The University of Texas at Austin
McCombs School of Business

MIS 381 INTRO. TO DATABASE MANAGEMENT

Indexes and Sequences

Data Manipulation Language

Tayfun Keskin
Visiting Clinical Professor, The University of Texas at Austin, McCombs School of Business
Associate Teaching Professor, University of Washington Seattle, Foster School of Business

QUESTIONS

Any questions

before we begin …

AGENDA

Lecture	Indexes Sequences
Hands-On	Exercises
Looking Forward	Exam 1 Homework 2

REVIEW QUESTION

What is a good primary key?

QUESTION

What do you think will happen if you try to input a primary key identical to an existing cell?

WHAT ARE INDEXES?

- **Oracle schema objects (like tables) created to improve the performance of data access**
- Oracle provides several types of indexes
  - Default (most common) type is known as B-tree
- A composite index can be created on multiple columns
- Primary keys or unique constraints implicitly create indexes

QUESTION

Imagine a large dataset :

How many rows will Oracle DB will scan if you want to retrieve the sales performed yesterday?

INDEXES

- Why?

Speeds up:
- Joins
- Searches

- When do you NOT use an index?

Columns that are updated frequently

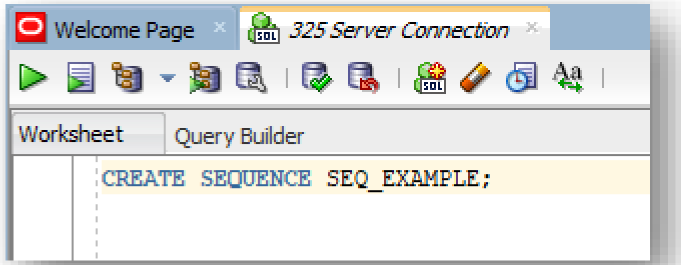
But why is this bad?

WHAT ARE SEQUENCES?

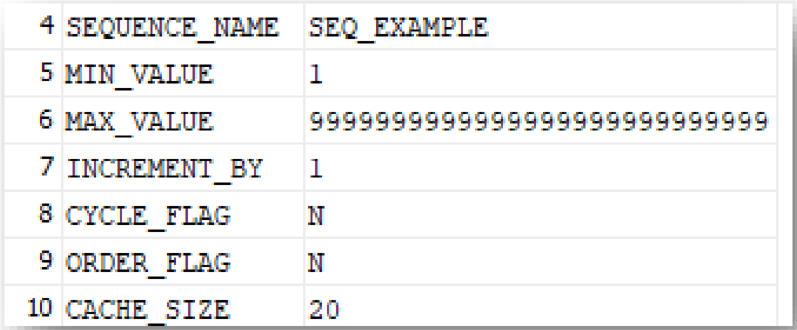
- Automatically generated sequence of integer values
- Sequences are independent Oracle schema objects (are not linked to any specific table in a database)
- You can reference the next value of a sequence when inserting data as many tables as you'd like (they can be used freely as a part of your insert statements)

SEQUENCE CHARACTERISTICS

- Min value
- Max value
- Increment by
- Cache
- Cycle or no cycle



```
CREATE SEQUENCE SEQ_EXAMPLE;
```



4	SEQUENCE_NAME	SEQ_EXAMPLE
5	MIN_VALUE	1
6	MAX_VALUE	9999999999999999999999999999
7	INCREMENT_BY	1
8	CYCLE_FLAG	N
9	ORDER_FLAG	N
10	CACHE_SIZE	20

QUESTION

Why would I ever need a sequence?

(an automatically incrementing column)

USE CASE FOR A SEQUENCE

- A database object that automatically generate a sequence of integer values
- Typically, it is used to generate a value for the primary key
- Much easier when adding new data

HANDS ON PRACTICE: OPEN ORACLE SQL DEVELOPER

IN-CLASS EXERCISE FILES

- ICE 1 – Drop / create script
- ICE 2 – Hands-on exercise on sequences and indexes
- ICE 3 – Self (team) practice on members table

PRACTICE FOR INDEXES

1. Start by discussing the following questions with your partner
    a. What would be likely fields to create an index on for the invoices table based on rules discussed?
    b. Should we create an index on vendor_id? If so, why?

2. Assuming we do want to create an index on invoices.vendor_id, write that syntax out and run it. Try to do this by reference only the syntax and not the examples. Confirm if your index was created successfully (i.e. without error).

