Number in a table with no more than one record per person) or it can be generated by the DBMS (such as a globally unique identifier, or GUID, in Microsoft SQL Server). Primary keys may consist of a single attribute or multiple attributes in combination.

Examples:

Imagine we have a STUDENTS table that contains a record for each student at a university. The student's unique student ID number would be a good choice for a primary key in the STUDENTS table. The student's first and last name would not be a good choice, as there is always the chance that more than one student might have the same name.

Creating a Table With a Primary Key

To create a table that declares attribute a to be a primary key:

```
CREATE TABLE <tableName> (... , a <type> PRIMARY KEY, b, ...);
```

To create a table that declares the set of attributes (a,b,c) to be a primary key:

```
CREATE TABLE <tableName> (<attrs and their types>, PRIMARY KEY (a,b,c));
```

Creates a new table.

Syntax

```

CREATE TABLE
  [ database_name.[ owner ] . | owner. ] table_name
  ( { < column_definition >
    | column_name AS computed_column_expression
    | < table_constraint > } [ ,...n ]
  )

```

Foreign key

a foreign key is a referential constraint between two tables.^[1] The foreign key identifies a column or a set of columns in one (referencing) table that refers to a set of columns in another (referenced) table. The columns in the referencing table must be the primary key or other candidate key in the referenced table. The values in one row of the referencing columns must occur in a single row in the referenced table. Thus, a row in the referencing table cannot contain values that don't exist in the referenced table (except potentially NULL). This way references can be made to link information together and it is an essential part of database normalization. Multiple rows in the referencing table may refer to the same row in the referenced table. Most of the time, it reflects the one (master table, or referenced table) to many (child table, or referencing table) relationship.

Defining Foreign Keys

Foreign keys are defined in the ANSI SQL Standard, through a FOREIGN KEY constraint. The syntax to add such a constraint to an existing table is defined in SQL:2003 as shown below. Omitting the column list in the REFERENCES clause implies that the foreign key shall reference the primary key of the referenced table.

```

ALTER TABLE <table identifier>
  ADD [ CONSTRAINT <constraint identifier> ]
    FOREIGN KEY ( <column expression> {, <column expression>}... )
    REFERENCES <table identifier> [ ( <column expression> {, <column
expression>}... ) ]
    [ ON UPDATE <referential action> ]
    [ ON DELETE <referential action> ]

```

Likewise, foreign keys can be defined as part of the CREATE TABLE SQL statement.

```

CREATE TABLE table_name (
  id INTEGER PRIMARY KEY,
  col2 CHARACTER VARYING(20),
  col3 INTEGER,
  ...
  FOREIGN KEY(col3)
    REFERENCES other_table(key_col) ON DELETE CASCADE,
  ... )

```

If the foreign key is a single column only, the column can be marked as such using the following syntax:

```
CREATE TABLE table_name (
  id INTEGER PRIMARY KEY,
  col2 CHARACTER VARYING(20),
  col3 INTEGER REFERENCES other_table(column_name),
  ... )
```

what is primary key, unique key, foreign key?

Primary key - Primary key means main key

def:- A primary key is one which uniquely identifies a row of a table. this key does not allow null values and also does not allow duplicate values. for ex,

empno	empname	salary
1	firoz	35000
2	basha	34000
3	chintoo	40000

it will not the values as follows:

1	firoz	35000
1	basha	34000
	chintoo	35000

Unique key - single and main key

A unique is one which uniquely identifies a row of a table, but there is a difference like it will not allow duplicate values and it will any number of allow null values(In oracle).

it allows only a single null value(In sql server 2000)

Both will function in a similar way but a slight difference will be there. So, decalaring it as a primary key is the best one.

foreign key - a foreign key is one which will refer to a primary key of another table

for ex,

emp_table			dept_table	
empno	empname	salary	deptno	deptname

In the above relation, deptno is there in emp_table which is a primary key of dept_table. that means, deptno is referring the dept_table.

Experiment no. 3

Objective : Study of Creation tables with integrity constraints.

Hardware and system software requirement: HDD 40 GB, 128 RAM and Window xp.

Software requirement: SQL, Window xp.

Theory: Integrity constraints are used to ensure accuracy and consistency of data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity. There are many types of integrity constraints that play a role in referential integrity.

Types

Codd initially defined two sets of constraints but, in his second version of the relational model, he came up with five integrity constraints:

Entity integrity

The entity integrity constraint states that no primary key value can be null. This is because the primary key value is used to identify individual tuples in a relation . Having null value for the primary key implies that we cannot identify some tuples. This also specifies that there may not be any duplicate entries in primary key column.

Referential Integrity

The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

Domain Integrity

The domain integrity states that every element from a relation should respect the type and restrictions of its corresponding attribute. A type can have a variable length which needs to be respected. Restrictions could be the range of values that the element can have, the default value if none is provided, and if the element can be NULL.

Referential Integrity

Introduction Often, in relational database, we wish to ensure that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation. This condition is called Referential integrity (RI). It is the concept of relationships between tables, based on the definition of a primary key and a foreign key.

The following table includes the important elements of Referential Integrity:

Term	Definition
Child Table	A table, where the referential constraints are defined. A child table is synonymous with the referencing table.
Parent Table	The table being referenced by a Child table. A Parent table is synonymous with the referenced table.
Primary Key	A primary key uniquely identifies a row of a table.
Foreign Key	A foreign key refers to columns in the Child table. A foreign key may consist of up to 16 columns.
Referencing columns	A referencing columns are within a referencing table that are foreign keys for columns in some other referenced table.
Referenced columns	A referenced columns are defined as either primary key columns or unique columns in a referenced table.

In the Teradata RDBMS, there are three choices to implement RI:

- Use the referential constraint checks supplied by the database software.
- Write site-specific macros.
- Enforce constraints through application code.

Teradata RDBMS provides a reliable mechanism to prevent accidental erasure or corruption of data in a database and ensuring data integrity and data consistency.

