Creating Tables

CIS 399 Week 2
CREATE TABLE simplified syntax

CREATE TABLE [schema.]table_name
(
    column_name column_type
    [PRIMARY KEY]
    [NOT NULL]
    [CONSTRAINT constraint_definition]
    [DEFAULT expression]
    [, ...]
    [constraint_definition] [, ...]
);

CREATE TABLE example

CREATE TABLE khuck.test

( name VARCHAR2(15) PRIMARY KEY
  CONSTRAINT ch_name CHECK (name != 'bad name'),
description VARCHAR2(256) NOT NULL,
entry_date DATE DEFAULT DEFAULT SYSDATE
);

# Table 3.1 Oracle character data types

<table>
<thead>
<tr>
<th>Type</th>
<th>Oracle Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed</td>
<td>CHAR(size)</td>
<td>Fixed-length character data of size characters padded with spaces. Maximum size is 2000 bytes.</td>
</tr>
<tr>
<td>Fixed National</td>
<td>NCHAR(size)</td>
<td>Same as CHAR except stores National characters of maximum length 2000 bytes.</td>
</tr>
<tr>
<td>Variable</td>
<td>VARCHAR2(size)</td>
<td>Variable-length character data of size characters. Maximum size is 4000 bytes.</td>
</tr>
<tr>
<td>Variable National</td>
<td>NVARCHAR2(size)</td>
<td>Variable-length character data of size National characters. Maximum size is 4000 bytes.</td>
</tr>
<tr>
<td>Memo</td>
<td>LONG</td>
<td>Character data of variable length up to 2 gigabytes. (Not recommended. Use CLOB data type instead.)</td>
</tr>
</tbody>
</table>
Table 3.2 Oracle numeric data types

<table>
<thead>
<tr>
<th>Type</th>
<th>Oracle Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT, INTEGER, SMALLINT</td>
<td>NUMBER(38)</td>
<td>An integer with up to 38 digits of precision.</td>
</tr>
<tr>
<td>Fixed precision</td>
<td>NUMBER(p,s)</td>
<td>A variable length number. Precision is the maximum number of digits, scale is the maximum number of digits to the right of the decimal point.</td>
</tr>
<tr>
<td>FLOAT, DOUBLE PRECISION</td>
<td>NUMBER</td>
<td>A floating-point number with up to 38 digits of precision.</td>
</tr>
</tbody>
</table>
Table 3.3 Oracle date and time data types

<table>
<thead>
<tr>
<th>Type</th>
<th>Oracle Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date and time to whole seconds</td>
<td>DATE</td>
<td>Date and time with the century, all four digits of the year, month, day, hour, minute and second</td>
</tr>
<tr>
<td>Time interval in years</td>
<td>INTERVAL YEAR(p) TO MONTH</td>
<td>Time in years and months, where precision is the number of digits in the YEAR date/time field.</td>
</tr>
<tr>
<td>Time interval in days</td>
<td>INTERVAL DAY(p) TO SECOND(f)</td>
<td>Time in days, hours, minutes and seconds. Precision, p, is the max number of digits in the day, and f is the number of fractional digits in the seconds field.</td>
</tr>
<tr>
<td>Date and time to fractions of a second</td>
<td>TIMESTAMP(p)</td>
<td>Stores date and time to fractions of a second. Decimal places specified by precision in parentheses.</td>
</tr>
</tbody>
</table>
Table 3.4 Oracle image data types

<table>
<thead>
<tr>
<th>Oracle Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB</td>
<td>Binary LOB stores binary, unstructured data up to 128 terabytes.</td>
</tr>
<tr>
<td>CLOB</td>
<td>Character LOB stores character data for very large objects—up to 128 terabytes.</td>
</tr>
<tr>
<td>NCLOB</td>
<td>Variable length Unicode national character data up to 128 terabytes.</td>
</tr>
</tbody>
</table>
Table and Column Comments

COMMENT ON TABLE test IS 'this is a test table';