3. Discuss if you should create an index on the invoice_date. If so, should we consider sorting the index in a specific way? Based on your discussion, create the index you think would be most valuable.

TOO MANY INDEXES?

- Creating too many indexes aren’t ideal
- The more indexes you create on your tables, the slower your insert, update, delete commands will be
  - Because each time you manipulate a row in your table, indexes associated with that table will also have to be updated as well
- Only frequently queried columns should be indexed

LOOKING FORWARD

We covered Chapter 10

What's next?

- DML

We will start Chapter 7

Exam 1

Homework 2

THANK YOU

DATA MANIPULATION LANGUAGE

- Insert
- Update
- Delete

REVIEW QUESTION

Which command do we use to start a SQL query?

```
SELECT *
FROM Employees
```

SQL STATEMENTS (SO FAR)

- **Queries:** Retrieving data from database tables

```
SELECT * FROM Employees;
```

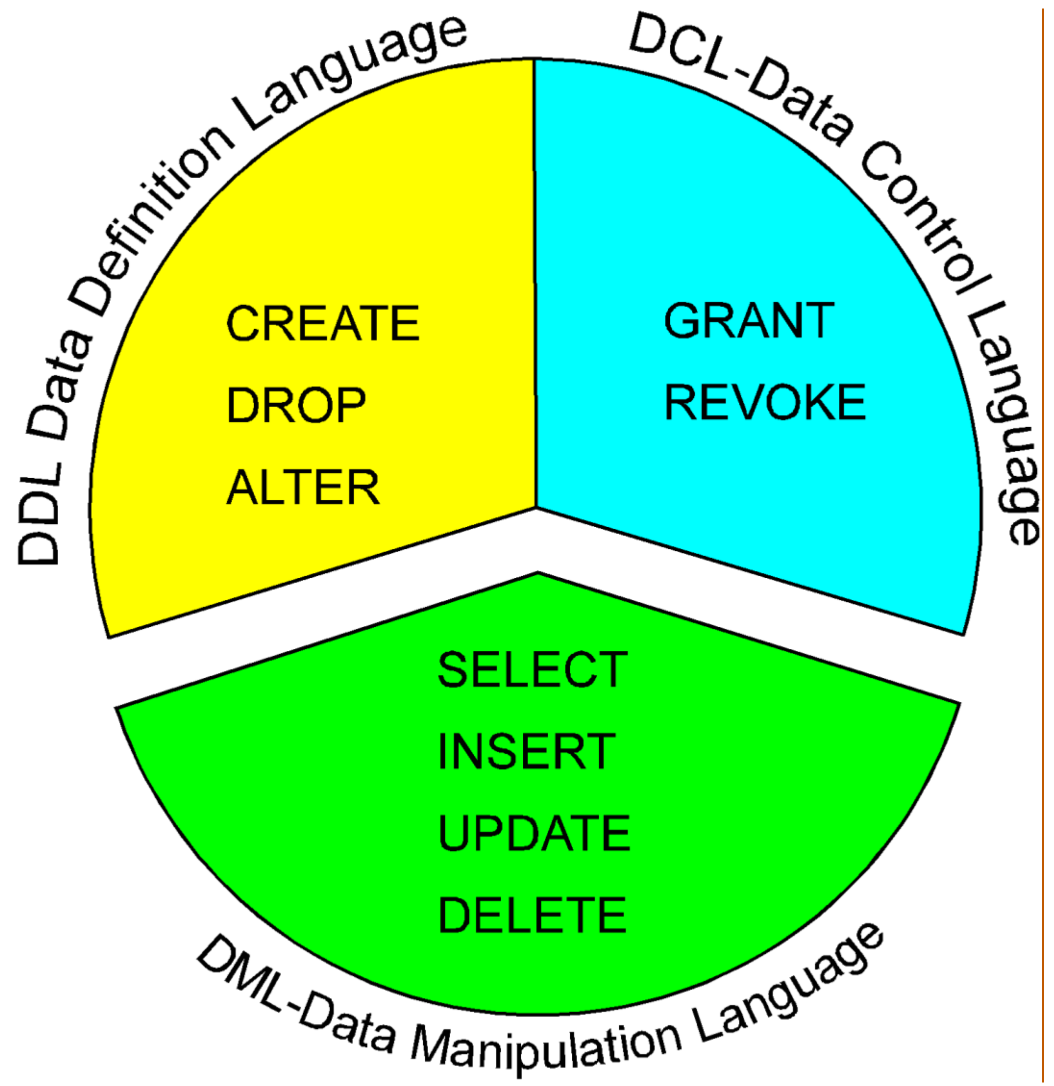
- **DDL:** Creating/defining new database objects

```
CREATE TABLE Employees
(EmpID Number, FName Varchar2(20), LName Varchar2(20));
```