Importance of Referential Integrity By providing specification of columns within a referencing table that are foreign keys for columns in some other referenced table, referential integrity is a reliable mechanism which prevents accidental database corruptions when doing inserts, updates, and deletes. It states that a row cannot exist in a table with a non-null value for a referencing column if an equal value does not exist in a referenced column.

For example, let's suppose in customer_service database, there is a tuple t1 in the employee_phone relation with t1[employee_number] = '1018', but there is no tuple in the employee relation for the 1018. This situation would be undesirable. We expect the employee relation to list all employee_number. Therefore, tuple t1 would refer to an employee that does not exist. Clearly, we would like to have an integrity constraint that prohibits this sort of situation. We could define employee_number in employee relation as a foreign key in employee_phone relation. Under this situation, employee is the referenced table, employee_name of employee table is the referenced column; employee_phone is the referencing table, employee_name of employee_phone is the referencing column.

Once we define the employee_number as a foreign key in employee_phone relation, if we try to insert a row with a primary key value that does not exist in employee table,

the system will not allow this insertion. This is the way the Teradata RDBMS maintains referential integrity. The following summarize the benefits of referential integrity:

- Ensure data integrity and consistency base on primary key and foreign key
- Increases development productivity, because it is not necessary to code SQL statements to enforce referential constraints, the Teradata RDBMS automatically enforces referential integrity.
- For large database system like the Teradata RDBMS, it is critique to ensure referential integrity.

Defining Referential Constraints Referential constraint provide a means of ensuring that changes made to the database by authorized users do not result in a loss of data consistency. The constraints are in the following forms:

- **Key declarations** The stipulation that certain attributes form a candidate key for a given entity set. The set of legal insertions and updates is constrained to those that do not create two entities with the same value on a candidate key.
- **Form of a relationship** Many to many, one to many, one to one. A one-to-one or one-to-many relationship restricts the set of legal relationships among entities of a collection of entity sets.

So in general, the referential constraint is the combination of the foreign key, the parent key, and the relationship between them.

Proper definition of referential constraints not only allows us to test values inserted in the database, but also permits us to test queries to ensure that the comparisons made make sense. The check clause in the Teradata RDBMS permit data to be restricted in powerful way that most programming language type systems do not permit. Specifically, the check clause permits the schema designer to specify a predicate that must be satisfied by any value assigned to a variable. The referential constraint can be defined by CREATE TABLE statement.

Referential constraints must meet the following criteria:

- The Parent key must exist when the referential constraint is defined, it must be either a unique primary index or a unique secondary index.
- The foreign and parent keys must have the same number of columns and their data types must match, they cannot exceed 16 columns.
- Duplicate referential constraints are not allowed.
- self-reference is allowed, but the foreign and parent keys cannot consist of identical columns.
- There can be no more than 64 referential constraints per table.

Examples of SQL data definition for key declarations

Referential constraints in SQL

Primary and foreign keys can be specified as part of the SQL CREATE TABLE statement:

Key word	Function
Primary key	The primary key clause of the CREATE TABLE statement includes a list of the attributes that constitute the primary key.
Unique	The unique clause of the CREATE TABLE statement includes a list of the attributes that constitute a candidate key.
Foreign key	The foreign key clause of CREATE TABLE statement includes both a list of the attributes that constitute the foreign key and the name of the relation referenced by the foreign key.

Using the partial SQL DDL definition of our customer_service database as an example to illustrate the syntax of primary- and foreign-key declarations.

```
CREATE TABLE employee_phone
    (employee_number INTEGER NOT NULL,
     area_code SMALLINT NOT NULL,
     phone INTEGER NOT NULL,
     extension INTEGER,
     comment_line CHAR(72)
     primary_key(employee_number)
     foreign key(employee_number) references employee)
PRIMARY INDEX (employee_number);
```

This CREATE TABLE statement is used for creating the table employee-phone with the following referential constraints.

- Restricts the employee_number, area_code, phone domains do not contain null value.
- Defines the employee_number domain as a primary key
- Defines the employee_number domain in employee table as a foreign key

After a referential constraint has been defined, it can be dropped or altered by an ALTER TABLE statement. To drop a foreign or parent key after a referential constraint has been defined, you must first drop the constraint and then alter the table.

Examples of SQL data definition for check constraints

Referential constraint checks

The Teradata RDBMS performs referential constraints checks whenever any of the following occur:

- **A referential constraint is added to a populated table** The table will be check by the new referential constraint, if the referential integrity is violated, error message appears and the request is aborted.
- **A row is inserted, deleted or updated**

RDBMS statement	Constraint check performed
INSERT	For parent table: None; for child table: Must have matching parent key value if the foreign key is not null
DELETE	For parent table: Abort the request if the deleted parent key is referenced by any foreign key; for child table: None
UPDATE	For parent table: Abort the request if the parent key is referenced by any foreign key; for child table: New value must match the parent key when the foreign key is updated.

- **A parent or foreign key is modified** Check if there are any violations of the referential constraints, if there are, error message appears and the request is aborted.

Experiment no. 4

Objective : Study of creating table in a database in oracle.

Hardware and system software requirement: HDD 40 GB, 128 RAM and Window xp.

Software requirement: SQL, Window xp.

Theory: Creating Tables

To create a new table in your schema, you must have the CREATE TABLE system privilege. To create a table in another user's schema, you must have the CREATE ANY TABLE system privilege. Additionally, the owner of the table must have a quota for the tablespace that contains the table, or the UNLIMITED TABLESPACE system privilege.

Create tables using the SQL statement CREATE TABLE.

