

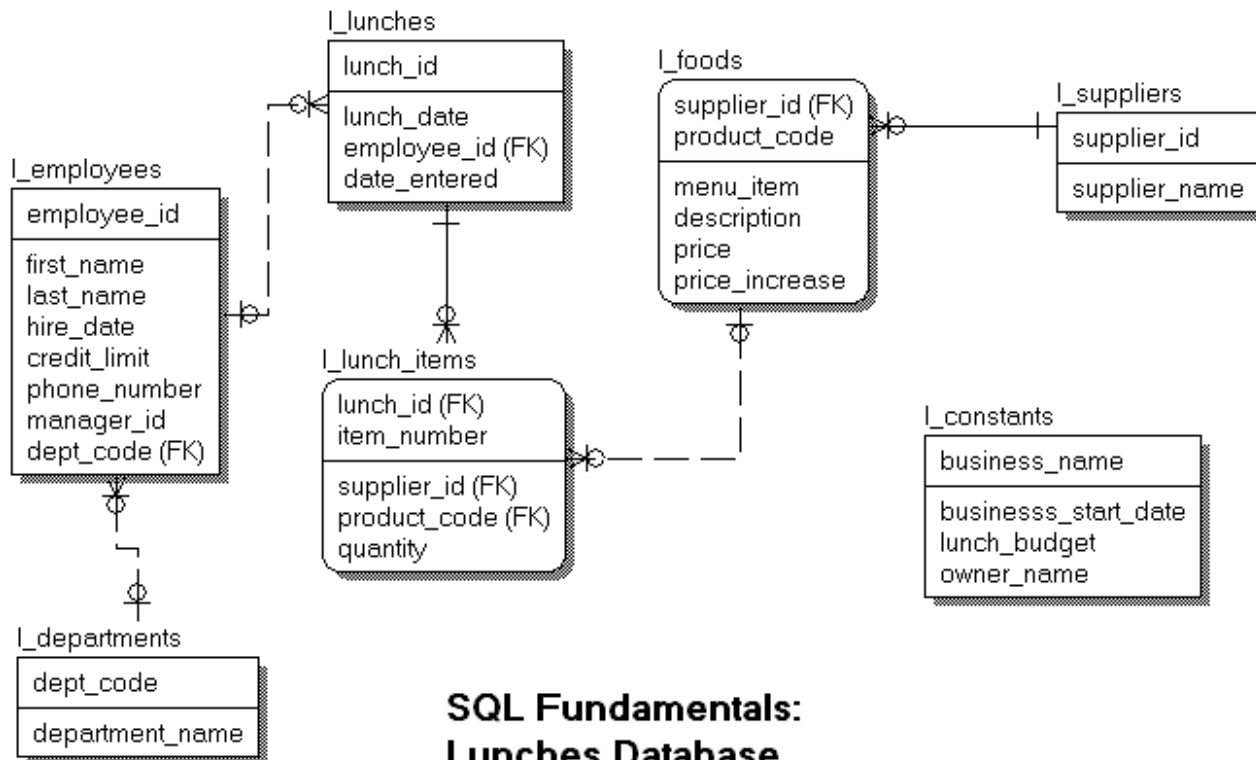
SQL:



Insert
Update
Delete

Before Inserting New Data

- Must know the structure of a table before you can insert data into it.
- You could look at the schema (model):



But what if you don't have the model?

Ways to Insert Data

- Use the **SQL INSERT** statement
- Use a utility program to load the data from a text file:
 - Oracle: **SQL Loader**
 - Microsoft: Bulk Copy
 - DB2: LOAD utility
 - MySQL: **“LOAD DATA INFILE ...”**
 - Access: Import function
 - PHPMyAdmin: Import tab

Every database has a "loader", but the name, syntax, and details will always be different.

INSERT: Syntax

**INSERT INTO table_name [(column-list)]
VALUES (value-list);**

General rules:

- Insert one row at a time
- You should know the order and data type of all the columns before you try to insert data.
- The syntax allows you to include data for some or ALL columns
- If you do not provide data for ALL columns, you must list the column name as well as the value.