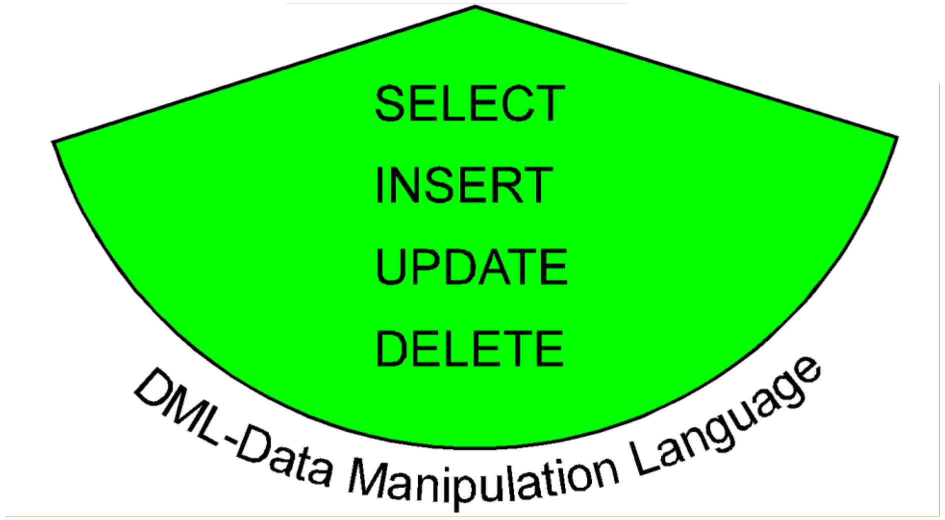
- **DML:** Modifying/manipulating existing data in your database

```
INSERT INTO Employees
VALUES (1, ‘John’, ‘Smith’);
```

SQL OVERVIEW


DDL Data Definition Language
CREATE
DROP
ALTER
DCL-Data Control Language
GRANT
REVOKE
SELECT
INSERT
UPDATE
DELETE
DML-Data Manipulation Language

SQL OVERVIEW



IMPORTANT TERMS

- **Transaction:** autonomous unit of work where data is modified
  - Any DML command will implicitly create a transaction
- **Commit:** make the changes persistent (closes the transaction)
  - Data isn't officially saved to table for other users to see until you commit
  - You'll be able to see data you insert before a commit
- **Rollback:** undo the changes
  - If you rollback before you commit, data will not be saved to table
  - Rollback will have no effect right after commit

THANK YOU

BACKUP SLIDES

PART 1

ICE 2 – Chapter 10 (Indexes and Sequences)

The syntax of the CREATE SEQUENCE statement

```
CREATE SEQUENCE sequence_name
  [START WITH starting_integer]
  [INCREMENT BY increment_integer]
  [{MINVALUE minimum_integer | NOMINVALUE}]
  [{MAXVALUE maximum_integer | NOMAXVALUE}]
  [{CYCLE|NOCYCLE}]
  [{CACHE cache_size|NOCACHE}]
  [{ORDER|NOORDER}]
```

A statement that creates a sequence

```
CREATE SEQUENCE vendor_id_seq
```

A statement that specifies a starting integer

```
CREATE SEQUENCE vendor_id_seq
  START WITH 124
```

A statement that specifies all parameters

```
CREATE SEQUENCE test_seq
  START WITH 100 INCREMENT BY 10
  MINVALUE 0 MAXVALUE 1000000
  CYCLE CACHE 100;
```

Let’s Clarify:

- ***Cycle*** = return to zero after seq hits the max value
- ***Cache*** = Saves the next set of numbers in the seq in memory to speed up performance. Default cache is 20.
- ***Order*** = Guarantees the numbers generated in order of request. *Only used in unique application design situations*

A statement that creates a sequence

```
CREATE SEQUENCE vendor_id_seq
```

A statement that specifies a starting integer

```
CREATE SEQUENCE vendor_id_seq
  START WITH 124
```