Creating a Table When you issue the following statement, you create a table named Employee in the your default schema and default tablespace. The below mentioned code can either be executed through SQL*PLUS or iSQL*PLUS.

```
CREATE TABLE employee (  
.....empno NUMBER(5) PRIMARY KEY,  
.....ename VARCHAR2(15) NOT NULL,  
.....job VARCHAR2(10),  
.....mgr NUMBER(5),  
.....hiredate DATE DEFAULT (sysdate),  
.....sal NUMBER(7,2),  
.....comm NUMBER(7,2),  
.....deptno NUMBER(3) NOT NULL  
); .....
```

```
Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE TABLE  employee (
2      empno      NUMBER(5) PRIMARY KEY,
3      ename      VARCHAR2(15) NOT NULL,
4      job        VARCHAR2(10),
5      mgr        NUMBER(5),
6      hiredate   DATE DEFAULT (sysdate),
7      sal        NUMBER(7,2),
8      comm       NUMBER(7,2),
9      deptno     NUMBER(3) NOT NULL
10     );

Table created.

SQL>
SQL>
```

Figure 1. Table creation through SQL*PLUS

CREATE TABLE Purpose

Use the CREATE TABLE statement to create one of the following types of tables:

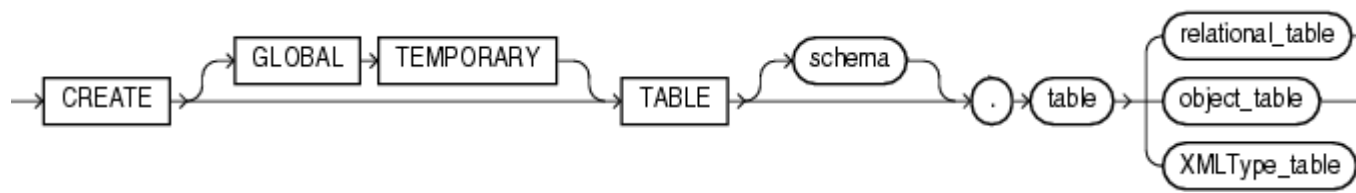
- A relational table, which is the basic structure to hold user data.
- An object table, which is a table that uses an object type for a column definition. An object table is explicitly defined to hold object instances of a particular type.

You can also create an object type and then use it in a column when creating a relational table.

Tables are created with no data unless a subquery is specified. You can add rows to a table with the INSERT statement. After creating a table, you can define additional columns, partitions, and integrity constraints with the ADD clause of the ALTER TABLE statement. You can change the definition of an existing column or partition with the MODIFY clause of the ALTER TABLE statement.

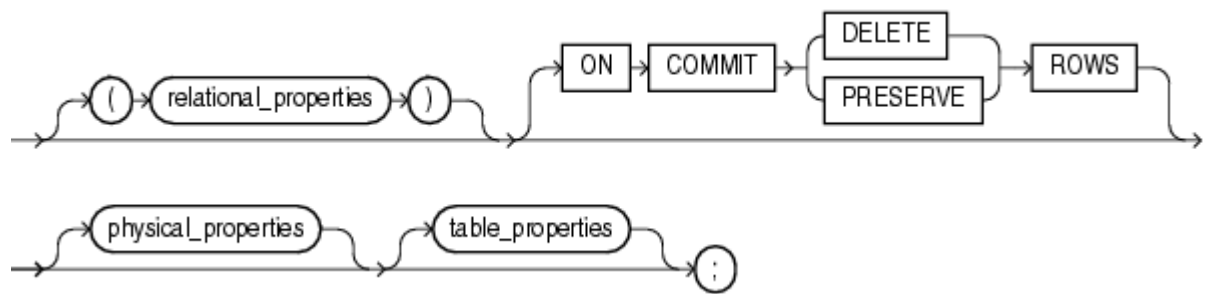
Syntax

create_table ::=



(relational table ::=, object table ::=, XMLType table ::=)

relational_table ::=



EXPERIMENT NO 5

Objective : To study of Inserting records into table.

Hardware and system software requirement: HDD 40 GB, 128 RAM and Window xp.

Software requirement: SQL, Window xp.

Theory: Insert Into Statement : In SQL, there are essentially basically two ways to INSERT data into a table: One is to insert it one row at a time, the other is to insert multiple rows at a time.

INSERT INTO VALUES

The syntax for inserting data into a table one row at a time is as follows :

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

Assuming that we have a table that has the following structure,

Table Store_Information

Column Name	Data Type
store_name	char(50)
Sales	float
Date	datetime

and now we wish to insert one additional row into the table representing the sales data for Los Angeles on January 10, 1999. On that day, this store had \$900 in sales. We will hence use the following SQL script:

```
INSERT INTO Store_Information (store_name, Sales, Date)
VALUES ('Los Angeles', 900, 'Jan-10-1999')
```

INSERT INTO SELECT

The second type of INSERT INTO allows us to insert multiple rows into a table. Unlike the previous example, where we insert a single row by specifying its values for all columns, we now use a SELECT statement to specify the data that we want to insert into the table. If you are thinking whether this means that you are using information from another table, you are correct. The syntax is as follows:

```
INSERT INTO "table1" ("column1", "column2", ...)
SELECT "column3", "column4", ...
FROM "table2"
```

Note that this is the simplest form. The entire statement can easily contain WHERE, GROUP BY, and HAVING clauses, as well as table joins and aliases.

So for example, if we wish to have a table, Store_Information, that collects the sales information for year 1998, and you already know that the source data resides in the Sales_Information table, we'll type in:

```
INSERT INTO Store_Information (store_name, Sales, Date)
SELECT store_name, Sales, Date
FROM Sales_Information
WHERE Year(Date) = 1998
```

Here I have used the SQL Server syntax to extract the year information out of a date. Other relational databases will have different syntax. For example, in Oracle, you will use to_char(date,'yyyy')=1998.

Example #1 - Simple example

```
INSERT INTO suppliers
(supplier_id, supplier_name)
VALUES
(24553, 'IBM');
```

This would result in one record being inserted into the suppliers table. This new record would have a supplier_id of 24553 and a supplier_name of IBM.

Example #2 - More complex example

You can also perform more complicated inserts using sub-selects.

For example:

```
INSERT INTO suppliers
(supplier_id, supplier_name)
SELECT account_no, name
FROM customers
WHERE city = 'Newark';
```

By placing a "select" in the insert statement, you can perform multiples inserts quickly.