SELECT table_name, comments FROM user_tab_comments WHERE table_name = 'test';

COMMENT ON COLUMN test.name IS 'this is the entity name';

SELECT * FROM user_col_comments WHERE table_name = 'test';
## Constraints & Naming Conventions

<table>
<thead>
<tr>
<th>Constraint Type</th>
<th>Constraint Type Prefix</th>
<th>Example Constraint Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK</td>
<td>ck</td>
<td>ck_customers_zipcode</td>
</tr>
<tr>
<td>FOREIGN KEY</td>
<td>fk</td>
<td>fk_customers_properties</td>
</tr>
<tr>
<td>NOT NULL</td>
<td>nn</td>
<td>nn_agents_lastname</td>
</tr>
<tr>
<td>PRIMARY KEY</td>
<td>pk</td>
<td>pk_agentid</td>
</tr>
<tr>
<td>UNIQUE</td>
<td>un</td>
<td>un_contactreason_description</td>
</tr>
</tbody>
</table>
CREATE TABLE employee (...
    id INTEGER PRIMARY KEY, ...
);

CREATE TABLE employee (...
    id INTEGER CONSTRAINT pk_employee PRIMARY KEY,
    ...
);

CREATE TABLE employee (...
    CONSTRAINT pk_employee PRIMARY KEY (id), ...
);

CREATE TABLE order_item (...
    CONSTRAINT pk_order_item PRIMARY KEY (order_id, item_id),...
);
Integrity - Foreign Key Column Constraints

\[
<\text{column-name}> <\text{datatype}> \ [\text{CONSTRAINT}
<\text{constraint-name}> \ \text{REFERENCES} \ <\text{tablename}>
[(<\text{column-name}>)][\text{ON DELETE} \ {\text{CASCADE}|\text{SET NULL}|\text{SET TO DEFAULT}|\text{RESTRICT}|\text{NO ACTION}}]
\]

CREATE TABLE employee (... ManagerID INTEGER
CONSTRAINT fk_employee_manager REFERENCES Manager(id),...);

CREATE TABLE employee (... ManagerID INTEGER
REFERENCES Manager(id),...);

CREATE TABLE employee (... ManagerID INTEGER
REFERENCES Manager,...); /* assumes PK */
CREATE TABLE employee ( 
  ManagerID INTEGER,  
  CONSTRAINT fk_employee_manager FOREIGN KEY ManagerFN, ManagerLN REFERENCES Manager(fn, ln), ...);
Domain - Not Null Column Constraints

CREATE TABLE Properties (  
    PropertyID INTEGER PRIMARY KEY,  
...  
    Address NVARCHAR(30) NOT NULL,  
    City NVARCHAR(30) NOT NULL,  
    State NVARCHAR(20),  
...);
CREATE TABLE Properties ( 
    PropertyID INTEGER PRIMARY KEY, 
    ... 
    Address NVARCHAR(30) UNIQUE, 
    City NVARCHAR(30), 
    State NVARCHAR(20), 
    ...);
CONSTRAINT <constraint-name> CHECK (<logical-expression>)

CREATE TABLE agents ( 
...
HomePhone NVARCHAR(20),
Title NVARCHAR(20) CONSTRAINT ck_agents_title CHECK ((Title = 'Broker') OR (Title = 'Salesperson')),
...
);
Modifying Constraints

ALTER TABLE <table-name> ADD CONSTRAINT ... 
ALTER TABLE <table-name> MODIFY CONSTRAINT ... 
ALTER TABLE <table-name> ENABLE CONSTRAINT <constraint-name> 
ALTER TABLE <table-name> DISABLE CONSTRAINT <constraint-name> 

ALTER TABLE Listings ADD CONSTRAINT fk_agents_listings FOREIGN KEY (ListingAgentID) REFERENCES Agents(AgentID) ON DELETE CASCADE;
ALTER TABLE `<table-name>` DROP CONSTRAINT `<constraint-name>`;