A statement that specifies all parameters

```
CREATE SEQUENCE test_seq
  START WITH 100 INCREMENT BY 10
  MINVALUE 0 MAXVALUE 1000000
  CYCLE CACHE 100;
```

Practice:

1. Create a sequence called ***member_id_seq*** that starts at 10 and increments by 1.

4. Run test INSERT statements and confirm sequence was created correctly and works

6. Update the DDL provided to default the member_id column to the next value of the newly created sequence

7. Run 2nd test INSERT statements and then confirm member_id defaults correctly

Example: How to use a sequence as the default value

```
CREATE TABLE invoices
(
  invoice_id     NUMBER      DEFAULT invoice_id_seq.NEXTVAL PRIMARY KEY,
  invoice_date   DATE         NOT NULL
)
```

The syntax of the ALTER SEQUENCE statement

```
ALTER SEQUENCE sequence_name
  [sequence_attributes]
```

A statement that alters a sequence

```
ALTER SEQUENCE test_seq
  INCREMENT BY 9
  MINVALUE 99 MAXVALUE 999999
  NOCYCLE CACHE 9 NOORDER;
```

A statement that drops a sequence

```
DROP SEQUENCE test_seq;
```

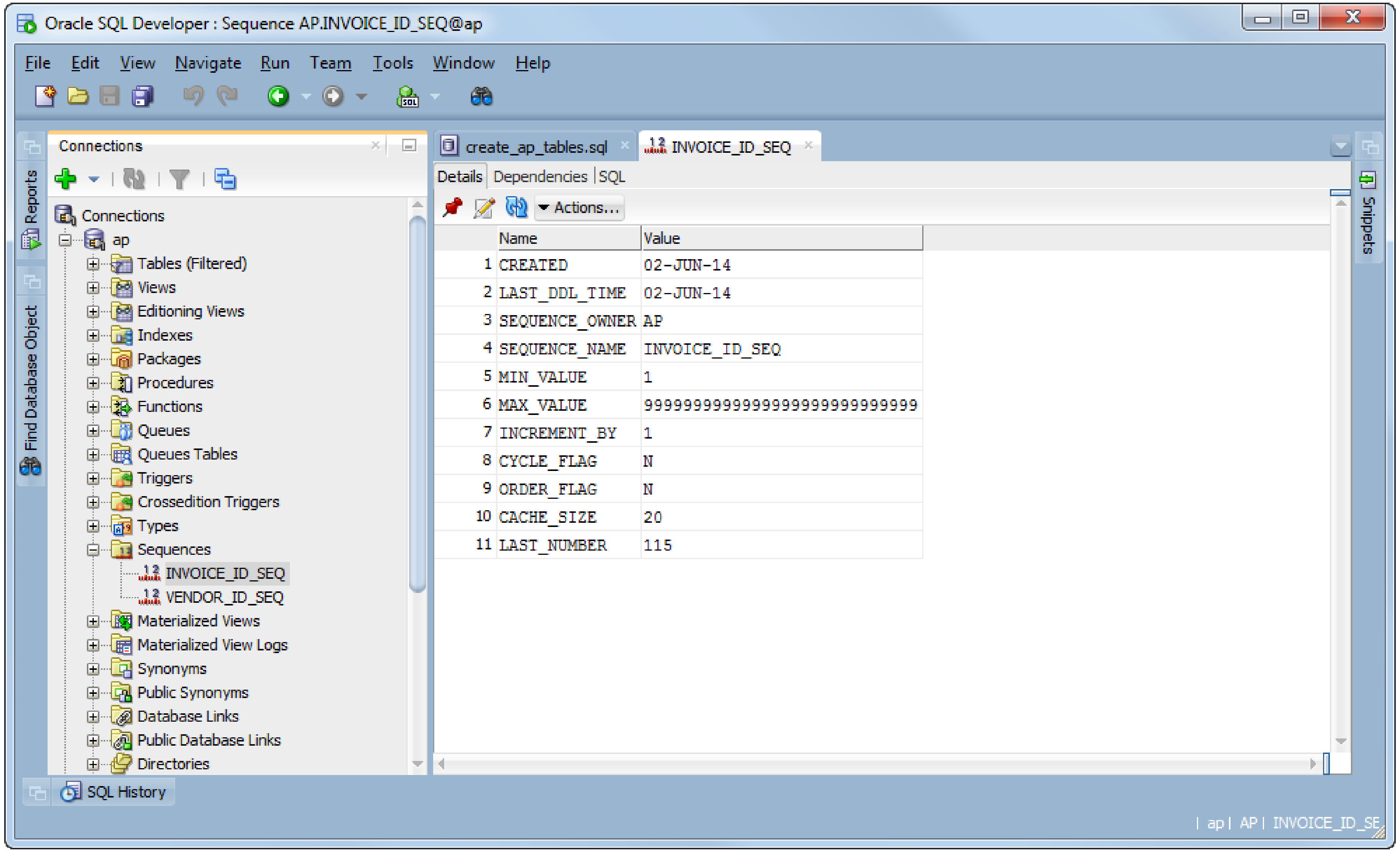
Practice

5. Update member_id_seq to increment by 10 instead of 1

6. Rerun 2nd test INSERTS and SELECT to see if they still worked.

7. Drop member_id_seq

The sequences for the AP schema



	Name	Value
1	CREATED	02-JUN-14
2	LAST_DDL_TIME	02-JUN-14
3	SEQUENCE_OWNER	AP
4	SEQUENCE_NAME	INVOICE_ID_SEQ
5	MIN_VALUE	1
6	MAX_VALUE	9999999999999999999999999999
7	INCREMENT_BY	1
8	CYCLE_FLAG	N
9	ORDER_FLAG	N
10	CACHE_SIZE	20
11	LAST_NUMBER	115

How to work with Indexes

- Speeds up joins and searches. How?
- By default, Oracle creates an index on each Primary Key
- Best to add Indexes on columns:
  - ❑ Frequently used in joins (i.e. Foreign Keys especially)