With this type of insert, you may wish to check for the number of rows being inserted. You can determine the number of rows that will be inserted by running the following SQL statement before performing the insert.

```
SELECT count(*)
FROM customers
WHERE city = 'Newark';
```

Q: I am setting up a database with clients. I know that you use the "insert" statement to insert information in the database, but how do I make sure that I do not enter the same client information again?

A: You can make sure that you do not insert duplicate information by using the EXISTS condition.

For example, if you had a table named *clients* with a primary key of *client_id*, you could use the following statement:

```
INSERT INTO clients
(client_id, client_name, client_type)
SELECT supplier_id, supplier_name, 'advertising'
FROM suppliers
WHERE not exists (select * from clients
where clients.client_id = suppliers.supplier_id);
```

This statement inserts multiple records with a subselect.

If you wanted to insert a single record, you could use the following statement:

```
INSERT INTO clients
(client_id, client_name, client_type)
SELECT 10345, 'IBM', 'advertising'
FROM dual
WHERE not exists (select * from clients
where clients.client_id = 10345);
```

The use of the dual table allows you to enter your values in a select statement, even though the values are not currently stored in a table.

Q1: How can I insert multiple rows of explicit data in one SQL command in Oracle?

A1: The following is an example of how you might insert 3 rows into the *suppliers* table in Oracle.

```
INSERT ALL
  INTO suppliers (supplier_id, supplier_name) VALUES (1000, 'IBM')
  INTO suppliers (supplier_id, supplier_name) VALUES (2000, 'Microsoft')
  INTO suppliers (supplier_id, supplier_name) VALUES (3000, 'Google')
SELECT * FROM dual;
```

Inserting rows into a table using a select-statement

You can use a select-statement within an INSERT statement to insert zero, one, or more rows into a table from the result table of the select-statement.

One use for this kind of INSERT statement is to move data into a table you created for summary data. For example, suppose you want a table that shows

each employee's time commitments to projects. Create a table called EMPTIME with the columns EMPNUMBER, PROJNUMBER, STARTDATE, and ENDDATE and then use the following INSERT statement to fill the table:

```
INSERT INTO CORPDATA.EMPTIME
  (EMPNUMBER, PROJNUMBER, STARTDATE, ENDDATE)
SELECT EMPNO, PROJNO, EMSTDATE, EMENDATE
FROM CORPDATA.EMPPROJECT
```

The select-statement embedded in the INSERT statement is no different from the select-statement you use to retrieve data. With the exception of FOR READ ONLY, FOR UPDATE, or the OPTIMIZE clause, you can use all the keywords, functions, and techniques used to retrieve data. SQL inserts all the rows that meet the search conditions into the table you specify. Inserting rows from one table into another table does not affect any existing rows in either the source table or the target table.

You should consider the following when inserting multiple rows into a table:

Notes:


1. The number of columns implicitly or explicitly listed in the INSERT statement must equal the number of columns listed in the select-statement.
2. The data in the columns you are selecting must be compatible with the columns you are inserting into when using the INSERT with select-statement.
3. In the event the select-statement embedded in the INSERT returns no rows, an SQLCODE of 100 is returned to alert you that no rows were inserted. If you successfully insert rows, the SQLERRD(3) field of the SQLCA has an integer representing the number of rows SQL actually inserted. This value is also available from the ROW_COUNT diagnostics item in the GET DIAGNOSTICS statement.
4. If SQL finds an error while running the INSERT statement, SQL stops the operation. If you specify COMMIT (*CHG), COMMIT(*CS), COMMIT (*ALL), or COMMIT(*RR), nothing is inserted into the table and a negative SQLCODE is returned. If you specify COMMIT(*NONE), any rows inserted before the error remain in the table.

EXPERIMENT NO: 6

Objective: To Study of updating data records in table.

Hardware and system software requirement: HDD 40 GB, 128 RAM and Window xp.

Software requirement: SQL, Window xp.

Theory: Update Statement:  Once there's data in the table, we might find that there is a need to modify the data. To do so, we can use the UPDATE command. The syntax for this is

```
UPDATE "table_name"  
SET "column_1" = [new value]  
WHERE {condition}
```

For example, say we currently have a table as below:

Table *Store_Information*

store_name	Sales	Date
Los Angeles	\$1500	Jan-05-1999
San Diego	\$250	Jan-07-1999
Los Angeles	\$300	Jan-08-1999
Boston	\$700	Jan-08-1999

and we notice that the sales for Los Angeles on 01/08/1999 is actually \$500 instead of \$300, and that particular entry needs to be updated. To do so, we use the following SQL query:

```
UPDATE Store_Information  
SET Sales = 500  
WHERE store_name = "Los Angeles"  
AND Date = "Jan-08-1999"
```

The resulting table would look like

Table *Store_Information*

store_name	Sales	Date
Los Angeles	\$1500	Jan-05-1999
San Diego	\$250	Jan-07-1999
Los Angeles	\$500	Jan-08-1999
Boston	\$700	Jan-08-1999

In this case, there is only one row that satisfies the condition in the WHERE clause. If there are multiple rows that satisfy the condition, all of them will be modified. If no WHERE clause is specified, all rows will be modified.

It is also possible to UPDATE multiple columns at the same time. The syntax in this case would look like the following:

```
UPDATE "table_name"  
SET column_1 = [value1], column_2 = [value2]  
WHERE {condition}
```

SQL UPDATE Syntax

```
UPDATE table_name  
SET column1=value, column2=value2,...  
WHERE some_column=some_value
```

Note: Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

SQL UPDATE Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob		

Now we want to update the person "Tjessem, Jakob" in the "Persons" table.