ALTER TABLE `<table-name>` RENAME CONSTRAINT `<constraint-name>` TO `<constraint-new-name>`;

ALTER TABLE Listings RENAME CONSTRAINT fk_agents_listings fk_listings_agents;
Modifying a column type or default

ALTER TABLE Listings MODIFY BeginListDate DEFAULT SYSDATE;

ALTER TABLE Listings MODIFY BeginListDate DEFAULT NULL;

ALTER TABLE Listings MODIFY AskingPrice NUMBER(12,2);

ALTER TABLE Customers ADD ContactDate DATE DEFAULT SYSDATE;

ALTER TABLE Customers DROP COLUMN HomePhone;

ALTER TABLE Students DROP COLUMN (age, hair_color, eye_color);
Hiding or Dropping a Column

ALTER TABLE <table-name> SET UNUSED COLUMN <column-name>;

ALTER TABLE <table-name> DROP COLUMN <column-name>;}
Displaying Table Information

```
SELECT table_name FROM user_tables;
SELECT table_name FROM all_tables;
SELECT table_name FROM dba_tables;
```
## Useful Tables (with user_, all_ or dba_)

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONS_COLUMNS</td>
<td>Table columns having constraints</td>
</tr>
<tr>
<td>CONSTRAINTS</td>
<td>Table constraints</td>
</tr>
<tr>
<td>INDEXES</td>
<td>Table indexes</td>
</tr>
<tr>
<td>OBJECTS</td>
<td>All database objects</td>
</tr>
<tr>
<td>SEQUENCES</td>
<td>Sequences generating unique keys</td>
</tr>
<tr>
<td>TAB_COLUMNS</td>
<td>Table columns. (USER_TAB_COLUMNS are the current user’s columns)</td>
</tr>
<tr>
<td>TABLES</td>
<td>Database tables</td>
</tr>
<tr>
<td>USERS</td>
<td>Names of all database users (ALL_USERS)</td>
</tr>
<tr>
<td>VIEWS</td>
<td>Database views</td>
</tr>
</tbody>
</table>
Dropping, Reinstating, Renaming Tables

DROP TABLE <table-name> [CASCADE CONSTRAINTS] [PURGE]

FLASHBACK TABLE <table-name> TO BEFORE DROP;

PURGE TABLE <tablename>

PURGE RECYCLEBIN

RENAME TABLE <table-name> TO <table-new-name>

Select * from user_dependencies;
Modifying Data and Auditing Table Operations

- INSERT
- SEQUENCE Objects
- UPDATE
- DELETE, TRUNCATE
- MERGE INTO Statement
- Transactions
- TRIGGER Objects
INSERT Statements

CREATE TABLE test (  
    name VARCHAR(15) NOT NULL,  
    description VARCHAR(15),  
    CONSTRAINT pk_name PRIMARY KEY (name)  
);  
/* insert a row into the database, specifying all columns */  
INSERT INTO test VALUES ('joe', 'a test user');  
/* insert another row into the database, specifying all columns */  
INSERT INTO test (description, name)  
VALUES ('another test user', 'joe');  
/* insert another row into the database, specifying only the required column */  
INSERT INTO test (name) VALUES ('fifth');
Dealing with integrity constraints

CREATE TABLE students (  
id CHAR(9) PRIMARY KEY,  
name VARCHAR2(256) NOT NULL );
CREATE TABLE classes (  
crn CHAR(5) PRIMARY KEY,  
name VARCHAR(256) );
CREATE TABLE grades (  
student_id CHAR(9) REFERENCES students (id),  
crn CHAR(5) REFERENCES classes(crn),  
grade CHAR(2) NOT NULL );

/* the following is allowed, but not recommended! */

INSERT INTO grades (student_id, crn, grade)  
VALUES (‘950123456’, NULL, ‘A–’);
Inserting Date and Time values

/* default formats for dates are DD–MON–YY (<49 assumes 2000, >49 assumes 1900) and DD–MON–YYYY */

INSERT INTO appointments (name, date)
VALUES ('Dr. Bob', '15–JUL–2006');