  - ❑ Frequently used in searches
  - ❑ Has a UNIQUE integrity constraint
  - ❑ Not updated a lot - indexes slow down insert, update, deletes

- FAQ: Is indexing everything bad? Who typically makes indexes?

The syntax of the CREATE INDEX statement

```
CREATE [UNIQUE] INDEX index_name
  ON table_name (column_name_1 [ASC|DESC]
              [, column_name_2 [ASC|DESC]]...)
```

TIP: Use a standard naming convention for readability. e.g. table_column_ix

A statement that creates an index based on a single column

```
CREATE INDEX invoices_vendor_id_ix
  ON invoices (vendor_id);
```

A statement that creates an index based on two columns

```
CREATE INDEX invoices_vendor_id_inv_no_ix
  ON invoices (vendor_id, invoice_number);
```

A statement that creates a unique index

```
CREATE UNIQUE INDEX vendors_vendor_phone_ix
  ON vendors (vendor_phone);
```

A statement that creates an index that's sorted in descending order

```
CREATE INDEX invoices_invoice_total_ix
  ON invoices (invoice_total DESC);
```

Index Syntax using *table_field_ix* naming standard

```
-- Create the indexes
CREATE INDEX vendors_terms_id_ix
  ON vendors (default_terms_id);
CREATE INDEX vendors_account_number_ix
  ON vendors (default_account_number);
```

Practice:

8. What would be likely fields to create an index on for the invoice table?

9. Create an index on vendor_id. Why?

10. Create an index on invoice_date why? Should we consider sorting? Create index.

A statement that creates a function-based index

```
CREATE INDEX vendors_vendor_name_upper_ix
  ON vendors (UPPER(vendor_name));
```

Another statement for a function-based index

```
CREATE INDEX invoices_balance_due_ix
  ON invoices (invoice_total - payment_total - credit_total DESC);
```