We use the following SQL statement:

```
UPDATE Persons  
SET Address='Nissestien 67', City='Sandnes'  
WHERE LastName='Tjessem' AND FirstName='Jakob'
```

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger

5	Tjessem	Jakob	Nissestien 67	Sandnes
---	---------	-------	---------------	---------

SQL UPDATE Warning

Be careful when updating records. If we had omitted the WHERE clause in the example above, like this:

```
UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
```

The "Persons" table would have looked like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Nissestien 67	Sandnes
2	Svendson	Tove	Nissestien 67	Sandnes
3	Pettersen	Kari	Nissestien 67	Sandnes
4	Nilsen	Johan	Nissestien 67	Sandnes
5	Tjessem	Jakob	Nissestien 67	Sandnes

Modifying Rows with UPDATE

Once data has been inserted into rows within the database, those rows can have one or more of their column values modified through use of the SQL UPDATE command. Column values may be updated either with constants, identifiers to other data sets, or expressions. They may apply to an entire column, or a subset of a column's values through specified conditions. The UPDATE command uses the following syntax:

```
UPDATE [ ONLY ] table SET
    column = expression [, ...]
    [ FROM source ]
    [ WHERE condition ]
UPDATE [ ONLY ] table
```

The ONLY keyword may be used to indicate that only the table *table* should be updated, and none of its sub-tables. This is only relevant if *table* is inherited by any other tables.

```
SET column = expression [, ...]
```

The required SET clause is followed by an update expression for each column name that needs to have its values modified, separated by commas. This expression is always of the form *column = expression*, where *column* is the name of the column to be updated (which may not be aliased, or dot-notated), and where *expression* describes the new value to be inserted into the column.

FROM *source*

The FROM clause is a non-standard PostgreSQL extension that allows table columns from other data sets to update a column's value.

WHERE *condition*

The WHERE clause describes the *condition* upon which a row in *table* will be updated. If unspecified, *all values* in *column* will be modified. This may be used to qualify sources in the FROM clause, as you would in a SELECT statement.

EXPERIMENT NO: 7

Objective : To Study of deleting records in the table.

Hardware and system software requirement: HDD 40 GB, 128 RAM and Window xp.

Software requirement: SQL, Window xp.

Theory: SQL DELETE Statement

The DELETE statement is used to delete records in a table.

The DELETE Statement

The DELETE statement is used to delete rows in a table.

SQL DELETE Syntax

DELETE FROM table_name

WHERE some_column=some_value

Note: Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

SQL DELETE Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	Nissestien 67	Sandnes

Now we want to delete the person "Tjessem, Jakob" in the "Persons" table.

We use the following SQL statement:

DELETE FROM Persons

WHERE LastName='Tjessem' AND FirstName='Jakob'

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger

Delete All Rows

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name
```

or

```
DELETE * FROM table_name
```

Note: Be very careful when deleting records. You cannot undo this statement!

Deleting Data from an SQL Table

Oftentimes, it becomes necessary to remove obsolete information from a relational database. Fortunately, Structured Query Language provides a flexible DELETE command that can be used to remove some or all of the information stored within a table.

Let's take a brief look at the command's syntax:

```
DELETE FROM  
{table name | view name}  
[WHERE search_conditions]
```

Notice that the command itself is quite simple. There are only two variables -- the table or view to delete from and the search conditions.

Let's first discuss the target of the deletion. According to the ANSI SQL standard, it is possible to delete from either a table or a view. However, I'd strongly encourage you to avoid using the DELETE command (or any data manipulation command, for that matter) on a view. Some versions of SQL simply don't support this syntax and the results of modifying a view can be somewhat unpredictable.

The *search_conditions* field offers no surprises to students of SQL -- it uses the same format as the *search_conditions* utilized with the SELECT statement. You can include any comparison operators to limit the data that is removed from the table. Notice that the *search_conditions* field is actually an optional argument (hence the square brackets surrounding it). Omission of this argument will result in the deletion of the entire table.

Now let's turn to some examples. Before we get started, we need to create a table and load it with some sample data. We'll create a students table for our small town high school. Execute the following SQL code against your DBMS of choice:

```
CREATE TABLE students
(
first_name varchar(50),
last_name varchar(50),
id integer PRIMARY KEY
)
```

```
INSERT INTO students VALUES ('John', 'Doe', 284)
INSERT INTO students VALUES ('Mike', 'Ryan', 302)
INSERT INTO students VALUES ('Jane', 'Smith', 245)
INSERT INTO students VALUES ('MaryAnn', 'Pringle', 142)
INSERT INTO students VALUES ('Charlotte', 'Bronte', 199)
INSERT INTO students VALUES ('Bill', 'Arlington', 410)
```

Bill Arlington had stellar academic achievement and was allowed to graduate early. Therefore, we must remove him from the database. As SQL experts, we know that when we want to select a single record, it's prudent to use the primary key in the search condition to prevent accidental removal of similar records. Here's our syntax:

```
DELETE FROM students
WHERE id = 410
```

Here are the contents of the modified table:

first_name	last_name	id
Mike	Ryan	302
MaryAnn	Pringle	142
Charlotte	Bronte	199
Jane	Smith	245
John	Doe	284

Now let's try something a bit more complicated -- deleting all of the students with ID numbers between 240 and 290. Here's the SQL:

```
DELETE FROM students
WHERE id BETWEEN 240 AND 290
```

and the newly modified table:

first_name	last_name	id
MaryAnn	Pringle	142

Charlotte	Bronte	199
Mike	Ryan	302

And finally, we're sorry to report that our school closed it's doors due to dwindling enrollment. Out of respect for student privacy, we need to remove all of the data from the table. This SQL command will do the trick:

```
DELETE FROM students
```

EXPERIMENT NO: 8

Objective :To study of modifying table structure.

Hardware and system software requirement: HDD 40 GB, 128 RAM and Window xp.

Software requirement: SQL, Window xp.

Theory: To modify the structure of a table with the Table Designer
In the Project Manager, select the table name and then choose Modify. The structure of the table is displayed in the Table Designer.

-or-

In the Database Designer, select the table in the schema and choose Modify from the Database menu.

-or-

Use the MODIFY STRUCTURE command.

To modify the structure of a table programmatically

Use the ALTER TABLE command.

The ALTER TABLE command offers extensive clauses that enable you to add or drop table fields, create or drop primary or unique keys or foreign key tags, and rename existing fields. Some clauses apply only to tables associated with a database. A few specific examples are included in this section

SQL ALTER TABLE Statement

The ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

SQL ALTER TABLE Syntax

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name  
ADD column_name datatype
```

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name  
DROP COLUMN column_name
```

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name  
ALTER COLUMN column_name datatype
```

SQL ALTER TABLE Example

Look at the "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to add a column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

```
ALTER TABLE Persons  
ADD DateOfBirth date
```

Notice that the new column, "DateOfBirth", is of type date and is going to hold a date. The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MS Access, MySQL, and SQL Server, go to our complete

The "Persons" table will now like this:

P_Id	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

EXPERIMENT NO: 9

Objective: To study of dropping table from database.