Insert Command – General Rules

INSERT INTO I_foods VALUES

('ASP', 'FS', 1, 'FRESH SALAD', 2, 0.25);

- Remember to use **single quotes** around TEXT and DATES.
- Numbers are left unquoted.
- Dollar signs are not stored in the database and not included in the INSERT statement.
- Notice that this format does not list column names and *we have supplied data for every column.*


Insert Command – Two Formats

- 1) A data value must be supplied (or null) for every column if columns are not named:

```
INSERT INTO l_foods -- notice no column names here  
VALUES ('ARR', 'AP', 11, 'APPLE PIE', 1.50, null);
```

- 2) If columns are named, only the values for the named columns need to appear in the “values” clause.

```
INSERT INTO l_foods  
(product_code, description, supplier_id, price)  
VALUES('AP', 'APPLE PIE', 'ARR', 1.60);
```



INSERT INTO...SELECT FROM

- It is also possible to insert rows of data into a table by running a SELECT command to get those rows. Two methods:

(1) First Method:

```
INSERT INTO table_name  
{select statement};
```

Before you can use this command, you must first create the l_foods_copy table.

```
INSERT INTO l_foods_copy      -- This table has 6 columns  
SELECT 'ARR', product_code, menu_item, description,  
null, null      -- so SELECT stmt must have 6 columns  
FROM l_foods  
WHERE supplier_id = 'ASP';
```

What's going on with 'ARR'?

Number of columns SELECTED must match number of columns INSERTED

Copy Data

```
create table I_foods_new (  
  supplier_id varchar(3) not null,  
  product_code varchar(3) not null,  
  menu_item integer not null,  
  description varchar(24) not null,  
  price decimal(5,2) null,  
  price_increase decimal(5,2) null);
```

I_foods

Results Explain Describe Saved SQL History

SUPPLIER_ID	PRODUCT_CODE	MENU_ITEM	DESCRIPTION	PRICE	PRICE_INCREASE
ASP	FS	1	FRESH SALAD	2	.25
ASP	SP	2	SOUP OF THE DAY	1.5	-
ASP	SW	3	SANDWICH	3.5	.4

3 rows returned in 0.09 seconds [Download](#)

```
INSERT INTO I_foods_new  
SELECT 'ARR', product_code, menu_item, description, null, null  
FROM I_foods  
WHERE supplier_id = 'ASP';
```

I_foods_new

SUPPLIER_ID	PRODUCT_CODE	MENU_ITEM	DESCRIPTION	PRICE	PRICE_INCREASE
ARR	FS	1	FRESH SALAD	-	-
ARR	SP	2	SOUP OF THE DAY	-	-
ARR	SW	3	SANDWICH	-	-

3 rows returned in 0.00 seconds [Download](#)

INSERT - Safest Way



- You will get the most reliable results if you actually name all the columns you are inserting into. Why?
 - Sometimes the backend will actually rearrange the order of columns and you won't realize this until you can't insert data.
 - You can actually skip some columns like autoincrementing key, or columns with null value.
 - You can insert the columns in any order you like.

INSERT INTO...SELECT FROM

(2) Second Method

```
INSERT INTO table_name {list_of_columns}  
    {select_statement};
```

-- 4 columns to INSERT

```
INSERT INTO I_foods_copy (supplier_id,  
    product_code, menu_item, description)  
SELECT 'ARR', product_code, menu_item,  
description  
FROM I_foods  
WHERE supplier_id = 'ASP';
```

-- 4 columns SELECTED

Multi-row INSERT ???

- MySQL, PostgreSQL, MSSQL, and DB2 have a feature that allows multiple rows of data to be inserted in one statement:

```
INSERT INTO I_suppliers VALUES
```

```
(900, 'Hannafords'),
```

```
(901, 'Shaws'),
```

```
(902, 'Price Chopper');
```

- Oracle can sort of do this: (not as nice)

```
INSERT ALL
```

```
  INTO I_suppliers VALUES (900, 'Hannafords')
```

```
  INTO I_suppliers VALUES (901, 'Shaws')
```

```
  INTO I_suppliers VALUES (902, 'Price Chopper')
```

```
SELECT * FROM dual;
```

Updating Columns – SET clause

```
UPDATE table_name  
SET col1 = value1,  
      col2 = value2,  
      column_n = value_n  
WHERE condition;
```

```
UPDATE l_foods  
SET price = price * 1.10  
WHERE supplier_id in ('JBR', 'FRV');
```

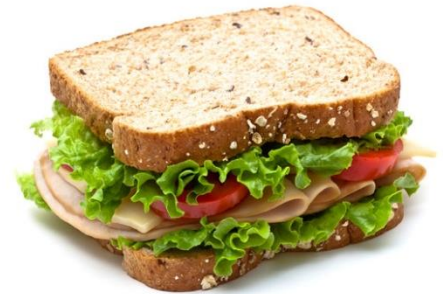
- BE CAREFUL, without **WHERE clause**, you will be updating EVERY row in the table!!
- To update a single row, specify primary key in **WHERE** clause.
- Values can be fixed value, a function, an expression, even a subquery.