/* use a format model if you have something different */

INSERT INTO appointments (name, date)
VALUES ('Dr. Bob', TO_DATE('15/07/06 09:15:00 AM', 'DD/MM/YY HH:MM:SS AM');

/* format rules:
http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10759/sql_elements004.htm#i34924 */
Inserting Data from Other tables

CREATE TABLE applicants (  
id CHAR(9) PRIMARY KEY,  
first_name VARCHAR2(256) NOT NULL ,  
last_name VARCHAR2(256) NOT NULL );

CREATE TABLE students (  
id CHAR(9) PRIMARY KEY,  
first_name VARCHAR2(256) NOT NULL ,  
last_name VARCHAR2(256) NOT NULL );

/* accept the applicant! */

INSERT INTO students (id, name)  
SELECT id, name FROM applicants  
WHERE id = '950123456';
SEQUENCE Objects
/* a SEQUENCE is a database object that generates a series of unique integers. Very useful for generating unique primary keys. Syntax:

CREATE SEQUENCE <sequence name>
[ START WITH <value> ]
[ INCREMENT BY <value> ]
[ { MAXVALUE <value> | NOMAXVALUE } ]
[ { MINVALUE <value> | NOMINVALUE } ]
[ { CYCLE | NOCYCLE } ] [ { ORDER | NOORDER } ]
[ { CACHE <value> | NOCACHE } ]

Example: */

CREATE SEQUENCE student_id_seq START WITH 950000000 INCREMENT BY 1;
Getting SEQUENCE values

/* CURRVAL: get the current value in the sequence (DUAL is a special table that doesn’t really exist */

SELECT student_id_seq.CURRVAL FROM DUAL;

/* NEXTVAL: get the next value in the sequence */

INSERT INTO students (id, first_name, last_name) VALUES
  (student_id_seq.NEXTVAL, ‘Sally’, ‘Student’)

/* ALTER SEQUENCE looks just like CREATE SEQUENCE */

/* DROP SEQUENCE is straightforward: */

DROP SEQUENCE student_id_seq_seq;
**UPDATE statement**

/* Used to update one or more rows in a table. Syntax:

UPDATE <table name>
SET <column name> = <expression1>
[, <column name> = <expression1>, ...]  
[ WHERE <condition> ]
*/

UPDATE students SET first_name = 'Sallie'
WHERE id = '950123456';

UPDATE grades SET grade = 'A+' WHERE grade LIKE 'F%' 
OR grade LIKE 'D%' OR grade LIKE 'C%' OR grade LIKE 'B%';

UPDATE grades SET grade = 'A+'
WHERE grade NOT LIKE 'A%';  /* <-- easier */
**WHERE clauses: relational operators**

<table>
<thead>
<tr>
<th>Relational Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&lt;&gt; or !=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
</tbody>
</table>
## WHERE clauses: logical operators

<table>
<thead>
<tr>
<th>Logical Operator</th>
<th>Meaning</th>
<th>WHERE Clause Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AND</strong></td>
<td>True only if both conditions are true; false otherwise</td>
<td>WHERE State = ‘MN’ AND Gender = ‘M’</td>
</tr>
<tr>
<td><strong>OR</strong></td>
<td>True if either condition is true; false otherwise</td>
<td>WHERE LicenseStatusID = 1001 OR LicenseStatusID = 1002</td>
</tr>
<tr>
<td><strong>NOT</strong></td>
<td>Negate expression</td>
<td>WHERE NOT State = ‘NE’</td>
</tr>
<tr>
<td><strong>IN</strong></td>
<td>True if among set of discrete values listed</td>
<td>WHERE City IN(‘Arcata’, ‘Fortuna’, ‘Orick’)</td>
</tr>
<tr>
<td><strong>LIKE</strong></td>
<td>Wildcard expression allowing “don’t care” conditions</td>
<td>WHERE LastName LIKE ‘Mc%’</td>
</tr>
<tr>
<td><strong>BETWEEN ... AND ...</strong></td>
<td>True if within the value range, inclusive</td>
<td>WHERE SqFt BETWEEN 1500 AND 2000</td>
</tr>
</tbody>
</table>
Using the CASE structure