Hardware and system software requirement: HDD 40 GB, 128 RAM and Window xp.

Software requirement: SQL, Window xp.

Theory: DROP TABLE removes tables from the database. You must be the owner of the table, or a superuser, in order to drop it.

Note: To empty a table (as opposed to completely deleting it), use either the TRUNCATE or DELETE command.

Deleting a table also destroys any indices that were placed on that table.

Example

The following command permanently removes the employees table from the booktown database:

```
booktown=# DROP TABLE employees;  
DROP
```

DROP TABLE

Name

DROP TABLE -- Removes a table from a database.

Synopsis

DROP TABLE *name* [, ...]

Parameters

name

The name of an existing table you intend to drop. You may drop multiple tables by specifying their names in a comma-delimited list.

Results

DROP

The message returned when a table is dropped successfully.

ERROR: table "*name*" does not exist!

The error returned if the specified table or view *name* does not exist in the database.

Deleting Database Tables with the DROP Command

The DROP command allows us to remove entire database objects from our database. For example, if we want to permanently remove the `personal_info` table that we created, we'd use the following command:

```
DROP TABLE personal_info
```

Similarly, the command below would be used to remove the entire `employees` database:

```
DROP DATABASE employees
```

Use this command with care! Remember that the DROP command removes entire data structures from your database. If you want to remove individual records, use the DELETE command of the Data Manipulation Language.

Drop Table in Database

[<-Prior](#) [Index](#) [Next->](#)

The Drop Statement in SQL Command is used to delete or remove indexes, tables and database. The Syntax used for Drop table statement in database is given as:

```
Drop Table table_ Name;
```

The above syntax is used to delete a table.

table Name : specify the name of the table, which you want to delete from database.

Understand with Example

The Tutorial illustrates an example from Drop Table in Database. In this tutorial we show you a table created in database using show tables.

```
mysql> show tables;
+-----+
| Tables_in_countryinfo |
+-----+
| country                |
+-----+
1 row in set (0.03 sec)
```


Now we create another table 'student information'. The insert statement is used to add the record to the table 'student information'.

```
mysql> insert into studentinformation values('Saurabh',01);
Query OK, 1 row affected (0.03 sec)

mysql> insert into studentinformation values('Romoe',02);
Query OK, 1 row affected (0.01 sec)
```

The select statement is used to retrieve the record from the table 'student information'.

```
mysql> select * from studentinformation;
+-----+-----+
| Name   | RollNo |
+-----+-----+
| Saurabh |      1 |
| Romoe  |      2 |
+-----+-----+
2 rows in set (0.00 sec)
```

 The show tables show you the list of table available in database.

```
mysql> show tables;
+-----+
| Tables_in_countryinfo |
+-----+
| country                |
| studentinformation     |
+-----+
2 rows in set (0.00 sec)
```

The drop statement is used to delete a table from database.

```
mysql> drop table studentinformation;
Query OK, 0 rows affected (0.01 sec)
```

Once the table is deleted using drop statement, the table is no longer to be shown in the database. When we write show tables, this will show you the table available in database.

```
mysql> show tables;
+-----+
| Tables_in_countryinfo |
+-----+
| country                |
+-----+
1 row in set (0.00 sec)
```

EXPERIMENT NO: 10

Objective: To study Creation of users & granting the privileges

(a) About object privileges

(b) Design & implementation of database for an organization

Hardware and system software requirement: HDD 40 GB, 128 RAM and Window xp.

Software requirement: SQL, Window xp.

Theory: A privilege is a right to execute an SQL statement or to access another user's object. In Oracle, there are two types of privileges: system privileges and object privileges. A privileges can be assigned to a user_or a role

The set of privileges is fixed, that is, there is no SQL statement like create privilege xyz...

System privileges

There are quite a few system privileges: in Oracle 9.2, we count 157 of them, and 10g has even 173. Those can be displayed with
select name from system_privilege_map

Executing this statement, we find privileges like *create session*, *drop user*, *alter database*, see system privileges.

System privileges can be audited.

Arguably, the most important system privileges are:

- create session (A user cannot login without this privilege. If he tries, he gets an ORA-01045).
- create table
- create view
- create procedure
- sysdba
- sysoper

Object privileges

privileges can be assigned to the following types of database objects:

- Tables
select, insert, update, delete, alter, debug, flashback, on commit refresh, query rewrite, references, all
- Views
select, insert, update, delete, under, references, flashback, debug
- Sequence
alter, select
- Packages, Procedures, Functions (Java classes, sources...)
execute, debug
- Materialized Views
delete, flashback, insert, select, update

- Directories
read, write
- Libraries
execute
- User defined types
execute, debug, under
- Operators
execute
- Indextypes
execute

For a user to be able to access an object in another user's schema, he needs the according object privilege.

Object privileges can be displayed using `all_tab_privs_made` or `user_tab_privs_made`.

Public

If a privilege is granted to the special role public, this privilege can be executed by all other users. However, sysdba cannot be granted to public.