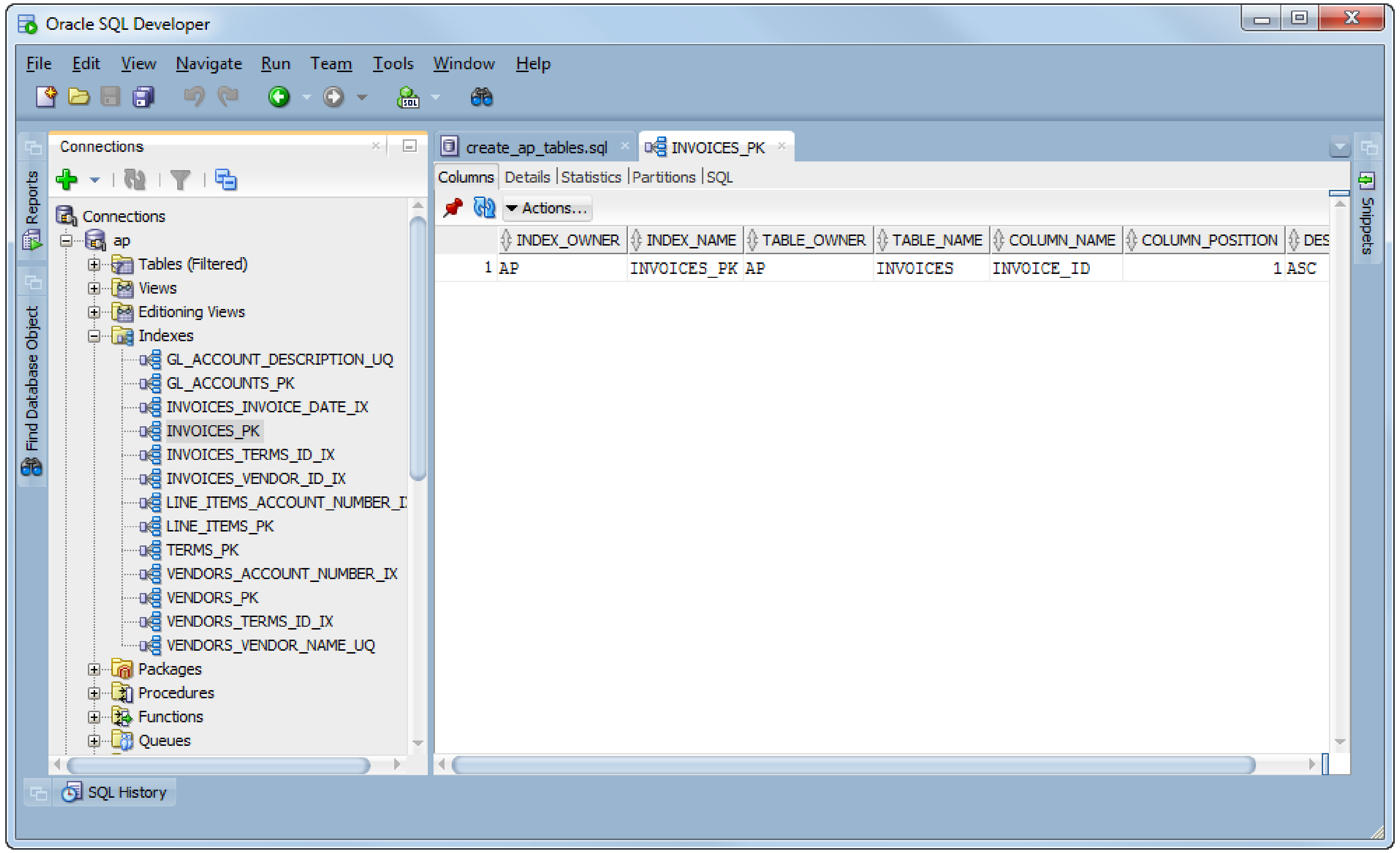
How to enable function-based indexes

```
CONNECT system/system;
ALTER SYSTEM SET QUERY_REWRITE_ENABLED=TRUE;
```

A statement that drops an index

```
DROP INDEX vendors_vendor_state_ix
```

The indexes for the AP schema



Connections
- ap
  - Tables (Filtered)
  - Views
  - Editioning Views
  - Indexes
    - GL_ACCOUNT_DESCRIPTION_UQ
    - GL_ACCOUNTS_PK
    - INVOICES_INVOICE_DATE_IX
    - INVOICES_PK
    - INVOICES_TERMS_ID_IX
    - INVOICES_VENDOR_ID_IX
    - LINE_ITEMS_ACCOUNT_NUMBER_I
    - LINE_ITEMS_PK
    - TERMS_PK
    - VENDORS_ACCOUNT_NUMBER_IX
    - VENDORS_PK
    - VENDORS_TERMS_ID_IX
    - VENDORS_VENDOR_NAME_UQ
  - Packages
  - Procedures
  - Functions
  - Queues

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DES
1	AP	INVOICES_PK	AP	INVOICES	INVOICE_ID	1	ASC

PART 2

Chapter 7

DML – Insert, Update, Delete

INSERT statements add data to tables

```
INSERT INTO invoices
VALUES (115, 97, '456789', '01-AUG-14', 8344.50, 0, 0, 1, '31-AUG-14', NULL)
```