/* CASE statement useful when you have multiple updates To do in one table. Syntax:

UPDATE <table name> SET <column name> = CASE
WHEN <condition1> THEN <value1>
WHEN <condition2> THEN <value2>
... [ ELSE <default value>] END

Example: remove + and – values from grades: */

UPDATE grades SET grade = CASE
WHEN grade like ‘A%’ THEN ‘A ’
WHEN grade like ‘B%’ THEN ‘B ’
WHEN grade like ‘C%’ THEN ‘C ’
WHEN grade like ‘D%’ THEN ‘D ’
ELSE ‘F ’ END;
DELETE, TRUNCATE TABLE Statements

/* DELETE statement used to delete one or more rows From a table. Syntax:

DELETE FROM {<table name> | <view name>}
[WHERE <condition>];

Examples: */

DELETE FROM students WHERE id = ‘950123456’;

DELETE FROM grades;

/* faster way (no flashback, PERMANENT): */

TRUNCATE TABLE grades;
MERGE INTO Statement

/* MERGE statement allows you to merge row from a source Table into a target table, combining INSERT, UPDATE and DELETE into one statement. Syntax:

MERGE INTO <target table name>
USING <source table name or query>
ON (<condition>)
[WHEN MATCHED THEN UPDATE <set clause>
   DELETE <condition>]
[WHEN NOT MATCHED THEN INSERT <insert clause>]

Example: */

MERGE INTO students s USING transfer_students ts ON (s.id = ts.id) WHEN MATCHED THEN UPDATE SET (s.name = ts.name) WHEN NOT MATCHED THEN INSERT (s.id, s.name) VALUES (ts.id, ts.name);
**MERGE example**

### UpdateLicenseStatus source table
- 1004  Passed Away
- 1005  Expired--Fee Not Paid
- 1101  License Probationary
- 1105  License Pending

### LicenseStatus target table
- 1001  Licensed
- 1002  Licensed NBA
- 1003  Canceled Officer
- 1004  Passed Away
- 1005  Expired--Fee Not Paid
- 1006  Government Service
- 1007  Military Service
- 1008  Conditional Suspension
- 1009  Restricted
- 1010  Revoked
- 1011  Flag Suspended
- 1012  Voided
- 1013  Withheld Denied
- 1014  17520 FC Suspended
- 1015  11350.6 W and I Suspended
- 1016  Surrendered
- 1101  License Probationary
- 1105  License Pending

### LicenseStatus table following MERGE

- **1004**  Passed Away
- **1005**  Expired--Fee Not Paid

### MERGE INTO LicenseStatus LS

```sql
MERGE INTO LicenseStatus LS
USING UpdateLicenseStatus ULS
ON (LS-LicenseStatusID = ULS-LicenseStatusID)
WHEN MATCHED THEN
  UPDATE SET LS.StatusText = ULS.StatusText
WHEN NOT MATCHED THEN
  INSERT (LS-LicenseStatusID, LS.StatusText)
VALUES (ULS-LicenseStatusID, ULS.StatusText);
```
TRANSACTIONS

/* TRANSACTIONS allow for a set of SQL statements to execute as one logical statement (atomicity). Transactions in Oracle are not explicitly started. For the duration of the transaction, Oracle locks the affected rows (other users do not see your changes until the transaction is committed). only DML statements can be in a transaction – all others are implicitly committed.