Users to be finished

Roles

Predefined Roles

Along with the installation, more exactly with the creation of an oracle database, Oracle creates predefined roles. These are:

- connect, resource, dba
These might not be created anymore in future versions of Oracle. Oracle 9.2 grants *create session, alter session, create synonym, create view, create database link, create table, create cluster* and *create sequence* to connect.
It also grants *create table, create cluster, create sequence, create trigger, create procedure, create_type, create indextype* and *create operator* to resource.
The role dba gets basically everything and that *with admin option*.
- delete_catalog_role, execute_catalog_role, select_catalog_role
Accessing data dictionary views (v\$ views and static dictionary views)
- exp_full_database, imp_full_database
This role is needed to export objects found in another user's schema.
- aq_user_role, aq_administrator_role, global_aq_user_role(?)
- logstdby_administrator
- snmpagent
- recovery_catalog_owner
- hs_admin_role
- oem_monitor, oem_advisor
- scheduler_admin
- gather_system_statistics
- plustrace

- xdbadmin
- xdbwebservice
- ctxapp

Assigning privileges to users and roles

A privilege can be assigned to a user with the grant sql statement. On the other hand, revoke allows to take away such privileges from users and roles. Oracle stores the granted privileges in its data dictionary.

Displaying the relationship between users, roles and privileges

Use this script to recursively list users, granted roles and privileges.



Group Privileges...

Searched for the last few days, and haven't found any info on this.

Personally, I don't consider it a feature request, but rather a bug that needs to be fixed:

Groups should have SEPERATE User Templates, rather than a single template for all.

As well, as far as I can tell, very little of the current implementation of Zentyal actually USES the Groups. I might suggest that when you enable a service (Radius, Proxy, VPN, Web Server, etc...), the option might be offered to create a group for that service, and choice given to add users to it.

And, I see admin rights being grantable per user, but no way to create an admins group, that would have those rights granted per the admins group template...

Examples:

Group	Privileges or Automatic Changes made
pam	Some should have, others should not
admins	If also in pam, system user would be added to
sudo-enabled group,	otherwise, should be able to log into Zentyal and
admin the system	
vpn (group PER vpn server)	Cert created, bundle placed in user's home dir
radius	Permitted to auth, as currently done via Radius
Module	
roaming	Some users may need it, others may not
web	Enable per user public_html, but ONLY for
those in this group	
proxy	Makes proxy transparent / automatic for these
users only	
jabber	Should be self-explanatory, if also in admins,
grant admin	
email	Should be self-explanatory, if also in admins,

```
grant admin
groupware                Should be self-explanatory, if also in admins,
grant admin
printers                  Should be self-explanatory, if also in admins,
grant admin
Granting Object Privileges
Grant A Single Privilege GRANT <privilege_name> ON <object_name> TO
<schema_name> conn uwclass/uwclass
```

```
CREATE TABLE test (
testcol VARCHAR2(20));
```

```
GRANT SELECT ON test TO abc;
```

```
set linesize 100
col grantee format a30
col table_name format a30
col privilege format a20
```

```
SELECT grantee, table_name, privilege
FROM user_tab_privs_made;
```

```
conn abc/abc
```

```
SELECT grantor, table_name, privilege
FROM user_tab_privs_recd;
Grant Multiple Privileges GRANT <privilege_name_list> ON <object_name>
TO <schema_name> conn uwclass/uwclass
```

```
GRANT INSERT, DELETE ON test TO abc;
```

```
SELECT grantee, table_name, privilege
FROM user_tab_privs_made;
```

```
conn abc/abc
```

```
SELECT grantor, table_name, privilege
FROM user_tab_privs_recd;
Grant All Privileges GRANT ALL ON <object_name> TO <schema_name>
conn abc/abc
```

```
GRANT ALL ON test TO uwclass;
```

```
SELECT grantee, table_name, privilege
FROM user_tab_privs_made;
```

```
conn uwclass/uwclass
```

```
SELECT grantor, table_name, privilege
FROM user_tab_privs_recd;
```

Grant Execute GRANT EXECUTE ON <object_name> TO <schema_name>
conn uwclass/uwclass

GRANT EXECUTE ON getosuser TO abc;

SELECT grantee, table_name, privilege
FROM user_tab_privs_made;

conn abc/abc

SELECT grantor, table_name, privilege
FROM user_tab_privs_recd; Revoking Object Privileges
Revoke A Single Privilege REVOKE <privilege_name> ON <object_name>
FROM <schema_name> conn uwclass/uwclass

REVOKE SELECT ON test FROM abc;

SELECT grantee, table_name, privilege
FROM user_tab_privs_made;

conn abc/abc

SELECT grantor, table_name, privilege
FROM user_tab_privs_recd;
Revoke Multiple Privileges REVOKE <privilege_name_list> ON
<object_name> FROM <schema_name> conn uwclass/uwclass

REVOKE INSERT, DELETE ON test FROM abc;

SELECT grantee, table_name, privilege
FROM user_tab_privs_made;

conn abc/abc

SELECT grantor, table_name, privilege
FROM user_tab_privs_recd;
Revoke All Privileges REVOKE ALL ON <object_name> FROM
<schema_name> conn uwclass/uwclass

REVOKE ALL ON test FROM abc;

SELECT grantee, table_name, privilege
FROM user_tab_privs_made;

conn abc/abc

SELECT grantor, table_name, privilege
FROM user_tab_privs_recd;
Revoke Execute REVOKE EXECUTE ON <object_name> FROM
<schema_name> conn uwclass/uwclass

```
REVOKE EXECUTE ON getosuser FROM abc;
```

```
SELECT grantee, table_name, privilege  
FROM user_tab_privs_made;
```

```
conn abc/abc
```

```
SELECT grantor, table_name, privilege  
FROM user_tab_privs_recd; Granting Column Privileges Grant Column  
Privileges GRANT <privilege_name> (<column_name>) ON <table_name>  
TO <schema_name>; GRANT UPDATE (first_name, last_name) ON person  
TO uwclass; Revoking Column Privileges Revoke Column Privilege  
REVOKE <privilege_name> (<column_name>) ON <table_name> FROM  
<schema_name>; REVOKE UPDATE (first_name, last_name) ON person  
FROM uwclass; Object Privilege Related Query  
Show privileges by object set linesize 121  
col select_priv format a10  
col insert_priv format a10  
col update_priv format a10  
col delete_priv format a10
```

```
SELECT table_name, grantee,  
MAX(DECODE(privilege, 'SELECT', 'SELECT')) AS select_priv,  
MAX(DECODE(privilege, 'DELETE', 'DELETE')) AS delete_priv,  
MAX(DECODE(privilege, 'UPDATE', 'UPDATE')) AS update_priv,  
MAX(DECODE(privilege, 'INSERT', 'INSERT')) AS insert_priv  
FROM dba_tab_privs  
WHERE grantee IN (  
  SELECT role  
  FROM dba_roles)  
GROUP BY table_name, grantee;
```