The response from the system

```
1 rows inserted
```

A COMMIT statement that commits the changes

```
COMMIT
```

The response from the system

```
COMMIT succeeded
```

A statement that rolls back the changes

```
ROLLBACK
```

The response from the system

```
ROLLBACK succeeded
```

Terms you should know

- Transaction
- Commit
    - NOTE: data isn’t officially saved to table for other users to see until you Commit.
    - NOTE: You’ll be able to see data you insert before a commit
- Rollback
    - NOTE: If you rollback before you commit, data will not be saved to table

The syntax of the CREATE TABLE AS statement

```
CREATE TABLE table_name AS
SELECT select_list
FROM table_source
[WHERE search_condition]
```

Example

```
CREATE TABLE invoices_copy AS
SELECT *
FROM invoices
```

When would this be useful?

- Avoid touching production tables in a live DB
- Let’s you practice on non-production tables.
- NOTE: It’s not likely you’ll be creating “test” tables in a Production environment

A statement that creates a partial copy of the Invoices table

```
CREATE TABLE old_invoices AS
SELECT *
FROM invoices
WHERE invoice_total - payment_total - credit_total = 0
```

A statement that creates a table with summary rows from the Invoices table

```
CREATE TABLE vendor_balances AS
SELECT vendor_id, SUM(invoice_total) AS sum_of_invoices
FROM invoices
WHERE (invoice_total - payment_total - credit_total) <> 0
GROUP BY vendor_id
```

A statement that deletes a table

```
DROP TABLE old_invoices
```

Warning

- When you use the SELECT statement to create a table, only the column definitions and data are copied.
- Definitions of primary keys, foreign keys, indexes, default values, and so on <u>are not included</u> in the new table.

INSERT syntax for inserting a single row

```
INSERT INTO table_name [(column_list)]
    VALUES (value_1 [, expression_2]...)
```

An INSERT statement that adds a new row without using a column list

```
INSERT INTO invoices
VALUES (115, 97, '456789', '01-AUG-14', 8344.50, 0, 0, 1, '31-AUG-14', NULL)

(1 rows inserted)
```

An INSERT statement that adds the new row using a column list

```
INSERT INTO invoices
    (invoice_id, vendor_id, invoice_number, invoice_total, payment_total,
     credit_total, terms_id, invoice_date, invoice_due_date)
VALUES
    (115, 97, '456789', 8344.50, 0,
      0, 1,'01-AUG-14', '31-AUG-14')
(1 rows inserted)
```

NOTE: It’s okay to continue on 2 lines

The definition of the Color_Sample table

Column name	Data Type	Not Null	Default Value
color_id	NUMBER	Yes	
color_number	NUMBER	Yes	0
color_name	VARCHAR2		

5 INSERT statements examples (NULL & Default)

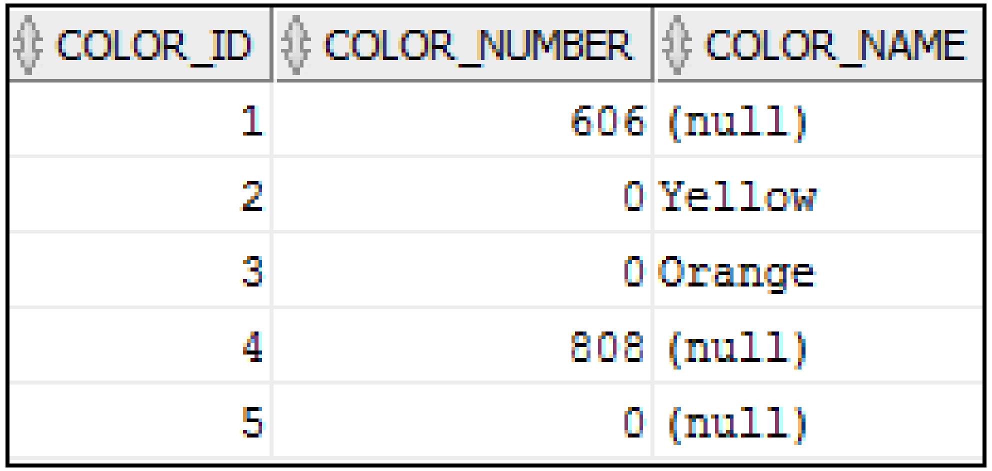
```
INSERT INTO color_sample (color_id, color_number)
VALUES (1, 606)

INSERT INTO color_sample (color_id, color_name)
VALUES (2, 'Yellow')

INSERT INTO color_sample
VALUES (3, DEFAULT, 'Orange')

INSERT INTO color_sample
VALUES (4, 808, NULL)

INSERT INTO color_sample
VALUES (5, DEFAULT, NULL)
```