Command Syntax:

SAVEPOINT <savepoint name>;
ROLLBACK [WORK] [TO [SAVEPOINT] <savepoint name>];
COMMIT;

See the book (p. 176) for examples of when implicit begin/end transactions occur. */
 TRANSACTION example

/* insert a row */
INSERT INTO students VALUES (student_id_seq.NEXTVAL, ‘Fred’, ‘Student’);

/* make a savepoint */
SAVEPOINT addedFred;

/* insert another row */
INSERT INTO student VALUES (’student_id_seq.NEXTVAL’, ‘George’, ‘Scholar’);

/* oops! undo that last one */
ROLLBACK TO addedFred;

/* commit fred */
COMMIT;
TRIGGERS

/* TRIGGERs are associated with a table, view, schema or the database. They fire when a certain event happens on one of those objects. Table triggers fire on DML statements. Useful in adding complex domain constraints, and useful in tracking modifications to table data.

Can be fired before, after or instead of the event. Triggers Are defined at the row-level or at the statement-level.

Syntax:

CREATE [OR REPLACE] TRIGGER <trigger name>
 { BEFORE | AFTER | INSTEAD OF } <trigger event> on
 <table name> [FOR EACH ROW [WHEN <condition>]]
 [DECLARE <declaration statements>]
 BEGIN <trigger body in PL/SQL code> END;
 */
TRIGGERs, continued

- Triggers cannot execute transaction statements.
- Trigger events are considered part of the transaction that caused the trigger.
- Any events in the trigger are committed or rolled back if the outer transaction is committed or rolled back.
- :OLD refers to pre-trigger data values, and :NEW refers to post-trigger data values.
TRIGGERS, cont.

<table>
<thead>
<tr>
<th>SQL Statement</th>
<th>Correlation Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INSERT</strong></td>
<td><strong>NEW</strong></td>
<td>Value supplied for the column in the statement that originated the transaction.</td>
</tr>
<tr>
<td></td>
<td><strong>OLD</strong></td>
<td>NULL</td>
</tr>
<tr>
<td><strong>UPDATE</strong></td>
<td><strong>NEW</strong></td>
<td>Value supplied for the column in the statement that originated the transaction.</td>
</tr>
<tr>
<td></td>
<td><strong>OLD</strong></td>
<td>Value of the column that was last committed into the table prior to the transaction.</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td><strong>NEW</strong></td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td><strong>OLD</strong></td>
<td>Value of the column that was last committed into the table before the transaction.</td>
</tr>
</tbody>
</table>
TRIGGER example

CREATE OR REPLACE TRIGGER students_bi_trg
BEFORE INSERT ON students FOR EACH ROW
BEGIN
    SELECT student_id_seq.NEXTVAL
    INTO :NEW.id FROM DUAL;
END;

/* this allows the following: */

INSERT INTO students (first_name, last_name) VALUES
    (‘Fred’, ‘Smith’);

/* to get the generated id: */

SELECT students_id_seq.CURRVAL FROM DUAL;
TRIGGERs for Auditing

/* first, create a table to store the audit records */

CREATE TABLE audit_students (  
event NVARCHAR2(6),  
before_after NVARCHAR2(6),  
username NVARCHAR(30),  
event_date DATE,  
student_id CHAR(9),  
first_name VARCHAR2(256),  
last_name VARCHAR2(256)  
);
TRIGGERs for Auditing

/* then, create the trigger to monitor the table */

CREATE OR REPLACE TRIGGER students_bu_trg
AFTER UPDATE ON students FOR EACH ROW
BEGIN
  INSERT INTO audit_students VALUES ('UPDATE',
   'BEFORE', user, SYSDATE, :OLD.id, :OLD.first_name,
   :OLD.last_name);
  INSERT INTO audit_students VALUES ('UPDATE',
   'AFTER', user, SYSDATE, :NEW.id, :NEW.first_name,
   :NEW.last_name);
END;
CREATE OR REPLACE TRIGGER students_aiu_trg
AFTER INSERT OR DELETE ON students
DECLARE
V_RowCount NUMBER;
BEGIN
SELECT COUNT(*) INTO V_RowCount FROM students;
DBMS_OUTPUT.PUT_LINE(V_RowCount || ' rows in students');
END;
/

/* notes: “set serveroutput on”, don’t forget the “/” at the end, and watch out for smart quotes... */