Update with a Subquery

```
UPDATE I_foods  
SET price =  
    (SELECT price from I_foods  
     WHERE supplier_id = 'ASP' AND  
           product_code = 'SW')  
WHERE description = 'FRESH SALAD';
```

What does this query do?

FYI: Sandwich has supplier_id = 'ASP' & product_code = 'SW'



Deleting Rows

- Syntax:
DELETE FROM table_name
WHERE {condition};
- Example:
DELETE I_foods
WHERE supplier_id in ('CBC','JBR');
- As with Update, **WHERE** clause is critical!!!



Deleting Rows

- What does this command do?

DELETE FROM I_employees;

- Delete can get a little more complicated.....

```
DELETE FROM  
(SELECT d.*  
FROM I_departments d  
LEFT OUTER JOIN I_employees e  
ON d.dept_code = e.dept_code  
WHERE e.dept_code IS NULL);
```

Don't worry about the details of this query. We will revisit deletes again when we study JOINS.

Constraints on Update, Insert, Delete

- These actions may fail due to constraints.
For example:
 - Data Type may not match
 - May be inserting a value that is a foreign key, but the parent key does not yet exist.
 - No nulls in primary key
 - Primary key must be unique
 - MANY, MANY more constraints possible
 - We will discuss this more when we study referential integrity.



What can we do if we accidentally change or delete data?

SQL Transactions: the short story

- In a transaction, all statements are executed as a single “Unit of Work”.
- Use transactions when all statements must either succeed or none succeed.
- We typically don’t use transactions for `SELECT` statements. They are used when we are inserting, updating, or deleting data and we need to execute a group of changes simultaneously.

Commit, Rollback, Autocommit

- Initially changes to a table are temporary.
- **Commit** = save changes
- **Rollback** = restore table (undo changes)
- Commit is automatic (**autocommit**) when:
 - exiting Oracle
 - when creating a new table or database object.
 - **The checkbox is set in Application Express.**
 - **By default in MySQL.**
- Also possible to set the autocommit value:

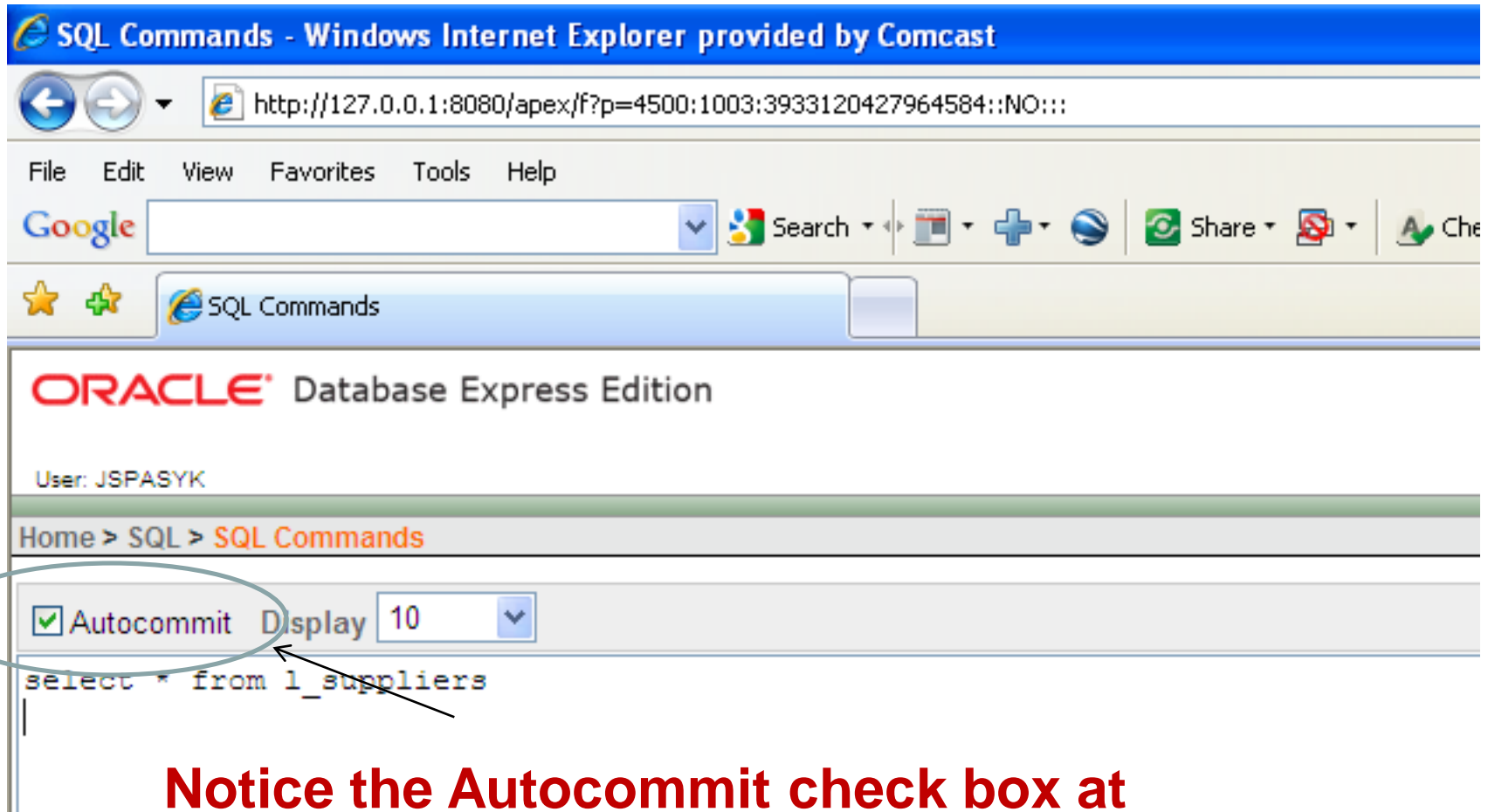
Oracle

Set autocommit on;
Set autocommit off;

MySQL

set autocommit = 1;
set autocommit = 0;

Autocommit in Oracle XE



The screenshot shows a web browser window titled "SQL Commands - Windows Internet Explorer provided by Comcast". The address bar shows the URL "http://127.0.0.1:8080/apex/f?p=4500:1003:3933120427964584::NO:::". The browser's menu bar includes "File", "Edit", "View", "Favorites", "Tools", and "Help". Below the menu bar is a search bar with the "Google" logo and a search button. The browser's address bar shows "SQL Commands".

The main content area displays the "ORACLE Database Express Edition" logo and the user "User: JSPASYK". The breadcrumb navigation shows "Home > SQL > SQL Commands". Below the breadcrumb is a control panel with a checked "Autocommit" checkbox, a "Display" label, and a dropdown menu showing "10". Below the control panel is a text area containing the SQL command "select * from l_suppliers".

Notice the Autocommit check box at the top of the SQL pane.

Oracle Transaction Example

COMMIT; -- end prior transaction; start a new one.

UPDATE checking_act

SET balance = balance + 200

WHERE account_num = 812384;

UPDATE savings_act

SET balance = balance – 200

WHERE account_num = 812385;

COMMIT; -- transaction ends here

Transaction Example - Rollback

COMMIT; -- transaction starts here

UPDATE checking_act

SET balance = balance + 200

WHERE account_num = 812384;

UPDATE savings_act

SET balance = balance – 200

WHERE account_num = 812385;

ROLLBACK; -- changes are not saved,
transaction is complete

Autocommit in MySQL

- Set autocommit = 1 means commit every statement. This is default for INNODB tables.
- Set autocommit = 0 means that you will be using transaction mode.
- However, most people don't set the value of autocommit. Instead, they break out of autocommit by issuing a:
- **START TRANSACTION** command
- Then they end the transaction with **COMMIT** or **ROLLBACK**.

MySQLTransaction Example

START TRANSACTION;

UPDATE checking_act

SET balance = balance + 200

WHERE account_num = 812384;

UPDATE savings_act

SET balance = balance – 200

WHERE account_num = 812385;

COMMIT; **(or ROLLBACK)**