COLOR_ID	COLOR_NUMBER	COLOR_NAME
1	606	(null)
2	0	Yellow
3	0	Orange
4	808	(null)
5	0	(null)

The Color_Sample table after the rows are inserted

The syntax of the INSERT statement for inserting rows selected from another table

```
INSERT [INTO] table_name [(column_list)]
SELECT column_list
FROM table_source
[WHERE search_condition]
```

An INSERT statement that inserts paid invoices in the Invoices table into the Invoice_Archive table

```
INSERT INTO invoice_archive
SELECT *
FROM invoices
WHERE invoice_total - payment_total - credit_total = 0
(74 rows inserted)
```

Commonly used when copying data from one table to another that have similar data structure.

The same INSERT statement with a column list

```
INSERT INTO invoice_archive
    (invoice_id, vendor_id, invoice_number,
     invoice_total, credit_total,
     payment_total, terms_id, invoice_date,
     invoice_due_date)
SELECT
    invoice_id, vendor_id, invoice_number, invoice_total,
    credit_total, payment_total, terms_id,
    invoice_date, invoice_due_date
FROM invoices
WHERE invoice_total - payment_total - credit_total = 0
(74 rows inserted)
```

Commonly used when copying data from one table to another that have similar data structure.

The syntax of the UPDATE statement

```
UPDATE table_name
SET column_name_1 = expression_1 [, column_name_2 = expression_2]...
[WHERE search_condition]
```

An UPDATE statement that assigns new values to two columns of a single row in the Invoices table

```
UPDATE invoices
SET payment_date = '21-SEP-14',
    payment_total = 19351.18
WHERE invoice_number = '97/522'
```

```
(1 rows updated)
```

An UPDATE statement that assigns a new value to one column of all invoices for a vendor

```
UPDATE invoices
SET terms_id = 1
WHERE vendor_id = 95
```

```
(6 rows updated)
```

An UPDATE statement that uses an arithmetic expression to assign a value to a column

```
UPDATE invoices
SET credit_total = credit_total + 100
WHERE invoice_number = '97/522'
```

```
(1 rows updated)
```

Warning

- If you omit the WHERE clause, all rows in the table will be updated.

An UPDATE statement that assigns the maximum due date in the Invoices table to a specific invoice

```
UPDATE invoices
SET credit_total = credit_total + 100,
    invoice_due_date =
        (SELECT MAX(invoice_due_date)
         FROM invoices)
WHERE invoice_number = '97/522'
```

```
(1 rows updated)
```

An UPDATE statement that updates all invoices for a vendor based on the vendor’s name

```
UPDATE invoices
SET terms_id = 1
WHERE vendor_id =
   (SELECT vendor_id
    FROM vendors
    WHERE vendor_name = 'Pacific Bell')
```

```
(6 rows updated)
```

An UPDATE statement that changes the terms of all invoices for vendors in three states

```
UPDATE invoices
SET terms_id = 1
WHERE vendor_id IN
   (SELECT vendor_id
    FROM vendors
    WHERE vendor_state IN ('CA', 'AZ', 'NV'))
```

```
(51 rows updated)
```

The syntax of the DELETE statement

```
DELETE [FROM] table_name
[WHERE search_condition]
```

A DELETE statement that deletes one row

```
DELETE FROM invoice_line_items
WHERE invoice_id = 100 AND invoice_sequence = 1
```

```
(1 rows deleted)
```

A DELETE statement that deletes four rows

```
DELETE FROM invoice_line_items
WHERE invoice_id = 100
```

```
(4 rows deleted)
```

A DELETE statement that uses a subquery to delete all invoice line items for a vendor

```
DELETE FROM invoice_line_items
WHERE invoice_id IN
    (SELECT invoice_id
    FROM invoices
    WHERE vendor_id = 115)
```

```
(4 rows deleted)
```

Warning

- If you omit the WHERE clause from a DELETE statement, all the rows in the table will be deleted.