Granting database object privileges

An object privilege allows a user to select or modify data from a specific database object, such as a table. The five object privileges are as follows:

- SELECT
- INSERT
- UPDATE
- DELETE
- ALL PRIVILEGES

As a member of the DBA system role (or as a RESOURCE member and creator of the object), you can use the GRANT statement to grant object privileges to database users. Object privileges are granted to one or more specified users or to all users, specified as PUBLIC. Users must be granted the CONNECT system role and assigned a password before being granted

object privileges. Object privileges can be removed at any time with the REVOKE statement.

The following table defines the actions permitted on database objects for a user who is a member of the DBA system role, created the object, and has been granted object privileges, and for all others.

Object privileges permitted	Users			
	DBA	Creator ¹	Grantee	PUBLIC
¹ Members of the RESOURCE system role have all object privileges on tables they create. ² To insert rows, users must have INSERT and SELECT privilege on the object.				
SELECT from object	Yes	Yes	Yes	No
INSERT into object	Yes	Yes	Yes ²	No
UPDATE object	Yes	Yes	Yes	No
DELETE from object	Yes	Yes	Yes	No
Use PUBLIC macro	Yes	Yes	Not applicable	Yes

For a complete discussion of the GRANT and REVOKE statements, see the *SQL Reference Guide*.

Example

This example illustrates how the user curly (with RESOURCE) can grant the SELECT privilege on a table named t1 that he created to the user moe (who has already been granted CONNECT).

```
grant select on t1 to moe ;
```

GRANT privilege

The GRANT statement can assign object privileges on a specific table to one or more users or user-created roles.

Authorization

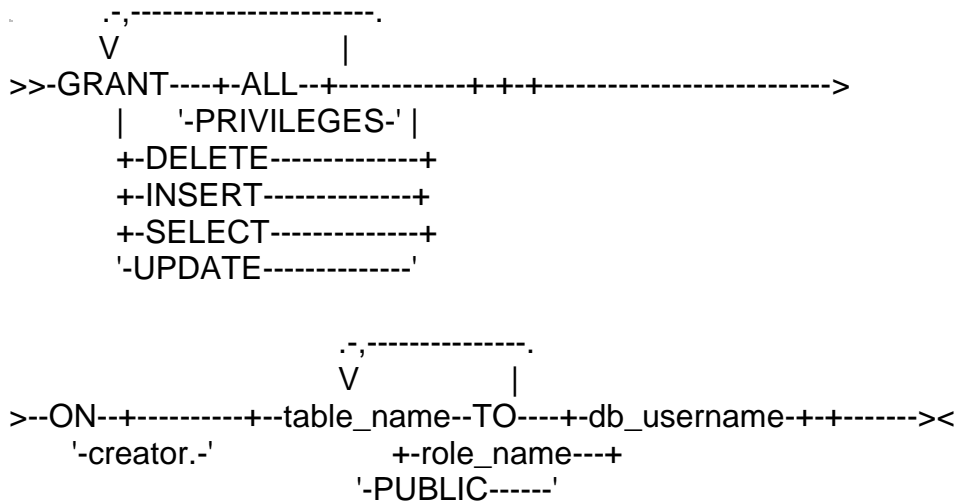
To grant an object privilege on a table, a user must meet at least one of the following requirements:

- Be a member of the DBA system role
- Be a member of the RESOURCE system role and be the creator of the table
- Be the creator of the table and have the GRANT_OWN task authorization, either explicitly or through membership in a user-created role

- Have the GRANT_TABLE task authorization, either explicitly or through membership in a user-created role

Syntax

The following syntax diagram shows how to construct a GRANT privilege statement.



PRIVILEGES

Object privileges can be granted to a specified user, a role, or to all users (to PUBLIC). A user must be granted CONNECT privilege and assigned a password before being granted object privileges. A role must be created with the CREATE ROLE statement before it can be granted object privileges.

System roles cannot be granted object privileges.

A user or role can be granted one or more of the following object privileges on a named table.

Object privilege	Description
DELETE	Delete rows
INSERT	Insert rows
SELECT	Retrieve rows
UPDATE	Modify rows
ALL PRIVILEGES	All the above

Any user with INSERT privilege on a table must also have SELECT privilege on the table to insert rows.

A user who creates a table automatically has all object privileges on that table. These object privileges cannot be revoked from the table creator. A member of the DBA system role has all object privileges on any nonsystem table in the database.

table_name

Specifies the name of a table, view, or synonym. If the named table is a view, only SELECT privilege can be granted. You cannot grant privileges on a temporary table. If specified, the creator must be the username of the user who created the table.

TO db_username

Grants all specified object privileges to a database user. A database username must exist before object privileges can be granted to it.

Grants all specified object privileges to a user-created role. All members of the role have the object privilege. A role name must exist before object privileges can be granted to it. A role name cannot be a system role because system roles cannot be altered.

TO PUBLIC

Grants all specified object privileges to all database users.

Examples

The following statement grants the SELECT privilege on the Product table to all database users:

```
grant select on product to public
```

The following statement grants SELECT, INSERT, DELETE, and UPDATE privileges on the Sales table to alison:

```
grant all privileges on sales to alison
```

The following statement grants the SELECT, INSERT, and DELETE privileges on the Market table to the market_research role. All members of the market_research role are able to perform these operations on the Market table.

```
grant select, insert, delete on market to market_research
```